

Parallel Maximum Weight Bipartite Matching Algorithms for Scheduling in Input-Queued Switches

Morteza Fayyazi
David Kaeli
Waleed Meleis

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
mfayyazi, kaeli, meleis@ece.neu.edu

Abstract

An input-queued switch with virtual output queuing is able to provide a maximum throughput of 100% in the supporting more sophisticated scheduling strategies. Switch scheduling can be cast as a maximum flow problem. In this paper we propose a maximum weight bipartite matching (MWBM) scheduling algorithm for input-queued switches. Our goal is to provide 100% throughput while maintaining fairness and stability. Our algorithm provides sublinear parallel run time complexity using a polynomial number of processing elements. We are able to obtain the MWBM for a time slot in sublinear time by using the matching produced in the previous time slot based on the observation that in input-queued cell-based switches, the weight of edges changes very little during successive time slots. To the best of our knowledge, our algorithm outperforms all previously proposed MWBM scheduling algorithms proposed for input-queued switches. We also describe a linear time complexity MWBM algorithm for a general bipartite graph which outperforms the best known sublinear MWBM algorithm for any bipartite graph with less than 10^{15} number of nodes.

1. Introduction

Our ability to build high-performance networks is limited by availability of high speed switching. The performance bottleneck of the network has shifted from the channel transmission speed to the processing speed at the switching nodes. Rapid advances have been made in optical communications, resulting in large scale deployment of optical transmission technology. Terabit per second technology is

becoming readily available. In addition to the demand for high speed throughput, a switch architecture is required to provide high reliability and fault-tolerance, fairness and stability, scalability, QoS support and a low packet switch latency. Many switch architectures have been proposed in the literature, though any single architecture can only address a subset of these issues [4, 5, 6, 7, 8, 9].

In this paper we propose a *maximum weight bipartite matching* scheduling algorithm (MWBM) that achieves 100% throughput¹, fairness and stability for admissible incoming traffic, stability is defined as:

$Q(i, j) < \infty$, where $Q(i, j)$ is the length of queue j in memory of input i , and

admissible traffic is defined as:

$\Sigma(\lambda(i, j)) < 1$ for all input i , and $\Sigma(\lambda(i, j)) < 1$ for all output j , where $\lambda(i, j)$ is the packet rate from input i to output j .

Using our proposed hardware implementation of this algorithm, MWBM runs in $O(\sqrt{n} \log^2 n)$ time using $O(n^3)$ processing elements.

A packet switch needs to be able to buffer packets because of contention for output links. Buffering can occur at input ports, output ports, the shared memory interface or the switching fabric [1, 2, 3]. Over the past few years, input-queued switch architectures have become dominant in high speed switching [12]. This growth in popularity is mainly due to the fact that input-queued switches place lower demands on memory bandwidth as compared to output-queued and shared-memory architectures. An input-queued switch with virtual output queuing (VOQ) [10, 11] is able to provide a maximum throughput of 100% when

¹100% throughput for an input queued switch means we obtain the same throughput as we would for an output queued switch assuming the same input workload.

utilizing more sophisticated scheduling strategies.

Figure 1 shows an architecture for input-queued switches. The elements in the left side of the Figure are inbound traffic nodes. Each ingress packet is examined by the Decision Making Machine (DMM) component. The DMM looks up the destination address of each packet in an address table. In Figure 1, M_i is a memory block. Memory blocks implement virtual output queuing. Memory blocks include several FIFO queues for each output. These queues can be used to support QoS. The egress scheduler retrieves packets from the queues and sends them out to the proper output port. The egress scheduler uses a crossbar switch for interconnection. McKeown et al. [12] showed that an input-queued switch that maintains a separate FIFO queue for each output at each input can achieve 100% throughput for both uniform and nonuniform independent arrivals when a maximum weight matching algorithm is used. However, if a maximum size matching (MSM) algorithm is used to schedule cells, a throughput of 100% may not be possible when arrivals are nonuniform. Dai and Prabhakar [13] showed that a maximum weight matching algorithm for connecting inputs and outputs can deliver 100% throughput when the input traffic pattern satisfies the strong law of large numbers and does not oversubscribe any input or any output. However, a maximal matching algorithm can also deliver a throughput of 100% for combined input and output-queued (CIOQ) switches that run at a speedup of 2 [14]².

This paper is focused on the design of a scheduler for high throughput switches. For an $n \times n$ input-queued switch it is well known that a crossbar-based switch interconnect scheduling problem maps well onto a matching problem in an $n \times n$ bipartite graph. Packet transfers from inputs to various outputs can be modeled as edges in a bipartite graph, in which two edges in the final edge assignment (i.e., the match) can not be incident on any single input, nor incident upon the same outputs. A maximum matching algorithm is one that finds a match with the maximum size or weight. A maximum bipartite matching algorithm will give maximum utilization of all output ports. As mentioned, a MSM is not necessarily desirable, since it can lead to instability and unfairness under admissible traffic patterns, and starvation under inadmissible traffic patterns. However, a switch that utilizes MWBM is stable for any admissible traffic pattern for cell-based systems, but it is not generally stable for packet-based switches. In a packet-based switch, we assume packets are divided into equal-sized cells and when an input is enabled to transmit the first cell of a packet of L cells, the input/output connection must remain connected for the following $L - 1$ time slots.

The switch architecture discussed in this paper is a cell-based switch with a MWBM scheduler. The weight of the

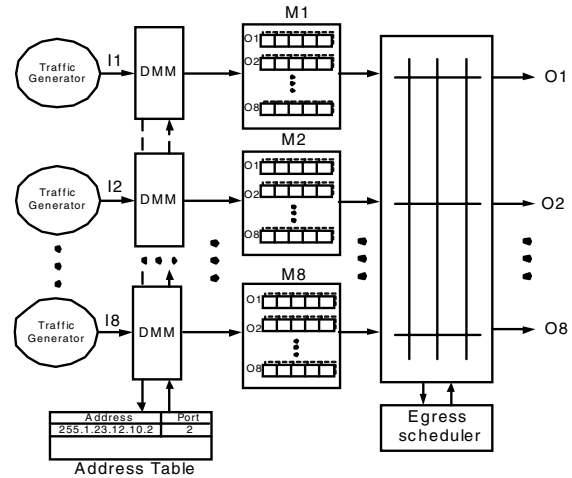


Figure 1. Input-queued switch architecture.

edge connecting input i to output j is often chosen to be some quantity that indicates the level of congestion; for example, the lengths of queues or the ages of packets can be used. A matching of particular importance for this paper is the MWBM when the weights are the lengths of queues. The length of each queue is a dynamic value that can change on each subsequent time slot. Given a weighted bipartite graph, the MWBM algorithm finds the matching whose weight is the heaviest. Figure 2 shows a weighted bipartite graph and its MWBM.

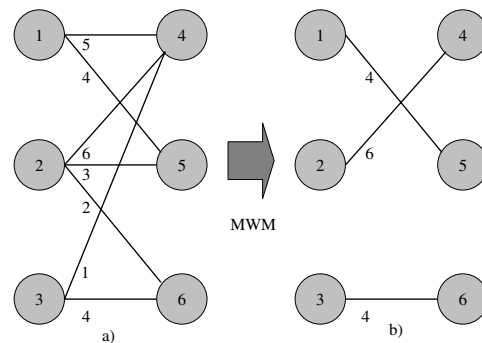


Figure 2. Weighted bipartite graph and its MWBM.

Bipartite matching has been studied extensively in the contexts of both sequential [15, 16] and parallel [17, 18, 19] computation. The most efficient known sequential algorithm for solving this problem converges in $O(n^3)$ run time, where n is the number of inputs [15]. However, the algo-

²Speedup is defined as increasing the memory access bandwidth in order to handle multiple packet transfers per time slot.

rithm reported in [15] is too complex to be implemented in hardware and too slow to be used in a fast switch.

Goldberg et al. [19] introduced a parallel sublinear algorithm to solve MWBM which runs in $O(n^{\frac{2}{3}} \times \log^3 n \times \log(nC))$ time, where C is the maximum weight of the graph edges, using a polynomial number of processing elements. Although the time complexity of this algorithm is sublinear, for a typical switch (e.g., 64×64) it is too slow and an efficient linear algorithm may run faster. The algorithm assumes a concurrent-read concurrent-write parallel random access machine (CRCW PRAM), which may be too complex to implement in hardware.

There has also been research on developing RNC (randomized Nick's Class) algorithms for MWBM. These include randomized parallel algorithms that run in expected polylogarithmic time on a polynomial number of processors and heuristic algorithms which find a maximal matching for the problem [20, 21, 22, 23, 24]. A maximal solution does not guarantee the maximum throughput for every admissible traffic pattern. In addition, a maximal solution may cause unfairness and instability under special traffic arrivals. Switches with randomized scheduling algorithms generally result in nondeterministic performance.

In this paper we describe an optimum solution for MWBM that runs in $O(\sqrt{n} \log^2 n)$ time. Our solution is based on the following constraint in input-queued cell-based switches: when switching unicast traffic (we do not consider multicast traffic in this paper), in each time slot, at most one cell may be received from each input and added to each input queue and at most one cell may be removed from each input queue and be transferred to each output. This means that the length of any queue will change very little during successive time slots. Since we utilize queue lengths to determine edge weights in MWBM, we are able to develop an $O(\sqrt{n} \log^2 n)$ time parallel algorithm that obtains the MWBM for time slot N by using the matching produced for time slot $N - 1$.

In the next section we describe our MWBM algorithm and its time complexity. Section 3 provides a description of our simulation environment and results. Section 4 provides conclusions and describes future work.

2. Parallel MWBM algorithm

Figure 3(a) shows a weighted bipartite graph $G(U, V, E)$, with $2n$ nodes. $U = \{1..n\}$ are the nodes on the left, $V = \{n + 1..2n\}$ are the nodes on the right and $E = \{e_{ij} | i \in U, j \in V\}$ are the edges of G . We assume G is a fully-connected bipartite graph. Edges can have zero weight. A MWBM algorithm searches among a space of $n!$ permutations (matchings) and selects one with the heaviest weight.

In a bipartite graph, a matching is defined as a set of

edges, no two of which share a node. An augmenting path is defined as a simple path whose edges are alternately matching³ and free⁴. An augmenting path starts from a free node and terminates on a free node. The length of an augmenting path is the number of edges it contains and its weight is the total weight of its free edges minus the weight of its matching edges.

Assuming M is a matching and P is an augmenting path, an exhaustive sequential solution to MWBM has the following steps:

```

Start with an empty matching M
While there is a positive P
{
  Find a P with the maximum weight
  Add free edges in P to M and remove
  matched edges from M
}
  
```

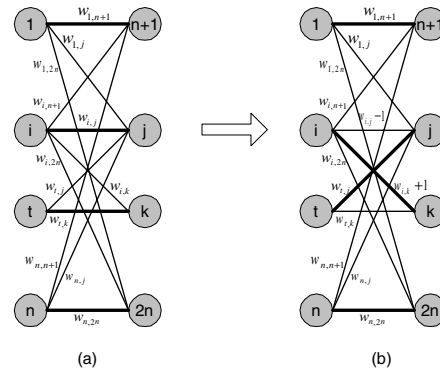


Figure 3. Weighted bipartite graph with $2n$ nodes.

The maximum weight augmenting path can be found by computing the longest path⁵ between any two free nodes of the bipartite graph. We use a matrix multiplication approach to find the longest path (LP) between every two nodes of the graph.

2.1. Finding the longest acyclic path

Assume $G(U, V, E)$ is a weighted bipartite graph with $m = 2n$ nodes. $U = \{1..n\}$ are the nodes on the left, $V = \{n + 1..2n\}$ are the nodes on the right and $E = \{e_{ij} | i \in U, j \in V\}$ are the edges of G . Edge e_{ij} has weight of w_{ij} . $M(T - 1)$ is the current matching ($T - 1$ denotes the time

³A matching edge is an edge contained in the matching.

⁴A free edge is an edge that is not in the matching.

⁵The longest path is defined as the heaviest path between two nodes.

slot prior to time slot T). When $m = 2n$, the $m \times m$ A matrix is created as follows:

$$a_{ij} = \begin{cases} w_{ij} & \text{if } (e_{ij} \notin M(T-1)) \\ & i = \{1..n\}, j = \{n+1..2n\} \\ -w_{ij} & \text{if } (e_{ij} \in M(T-1)) \\ & i = \{n+1..2n\}, j = \{1..n\} \\ 0 & \text{if } i = j \\ -\infty & \text{otherwise.} \end{cases}$$

Matrix A represents the value of every edge between each pair of nodes in graph G' . G' is a directed version of G such that every unmatched edge is directed left to right, and every matched edge is directed right to left. The weight of matched edges are multiplied by -1 . We use a matrix-matrix operation similar to matrix multiplication, but we substitute \times with $+$ and \sum with a max operation. We call this new operation a matrix sum-max operation. To avoid loops, we set the values of the main diagonal elements in matrix A to zero (i.e., the longest path from each node to itself is zero). We can show that applying the sum-max operation to A $m-2$ times or more produces matrix $D^{(m-1)}$, such that each element of this matrix ($d_{ij}^{(m-1)}$) denotes the acyclic longest path between any two nodes i and j [28]. Matrix $D^{(m)}$ is defined recursively as follows:

$D^{(m)} = \{d_{ij}^{(m)} = \max\{d_{ik}^{(m/2)} + d_{kj}^{(m/2)}\} \text{ for } 1 \leq k \leq m\}$. We have $D^{(1)} = A$.

Therefore, the longest path between every two nodes can be computed in $O(\log n)$ number of matrix sum-max operations. The matrix sum-max operation can be implemented in parallel. Parallel time complexity of this operation is $O(1)$ when using $O(n^4)$ processing elements on a CRCW PRAM model, and $O(\log n)$ when using $O(n^3)$ processing elements on a CREW PRAM model. In both approaches, $O(n^3)$ processing elements are used to calculate sum operations in $O(1)$ for each element on a CREW PRAM model. Finding the largest element in an array with n elements can be implemented in parallel. Obviously, using $O(n)$ processing elements we can implement a max operation in $O(\log n)$. The following pseudocode presents a parallel implementation of the max operation that runs in $O(1)$ time using $O(n^2)$ processing elements on a CRCW PRAM model. The algorithm returns the index and the value of the largest element in array B using $n(n-1)/2$ processing elements.

Phase 1:

```
For i = 1 to n-1 and j = i+1 to n,
  PE[i, j]:
  1- Read B[i]
  2- Read B[j]
  3- Clear B[i] if B[i] < B[j],
     otherwise clear B[j]
```

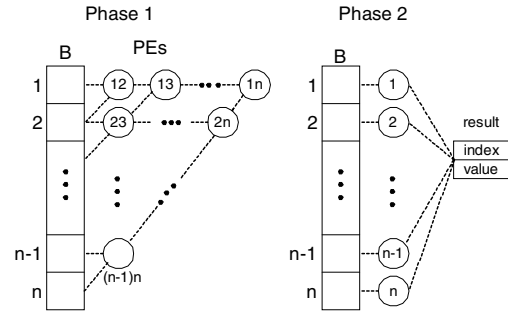


Figure 4. Finding largest element in an array in parallel in constant time. The PE's are processing elements and the B array holds weights.

Phase 2:

```
For i = 1 to n, PE[i]
  1- Read B[i]
  2- Write i to result_index and
     B[i] to result_value if B[i] > 0
```

Figure 4 shows the logical interconnection of processing elements to the memory elements of array B in a CRCW PRAM model. Note that we split the algorithm into two phases. In phase 1, we clear all memory elements except for the largest value, and in phase 2 we then read the values stored and write only if the value read was nonzero.

The longest path algorithm requires $O(n^3)$ memory space. This memory space is used to store intermediate paths between every two nodes during matrix sum-max operations. The overall parallel time complexity of the longest path algorithm is $O(\log n)$ with $O(n^4)$ processing elements using a CRCW PRAM model and $O(\log^2 n)$ with $O(n^3)$ processing elements on a CREW PRAM model. In the next section, we use the longest path algorithm to propose a linear time complexity parallel MWBM algorithm for a general bipartite graph. Our algorithm outperforms Goldberg's [19] parallel sublinear algorithm for any n less than 10^{15} , even when the maximum weight of edges is a small value. Our algorithm runs in $O(\log(n!))$ time, compared to the sublinear algorithm which runs in $O(n^{\frac{2}{3}} \times \log^3 n \times \log(nC))$ time, where C is the maximum weight of the graph edges. Although obtaining MWBM of a general bipartite graph is unlikely to be in the class NC, we will introduce a sublinear algorithm to obtain MWBM for input-queued switch architectures.

2.2. Linear time complexity parallel MWBM algorithm

Assume $G(U, V, E)$ is a weighted bipartite graph with $m = 2n$ nodes and M is a perfect MWBM for G . A perfect MWBM is a matching of size n . Since we can always assume G is a fully-connected bipartite graph (i.e., edges can have zero weight), every MWBM can be extended to be a perfect MWBM. Now we construct a new graph G' by adding one node to each side of G and adding as many edges to make G' a fully-connected bipartite graph. Therefore, $G'(U', V', E')$ will be a weighted bipartite graph with $m+2$ nodes such that $U' = U \cup \{u_1\}, V' = V \cup \{v_1\}, E' = E \cup E1 \cup E2$ when $E1 = \{(u_1, v) : \forall v \in V'\}$ and $E2 = \{(u, v_1) : \forall u \in U'\}$.

Theorem : The perfect MWBM for graph G' is equal to $M \oplus P$ when P is the longest augmenting path from u_1 to v_1 .

Proof : Assume M' is the perfect MWBM for graph G' and $P' = M \oplus M'$. P' is a path from u_1 to v_1 because P' can not contain a loop in G , nor can it be an unconnected graph. If P' contains a loop, this loop will be a positive augmenting path in G with respect to M and this contradicts the assumption that M is a MWBM for G . P' can not be unconnected because both M and M' are perfect matchings and every node in G which is also in P' is connected to M and M' by two different edges. P' must originate at u_1 and terminate at v_1 . Therefore, P' is a path from u_1 to v_1 and since M' is a MWBM for G' , P' must be the longest path from u_1 to v_1 .

Next, we propose an algorithm with $O(\log(n!))$ time complexity to find a MWBM for a weighted bipartite graph. Assume $G(U, V, E)$ is a weighted bipartite graph with $m = 2n$ nodes. $U = \{1..n\}$ are the nodes on the left, $V = \{n+1..2n\}$ are the nodes on the right and $E = \{e_{ij} | i \in U, j \in V\}$ are the edges of G . We can reconstruct the graph by starting with a graph G_1 which has only two nodes u_1 and v_{n+1} and one edge $e_{1,n+1}$. The MWBM for G_1 includes the single edge in the graph. Graph G_2 is created from G_1 by adding two other nodes u_2 and v_{n+2} from G and their incident edges to nodes in G_1 . We can find the MWBM for G_2 by finding the longest path from u_2 to v_{n+2} . And eventually, $G_n = G$ can be created from G_{n-1} by adding u_n and v_{2n} from G and their incident edges to nodes in G_{n-1} . The MWBM for the graph G_n can be achieved by applying the longest path algorithm. We showed that the longest path algorithm can be implemented in $O(\log n)$ on CRCW PRAM. Therefore, the time complexity of the MWBM algorithm is as follows. Finding a MWBM for graph G_i from G_{i-1} is found in $O(\log i)$ time, where i is the number of nodes in each of the graph. The overall complexity is $O(\log 2 + \log 4 + \dots + \log 2n)$, which is equal to $O(\log(n!))$. To obtain the MWBM for graph G_i

from G_{i-1} , we compute the LP between the new nodes in G_i that were not in G_{i-1} , and xor'ing these paths with the matching found for G_{i-1} .

The following pseudocode shows the complete algorithm of finding $M(n)$, starting from $M(1)$. The xor function finds the differences (in terms of edges) between two graphs:

```

Create G[1]
M[1] includes the single edge of G[1]
For i = (2..n) {
  Create G[i] from G[i-1] by adding
  nodes i and n+i and their edges
  Find the LP between i and i+n
  M[i] = M[i-1] xor LP
}
Return M(n) as the MWBM for G.

```

2.3. Sublinear time complexity parallel MWBM algorithm

Input-queued switches have the constraint that, in each time slot, at most one cell can be received from each input and added to each input queue, and at most one cell can be removed from each input queue and be transferred to each output. This means that in each time slot, at most, only a weight of a single edge incident upon an input node will be incremented and the weight of at most one edge weight will be decremented. In Figure 3, incident edges on node i have weights of w_{ij} for $j = \{n+1..2n\}$. For each node i in time slot T , there will be at most one j such that $w_{ij}(T) = w_{ij}(T-1) - 1$ and there might be at most one k such that $w_{ik}(T) = w_{ik}(T-1) + 1$. If there is any j , e_{ij} must be a matched edge (i.e., $e_{ij} \in M(T-1)$). When there is a k that satisfies this constraint, the following lemma can be used:

Lemma 1 : If $e_{ik} \in M(T-1)$ then $M(T) = M(T-1)$.

When e_{ik} is a free edge at time $T-1$ or there is no k that satisfies the equation above, but there is one j that satisfies the equation, we need to find an augmenting path P with maximum weight, starting from node i and terminating at node j . When P is found, we add free edges in P to $M(T-1)$ and remove matched edges in P from $M(T-1)$. The result will be a MWBM at time T for the weight changes to only one input node. We can use a parallel acyclic longest-path algorithm to find P in logarithmic time. By tracking all the input nodes during each iteration of the loop, we can find a MWBM in $O(n \log n)$ time. The following algorithm shows the complete algorithm of finding $M(T)$ starting from $M(T-1)$:⁶

⁶In the pseudocode, $w[i,j] == w_{ij}$.

```

For every node  $i = (1..n)$  {
  Update  $i$ th row of weights matrix  $W(T)$ 
  If  $((w[i,k](T) = w[i,k](T-1)+1$  AND
   $e[i,k]$  is not a member of  $M(T-1)$ 
  for  $k = (n+1..2n)$ )
  OR
   $(w[i,j](T) = w[i,j](T-1)-1$  AND
   $e[i,j]$  is a member of  $M(T-1)$ 
  for  $j = (n+1..2n))$  {
    Create matrix  $A$  based on  $M(T-1)$ 
    Find  $D(m-1)$ 
    if  $(D(m-1)[i,j] > e[i,j])$ 
      Update  $M(T-1)$  using the
      longest path between
       $i$  and  $j$  as an augmenting path.
  }
}
Return  $M(T-1)$  as the matching  $M(T)$ .

```

By using $O(n^4)$ processing elements, we can find D^{m-1} in $O(\log n)$ parallel run time. Other items inside the loop have constant parallel run time, therefore, the overall parallel run time for the loop will be $O(n \log n)$. Each processing element requires two modules, an add/subtract module and a min/max module.

We can improve our algorithm by running iterations of the main loop in parallel, if possible. Matrix $D^{(m-1)}$ already possesses information about the longest path between every two nodes of the bipartite graph. In our algorithm, we have only used the longest path between two nodes at each iteration. We propose to change the algorithm as follows:

1. update all rows of the weight matrix initially,
2. find the longest path between every two nodes and their corresponding augmenting paths,
3. apply disjoint augmenting paths to the matching matrix, and
4. repeat these steps until there are no more augmenting paths.

We can prove that after \sqrt{n} number of iterations, a maximum weight matching is achieved. The following pseudocode shows the complete algorithm of finding $M(T)$:

```

Update weights matrix ( $W(T)$ )
Do {
  Create matrix  $A$ 
  (using  $M(T-1)$  and  $W(T)$ )
  Find  $D(m-1)$ 
  Find disjoint augmenting paths
  (using MIS algorithm)
  Update  $M(T-1)$ 

```

```

  (using disjoint augmenting paths)
} While (no augmenting path is found)
Return  $M(T-1)$  as the matching  $M(T)$ .

```

In Figure 5 we show an example of our MWBM algorithm. In the upper left of the Figure, we show a bipartite graph with six nodes. The graph contains an initial matching (denoted with dashed lines). Based on the initial matching, the graph edges have been directed from left to right for unmatched edges and right to left for the matched edges. Matrix A , which is shown in the upper right portion of Figure 5, has been created based on the weights in the graph and the current matching. A^5 represents the longest path between any two nodes. In the lower left of the Figure we show only the elements of A^5 which are used by the algorithm. The positive longest paths are shown below A^5 . Both paths are identified by the algorithm as similar, and one of them is selected and applied to the graph. The result is shown in the lower right of the Figure. Any further attempt will not produce any positive longest path. Therefore, the result is a MWBM for the graph.

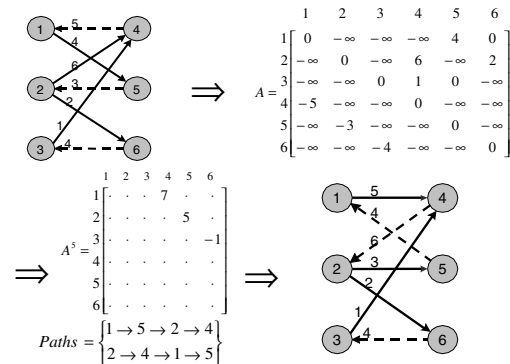


Figure 5. An example of our MWBM algorithm.

Creating the A matrix using $O(n^2)$ processing elements has a $O(1)$ time complexity. We can find D^{m-1} in $O(\log n)$ parallel run time by using $O(n^4)$ processing elements or in $O(\log^2 n)$ parallel run time by using $O(n^3)$ processing elements. We can find disjoint augmenting paths by using a greedy maximal independent set (MIS) algorithm. The algorithm has $O(\log^2 n)$ parallel run time by using $O(n^2)$ processing elements. Therefore each iteration has parallel run time of $O(\log^2 n)$ by using $O(n^3)$ processing elements. Since there are \sqrt{n} number of iterations, our complete MWBM algorithm has a parallel run time of $O(\sqrt{n} \log^2 n)$ using $O(n^3)$ processing elements.

3. Simulation and Comparison

We have developed a model to verify our algorithms and compare different MWBM algorithms. We have also developed a random bipartite graph (RBG) generator. Our RBG generator receives the number of graph inputs, the average number of edges per node, and the maximum weight of edges and generates a random bipartite graph. The average number of edges per node can be varied to simulate typical workloads for a input-queued switch. For example, the MWBM of a graph with an average number of one(n) edge(s) per node can represent scheduling of a switch with a light(heavy) load. Our model for a MWBM reads a graph and reports a MWBM for the graph, in addition to the number of iterations that the program takes to find the maximum weight matching. Figure 6 compares the associated time complexity for four different MWBM algorithms proposed for scheduling in input-queued switches. The first algorithm is a sequential Hungarian algorithm [16], which runs in $O(n^3)$ time. We have used this algorithm to show the differences between a sequential algorithm versus a parallel algorithm. The second algorithm is a parallel Goldberg algorithm [19] which runs in $O(n^{\frac{2}{3}} \times \log^3 n \times \log(nC))$ time, where C is the maximum weight of the graph edges. In our simulations, we use $C = 100$ for the maximum length of queues in our input-queued switch. The third and the fourth algorithms shown in Figure 6 are our linear and sublinear time complexity algorithms. On the X-axis we show the switch size, which represents the number of nodes in the graph and on the Y-axis we plot time complexity of each algorithm using a logarithmic rate.

As we can see from the results, for a switch size of 0-100 inputs, our two algorithms always outperform either the sequential Hungarian or parallel Goldberg algorithms. Our sublinear algorithm slightly outperforms our linear algorithm as the number of switch inputs increases.

4. Conclusions

In this paper we presented two maximum weight bipartite matching algorithms. We proposed a linear time complexity algorithm to find a MWBM for a general bipartite graph. We showed that for a range of the graph sizes, our algorithm outperforms the best known parallel MWBM algorithm. We also proposed a sublinear time complexity algorithm for input-queued switch applications. In future work we plan to consider developing a detailed hardware description of the MWBM scheduler, and considering how we can consider scalability within a unified switch fabric.

This work was supported by CenSSIS the Center for Subsurface Sensing and Imaging Systems, under the Engineering Research Centers Program of the NSF (Award Number EEC-9986821).

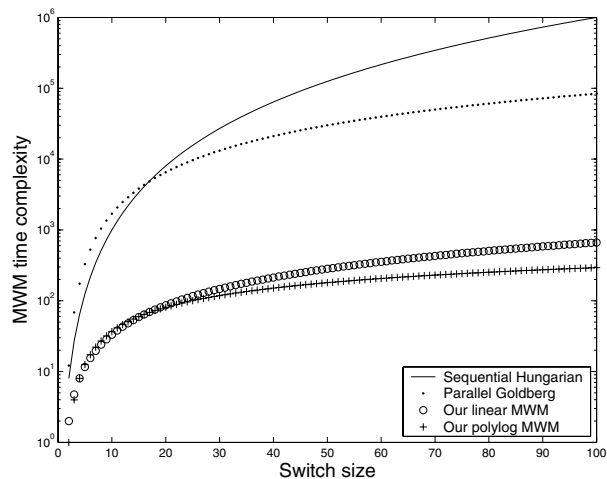


Figure 6. Comparing time complexity of different MWBM algorithms.

References

- [1] M. J. Karol, M. Hluchyj, and S. Morgan, "Input vs. output queuing on a space-division packet switch," in Proc. GLOBECOM 1986, pp. 659-665.
- [2] M. J. Karol, M. Hluchyj, "Queuing in high performance packet switching," IEEE JSAC vol. 6, no. 9, Dec. 1988, pp. 1587-1597.
- [3] M. Nabeshima, "Performance evaluation of a combined input- and crosspoint-queued switch," IEICE Trans. Communications, vol. E83-B, no. 3, Mar. 2000, pp. 737-741.
- [4] H. Ahmadi, W. E. Denzel, "A survey of modern high-performance switching techniques," IEEE JSAC, vol. 7, no. 7, pp. 1091-1103, Sep. 1989.
- [5] R. Y. Awdeh, H. T. Mofteh, "Survey of ATM switch architectures," Computer networks and ISDN systems, vol. 27, pp. 1567-1613, 1995.
- [6] S. Iyer, N. McKeown, "Making parallel packet switches practical," IEEE INFOCOM'01, Alaska, USA, pp. 1680-1687, Mar. 2001.
- [7] S. Iyer, R. Zhang, N. McKeown, "Routers with a single stage of buffering," ACM SigCOMM, pp. 251-264, Aug. 2002.
- [8] F. Abel, C. Minkenberg et al., "A four-terabit packet switch supporting long round-trip times," IEEE Micro magazine, pp. 10-23, Jan./Feb. 2003.
- [9] C. Minkenberg et al., "Current issues in packet switch design," ACM SigCOMM Computer Comm. Rev., vol. 33, no. 1, pp. 199-124, Jan. 2003.
- [10] Y. Tamir, G. Frazier, "High performance multi-queue buffers for VLSI communication switches," in Proc. 15th Ann. Symp. Computer Architecture, pp. 343-354, June 1988.
- [11] H. Obara, "Optimum architecture for input queuing ATM switches," IEE Electron. Lett., pp. 555-557, Mar. 28, 1991.
- [12] N. McKeown, V. Anantharam, J. Walrand, "Achieving 100% throughput in an input-queued switch," IEEE INFOCOM'96, pp. 296-302, 1996.

- [13] J. G. Dai, B. Prabhakar, "The throughput of data switches with and without speedup," IEEE INFOCOM 2000, vol. 2, pp. 556-564, Mar. 2000.
- [14] B. Prabhakar, N. McKeown, "On the speedup required for combined input and output queued switching," Automatica, vol. 35, pp. 1909-1920, 1999.
- [15] R. E. Tarjan, Data structures and network algorithms, Bell laboratories, 1983.
- [16] C. H. Papadimitriou, K. Steiglitz, Combinatorial optimization, algorithms and complexity, Prentice Hall 1982.
- [17] K. Mulmuley, U. V. Vazirani, V. V. Vazirani, "Matching is as easy as matrix inversion," Combinatorica, pp. 105-131, 1987.
- [18] C. H. Papadimitriou, Computational complexity, Addison-Wesley Publishing, 1994.
- [19] A. V. Goldberg, S. A. Plotkin, P. M. Vaidya, "Sublinear-time parallel algorithms for matching and related problems," IEEE symposium on foundations of computer science, pp. 174-185, 1993.
- [20] R. M. Karp, E. Upfal, A. Wigderson, "Constructing a maximum matching is in random NC," Combinatorica, 6:35-48, 1986.
- [21] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," IEEE INFOCOM'98, vol. 2, pp. 533-539, 1998.
- [22] D. Shah, P. Giaccone, B. Prabhakar, "An efficient randomized algorithm for input-queued switch scheduling," Hot Interconnects 9, pp. 3-8, Aug. 2001.
- [23] P. Giaccone, B. Prabhakar, D. Shah, "Towards simple, high-performance schedulers for high-aggregate bandwidth switches," IEEE INFOCOM'02, pp. 3:1160-1169, June 2002.
- [24] V. Tabatabaee, L. Tassiulas, "MNCM a new class of efficient scheduling algorithms for input-buffered switches with no speedup," IEEE INFOCOM'03, April 2003.
- [25] M. Fayyazi, D. Kaeli, Z. Navabi, "Dynamic input buffer allocation (DIBA) for fault tolerant Ethernet packet switching," International conference on parallel and distributed processing techniques and applications (PDPTA'03), vol. 2, pp. 819-823, June 2003.
- [26] N. McKeown, "Scheduling algorithms for input-queued cell switches," PhD Thesis, University of California at Berkeley, May 1995.
- [27] W. E. Leland, W. Willinger, M. S. Taqqu, D. V. Willson, "On the self-similar nature of ethernet traffic," IEEE/ACM Trans. on Networking, vol. 2, no. 1, pp. 1-15, Feb. 1994.
- [28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to algorithms, The MIT press, 1999.