Greedy Maximal Weighted Scheduling for Optical Packet Switches

Zhen Zhou and Mounir Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
Email: {cszz, hamdi}@cs.ust.hk

***Abstract*— Greedy algorithms are appealing not only because of their simplicity but also because of their effectiveness. In this paper, we study the possible greedy solutions for the optical switch scheduling problem. In particular, our algorithm 2-AUGMENTATION combines the idea of simple greedy algorithm [7] and augmenting paths in maximal matching algorithms [9]. Our analysis and simulation shows that it yields satisfactory performance after comparing with the simple greedy algorithm. By generalizing this approach, we exploit the possibility of a class of greedy algorithms based on the maximal weighted matching heuristic.**

I. INTRODUCTION

Researches on optical fabrics inside switches and routers have received an increasing number of interests over the past few years. It is because they provide more scalability, higher bit rate, and lower power consumption than their electronic counterparts. Current technologies for optical fabric include optical micro-electro-mechanical systems (MEMS), liquid crystal, bubble switches, thermo-optic, etc [1].

On the contrary of the advantages on the economic basis, optical switches suffer certain technical constraints. For one thing, the time required for establishing a connection between input-output ports in optical switches (or *reconfiguration delay*) takes hundreds of nanoseconds to a few milliseconds [1]. This delay possibly includes the time for mechanical settling, synchronization, etc. In other words, on a system with slotted time equals to 50 ns (64 bytes at 10 Gb/s), it takes around 10 to 10^5 time slots to reconfigurate the optical fabric. Therefore, traditional slot-by-slot scheduling is no longer a feasible approach. Because even when reconfiguration takes only one time slot, at least half of the bandwidth is wasted on setting up the fabric in between each transmission. The efficiency of such an optical switch is then at most 50%.

In order to promote the switching efficiency, one has to reduce the scheduling rate such that each schedule holds for several time slots. This scheduling problem is called Time Slot Assignment (TSA) problem and first studied in the context of Satellite Switched Time Division Multiple Access (SS/TDMA) systems. The goal is to minimize the *switching cost*, which is the time spent on the makespan of transmitting a batch of packets and on reconfiguring optical fabrics. Several scheduling algorithms have been proposed [2]–[7] toward efficient solutions. It has also been proven to be NP-hard by Gopal and Wong [3] for the case when the number of the packets in each batch are small, and by Li and Hamdi [5] for the more general case. Therefore, we shall find good approximation algorithms that minimize the switching cost.

This work was supported in part by a grant from Hong Kong Research Grant Council (Grant Number: RGC HKUST6160/03E).

Among those approaches proposed, greedy algorithms are appealing not only because of their simplicity but also because of their effectiveness. In general, there are two sorts of greedy heuristics we could apply to this problem.

1) To transmit requests of similar magnitude in a configuration;
2) to transmit requests that contain as much magnitude as possible in a configuration.

The greedy algorithm (or GREEDY for short) introduced in [7] and K-Transponders introduced in [3] applied the first heuristic. In particular, study in [7] has shown that the simple greedy algorithm is a 2-approximation algorithm with a reasonable time complexity.

In this paper, we introduce a greedy approach of the second heuristic. This Greedy Maximal Weighted Scheduling (GMWS) approach actually consists of a class of algorithms parameterized on the number of edges augmented in each step. After formally formulate the TSA problem in Section II, we will first introduce a representative of the GMWS algorithms, named 2-AUGMENTATION, in Section III. As the name suggests, in each step the 2-AUGMENTATION algorithm augments the magnitude of two requests that are greedily picked from the pre-sorted request list.

The performance of 2-AUGMENTATION will be analyzed in Section IV. The time complexity of the 2-AUGMENTATION algorithm is $O(N^2 \log N)$ and it produces at most $2N - 1$ configurations, given an $N \times N$ optical switch. By simulation, we show that 2-AUGMENTATION performs slightly better than GREEDY in terms of the scheduling makespan and also the number of configurations it incurs, on uniformly random input traffics.

In Section V, we introduce a class of greedy algorithms that generalize 2-AUGMENTATION, in which we augment more than two requests at a time. The extreme of this generalization is actually the Maximum Weighted Matching (MWM) algorithm, if we treat the scheduling problem as searching for matchings on a bipartite graph. However, augmenting paths for MWM is very costly in terms of running time ($O(N^3)$ for each maximum matching [8]). That is also why we shall reveal

0-7803-9569-7/06/$20.00

in our further discussion that augmenting many requests at the same time may not be practical since it is time-consuming.

Finally, we conclude this paper in Section VI with remarks and open questions.

II. Preliminaries

In this section we formally describe the optical switching model and its scheduling problem. We consider an $N \times N$ input-queued switch with Virtual Output Queuing (VOQ) for storing fixed size packets, and an optical fabric for switching packets (shown in Fig. 1). Take the popular two-dimensional

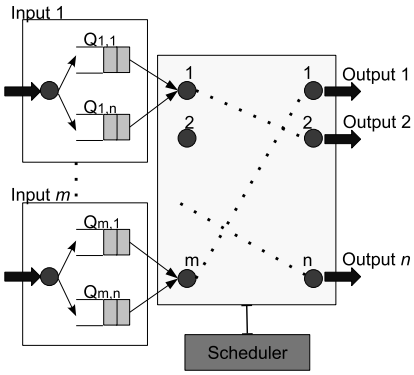

Fig. 1. A router structure with virtual output queuing.

(2D) MEMS as an example of the optical fabrics. The basic switching elements are tiny mirrors with binary ON/OFF positions, which are arranged in crossbar configuration. Switching is done by reflection of light. In general, if the (i, j) mirror is raised up (i.e., ON position), it directs light from the ith input fiber to the jth output fiber. In this model, packets arrive at input ports and will be temporarily stored in a queue associated to the given output port. Time is slotted and we assume traffic is *admissible*, so that there is at most one fixed-size packet received from any input port or dispatched to any output port in one time slot. In batch scheduling, packets are first accumulated for T time slots, where T, or the *batch length*, is a predefined system parameter. We then obtain a *traffic matrix* (i.e., *batch*)

$$D = [d_{i,j}]_{N\times N} \quad d_{i,j} \geq 0,$$

in which any row sum and column sum should not exceed the accumulated port capacity under admissible traffic. That is,

$$\sum_i d_{i,j} \leq T \text{ and } \sum_j d_{i,j} \leq T, \quad (T > N).$$

A centralized scheduler is responsible to find matchings, i.e., *switching configurations*, between the inputs and outputs. Then the fabric is configured according to the matchings in order to deliver packets from the inputs to the outputs. Each time the switch starts up a new switching configuration, it introduces a *reconfiguration overhead* δ for arranging and synchronizing the mirrors. The objective of the scheduler is to minimize the scheduling cost, which is the sum of the scheduling makespan and the total overhead.

In TSA setting, we reduce the rate of finding matchings and pipeline the switching tasks. Instead of finding a matching for every time slot, we accumulate packets for T time slots (in the accumulating phase). Then the optical fabric is configured according to the computed matching scheme (in the scheduling phase). Finally, it holds for another T time slots to switch the packets to the output ports (in the transmitting phase). In such a manner, pipelining is allowed as shown in Fig. 2. We are guaranteed to have 100% throughput and the worst case switching delay bounded by the sum of the time spent in the three phases, given that packets that arrive in the first phase are transmitted after the third phase. It is also critical that the third phase never takes longer than the time of the accumulating phase, because otherwise the switch is not stable.

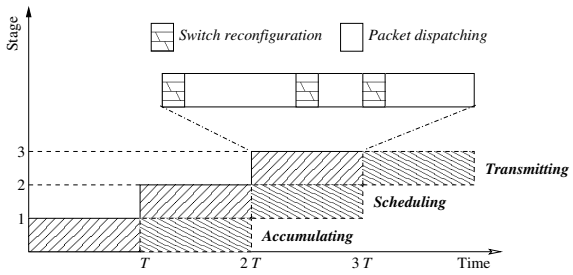

Fig. 2. Three phases of batch scheduling and pipelining.

There are interchangeably two representation of the TSA problem. First, we may denote each configuration as a *(partial) permutation matrix* $P = [p_{i,j}]_{N\times N}$ which is a 0-1 matrix with at most one "1" on each row or column. An "1" on ith row jth column indicates that input i shall connect with output j in the current switching. A scheduling algorithm produces S configurations P_k $(1 \leq k \leq S)$, each lasts for w_k time slots, that *cover* the traffic matrix. So $W = \sum_{k=1}^{S} w_k$ is the *scheduling makespan* for batch D, and $S\delta$ is the total overhead. The *scheduling cost* C charged to the batch scheduler is the sum of both. We can then express our problem as follows.

$$\begin{array}{ll} \text{objective:} & \min \sum_{k=1}^{S} w_k + S\delta \\ \text{subject to} & D \leq \sum_{k=1}^{S} w_k P_k \qquad 1 \leq k \leq S \end{array}$$

Note that it is generally difficult to determine the set of configurations $\{P_k\}$. Because there are as much as $\binom{N!}{S}$ such candidates [1]. Hence it is not a Linear Programming problem because the constraints are not fixed. Once the set $\{P_k\}$ is determined, corresponding weights are merely the duration of each configuration in order to cover the batch entries.

Alternatively (see Fig. 3), we can formulate the problem as bipartite matching on graph $G = (V_I, V_O, D)$. The vertices in V_I are the inputs, and the ones in V_O are the outputs. The edge connecting input vertex i and output vertex j is weighted by the traffic demand $d_{i,j}$. The traffic matrix D is actually an adjacent matrix representation of the edge weights. So the equivalent representation of the scheduling problem is

Find S matchings $\{M_1, \ldots, M_S\}$ with corresponding weights $\{w(M_1), \ldots, w(M_S)\}$ that

[1] $\binom{a}{b}$ is the notation for $\frac{a!}{(a-b)!b!}$, and $N^{\underline{S}}$ for $\frac{N!}{(N-S)!}$.

cover the bipartite graph $G = (V_I, V_O, D)$ such that $\sum_{k=1}^{S} w(M_k) + S\delta$ is minimized.

The problem can be clearly approximated by making use of maximum matching algorithms. But each run of Maximum Size Matching or Maximum Weight Matching will incur $O(N^{2.5})$ time [9] or $O(N^3)$ time [8] respectively. This will force the overall TSA algorithm run for at least $O(N^{3.5})$ time. Instead, we may use the greedy approach that largely simplifies the complexity yet provides satisfactory results.

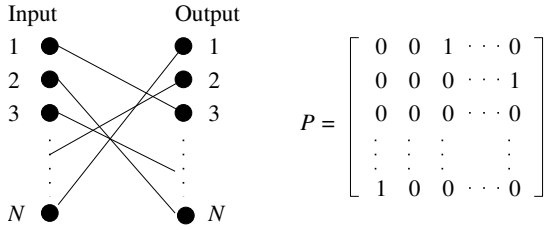

Fig. 3. Example of bipartite matching (left) and permutation matrix (right).

We consider in this paper *on-preemptive scheduling*, where all $d_{i,j}$ packets have to be transmitted during one connection of input i and output j. Whereas in *preemptive scheduling*, every transmission request $d_{i,j}$ can be split and covered by several configurations if necessary. However, it will generally introduce more than N configurations, up to a number of $(N^2 - 2N + 2)$ [2]. Crescenzi et al. [10] considered the preemptive batch scheduling problem and provided two 2-approximation algorithms. Afrati et al. [11] derived an algorithm with an improved approximation factor of $2 - \frac{1}{\delta+1}$. However, the running time of the known preemptive batch scheduling algorithms are either peudo-polynomial or in $\Omega(N^4)$ time.

III. 2-Augmentation Algorithm

In this section, we propose a greedy algorithm for TSA problem, which shall make good use of the information of the input traffic in making scheduling decisions.

Intuitively, we first sort the entries in D into non-increasing ordered list. To fill up a configuration, we repeat the following operation: We look ahead for two entries to fill in the configuration greedily, meanwhile augmenting them for a larger total weight. Then fill the larger one of the pair into the configuration. In comparison, GREEDY does not have the augmenting operation. It will just fill in the entries in sorted order whenever they are fit into the configuration.

The deterministic algorithm will make scheduling decisions based on the entry weights of the input batches. Because we augment two entries at a time (corresponding to two nonadjacent edges in the bipartite graph setting), our algorithms is called 2-AUGMENTATION (2AUG). It is presented below.

Algorithm 2AUG(D)

Input:

$N \times N$ non-negative integer matrix D.

Output:

A set of permutation matrices $P_1, \ldots, P_S$ and corresponding non-negative integer weights $w_1, \ldots, w_S$.

Procedure:

1) Sort the N^2 entries of D in non-increasing order, and put them in a list L.
2) While L is not empty.
   a) Create a new (empty) configuration P_k.
   b) While there are two entries $d_{i,j}, d_{r,c} \in L$ that can be filled to P_k ($i \neq r$, $j \neq c$).
   Check if $\exists d_{i,c}, d_{r,j} \in L$ such that
   $$d_{i,c} + d_{r,j} > d_{i,j} + d_{r,c}.$$
   Yes. Fill in $d_{i,c}$ (w.l.o.g. assume $d_{i,c} \geq d_{r,j}$), and remove it from L.
   No. Fill in $d_{i,j}$ (w.l.o.g. assume $d_{i,j} \geq d_{r,c}$), and remove it from L.
   c) Find the last entry from L that can be filled. Fill it and remove it from L.
3) Set weight w_k for each P_k.
4) Output.

Let us demonstrate the execution of 2AUG on a 4×4 traffic matrix given below.

$$D = \begin{bmatrix} 3 & 2 & 0 & 7 \\ 4 & 1 & 9 & 3 \\ 5 & 4 & 6 & 8 \\ 5 & 6 & 7 & 2 \end{bmatrix}$$

We first come up with the non-decreasing ordered list $L = \{d_{2,3}, d_{3,4}, \ldots, d_{1,3}\}$. We find $d_{2,3}$ and $d_{3,4}$ and then fill $d_{2,3}$ in P_1, because $d_{2,3} + d_{3,4} > d_{2,4} + d_{3,3}$. Next, we find $d_{3,4}$ and $d_{4,2}$ and fill $d_{3,4}$ in P_1 for the same reason. The next one filled is $d_{4,2}$, because $d_{4,2} + d_{1,1} > d_{1,2} + d_{4,1}$. Finally for P_1, we have no other choice but filling in $d_{1,1}$.

Repeating this procedure for P_2, we can fill in $d_{1,4}$ and $d_{4,3}$ with no doubt. The next pair we find according to the list L is ($d_{3,1}$, $d_{2,2}$). However, we can augment this because $d_{3,1} + d_{2,2} < d_{2,1} + d_{3,2}$. So we fill $d_{2,1}$ into P_2 instead of $d_{3,1}$. Following this procedure, P_3 and P_4 are obtained in the same manner.

As a result, we obtain the following four configurations.

$$P_1 = \begin{bmatrix} 3 & & & \\ & & 9 & \\ & & & 8 \\ & 6 & & \end{bmatrix}, \quad P_2 = \begin{bmatrix} & & & 7 \\ 4 & & & \\ & 4 & & \\ & & 7 & \end{bmatrix}$$

$$P_3 = \begin{bmatrix} & 2 & & \\ & & & 3 \\ & & 6 & \\ 5 & & & \end{bmatrix}, \quad P_4 = \begin{bmatrix} & & 0 & \\ & 1 & & \\ 5 & & & \\ & & & 2 \end{bmatrix}$$

The scheduling makespan $W_{2AUG}(D)$ is the sum of the largest entry in the configurations, equals 27. The maximum row/column sum is $T = 23$. This means that 2AUG approximates the optimal makespan within a factor of 2.

IV. Performance of 2-Augmentation

In this section, we analyze or experiment the performance of 2AUG in three aspects, namely its time complexity, the number of configurations it produces, and its scheduling efficiency.

Let us first consider the running time of 2AUG. Sorting the entries of D takes $O(N^2 \log N)$. Within the recursion step, finding two entries in L and augmenting them with their neighbors takes $O(N)$ time in the worst case. However, we may speed it up in a twofold way. First, observe that when we go through L for finding the two entries to fill, we may encounter a few entries that are not suitable for the current configuration. They certainly have to be put in some later configurations. To save time, we may remember them and prepare them in advance. We take the traffic matrix D in Section III for example. Entry $d_{4,3} = 7$ will be encountered in list L when packing P_1. We know that it must appear in the next configuration. We may put it into P_2 in advance so that we do not need to go back to it later when packing P_2. In such a processing manner, each entry in list L can be encountered only once. That is, we actually perform a linear scan through the list. Step 2b in the algorithm's procedure will be executed at most twice. Secondly, a careful implementation of the matrix representation for each packet batch helps to ease the neighbors' searching. The total running time is dominated by the sorting routine, which accounts for $O(N^2 \log N)$.

Another performance measure is how many configurations 2AUG will construct. Therefore we prove the following lemma.

Lemma 1: The number of configurations constructed by 2AUG is at most $2N - 1$.

Proof: Because of the greedy nature of 2AUG, if $d_{i,j}$ is not in a configuration P_k, then P_k must have included either $d_{i,c}$ (for some $c \neq j$) or $d_{r,j}$ (for some $r \neq i$) which blocks $d_{i,j}$ in this configuration.

Now suppose there are more than $2N-1$ configurations and $d_{i,j}$ is covered by P_{2N}. However, there are at most $2N - 2$ non-zero entries on row i and column j. That means $d_{i,j}$ is not blocked by any entry in P_{2N-1}, which is a contradiction to our assumption. ■

In order to show the scheduling efficiency, we compare the makespan incurred by 2AUG and GREEDY. In [7], Kesselman and Kogan proved that GREEDY is a 2-approximation algorithm for arbitrary configuration delay. We shall see in our simulation results that 2AUG perform as well as GREEDY, sometimes a little bit better. Actually it is not a coincidence. Because without augmentation, 2AUG is almost the same as GREEDY. Notice that because we do not have any idea of the reconfiguration delay δ, we abuse the notion of the approximation ratio. Instead of the ratio of scheduling costs, we measure the ratio of the algorithms' scheduling makespan W_{ALG} and the optimal makespan W_{OPT}.

We have conducted simulations to compare the performance of GREEDY and 2AUG for various switches and traffic settings. The input traffic is defined as follows: Packets arrive at inputs according to independent and identical distributed (i.i.d.) Bernoulli processes. The traffic is admissible with respect to the batch length T. To test for different value of T, we made it also random and $T \gg N$. In our figures below, we present the simulation results on a 32×32 switch.

We first looked into fifty arbitrarily selected samples. The blue (upper) polyline in Fig. 4 depicts the makespan ratio W_{GREEDY}/T, and the red (lower) polyline depicts W_{2AUG}/T. Even though the difference is small, we observe that 2AUG often incurs a smaller makespan than GREEDY. However, generating smaller makespan does not necessarily imply producing more configurations. In fact in Table I, we observe that 2AUG produces more or less the same number of configurations as GREEDY does. Depending on the actually traffic, they both produce 35 to 38 configurations for the one thousand input batches. 2AUG has fewer occurrences to produce 35 or 37 configurations than GREEDY. On average, 2AUG is 1.4% better.

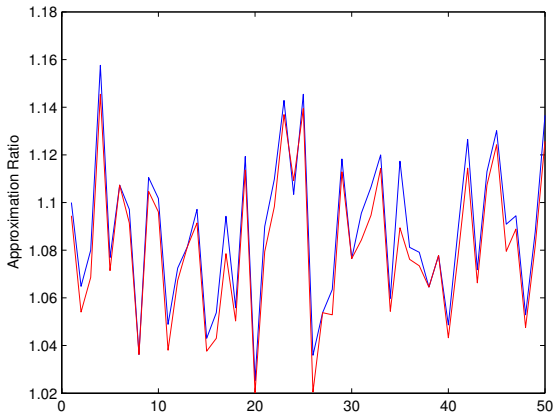

Fig. 4. Fifty samples of the makespan ratios on a 32×32 switch; blue (upper) for GREEDY and red (lower) for 2AUG.

TABLE I

The number of configurations produced by GREEDY and 2AUG (on a 32×32 switch over 10^3 uniformly random inputs)

# of Configurations	35	36	37	38	*Average*
GREEDY	268	623	105	3	35.5
2AUG	261	632	101	6	35

The overall distribution on the makespan ratios of these two algorithms is presented in Fig. 5. The average makespan ratio for GREEDY is 1.09945, and 1.093925 for 2AUG. We also did simulations for a 64×64 switch setting with 10^3 uniformly random input batches. On average, the makespan ratio of GREEDY is 1.06297597 by utilizing 69.636 configurations; and the makespan ratio of 2AUG is 1.06189832 by utilizing 69.632 configurations.

Despite the positive evidence displayed above, we have to point out that 2AUG is not always better than GREEDY. Indeed, the 4 example we provided in Section III was carefully manipulated such that the makespan by GREEDY is one less. Therefore, it is really a matter of traffic inputs that make

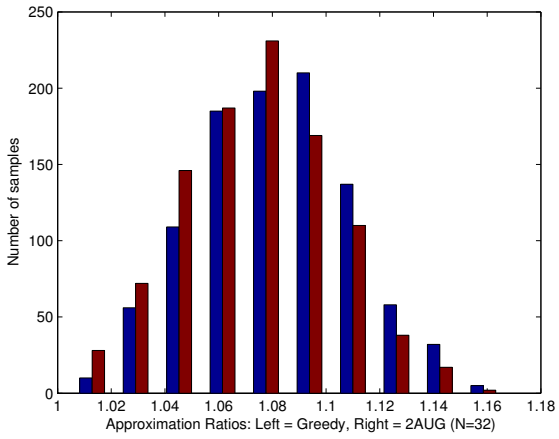

Fig. 5. Statistical distribution of W_{ALG}/W_{OPT} on a 32×32 switch.

the 2AUG and GREEDY slightly differ. We also conducted simulations with some sparse traffic matrices and some dense traffic matrices. The difference between their performances is negligible.

V. Greedy Maximal Weighted Scheduling

After we have seen a greedy algorithm that augments two edges every time in matching, we may now think of an extension of this idea. In general, we may look ahead and then augment l edges ($2 \leq l \leq N$) in each iteration. As to the extreme, N-augmentation is exactly a Maximum Weighted Matching algorithm, which augments N edges simultaneously for a perfect matching. Although we may not provide a rigorous proof, our intuition tells us that the more augmentation work the algorithm does per iteration, the better the performance is. We are expected to see that the scheduling cost approaches the optimal one, and the number of configurations approaches N.

The cost we pay for better performance is the increase of the time complexity. In 2AUG, we only augment the total weight among two sets of neighboring edges. Implicitly, we compute and find the maximum of 2! choices in $O(2! \cdot 2)$ time. If the number of the edges to augment per iteration is l, there will be $l!$ possible sets of edges. Finding the maximum weighted set among the $l!$ candidates takes $O(l! \cdot l)$ time in the worst case. It is so time-consuming, even not mentioning the time spent for searching the l edges that can be augmented. Therefore, it may not be worthwhile to augment several edges at a time, unless they are proved to be advantageous in terms of efficiency.

VI. Conclusion

Along with the fast development of Internet, optical switching technologies are becoming attractive for its huge capability and scalability. The disadvantage of optical switching comes from large reconfiguration overhead due to the technology constraints. Besides, researchers have to compromise other difficulties in designing scheduling algorithms such as simplicity, time limitation and efficiency. It turns out that greedy algorithms are suitable for such optimization problems.

The 2-AUGMENTATION algorithm proposed in this paper tries to achieve a better performance by applying augmentation with the greedy strategy. It slightly outperforms the simple greedy algorithm in $O(N^2 \log N)$ time according to our simulation. We also exploited the extension of this approach and analyzed its major disadvantage of large time complexity. It remains open whether augmenting more than two edges at a time will yield a better result. It will be also interesting to study how greedy strategy could be applied to preemptive scheduling in optical switches.

References

[1] X. H. Ma and G. H. Kuo, "Optical switching technology comparison: optical MEMS vs. other technologies," *IEEE Commun. Mag.*, vol. 41, pp. S16–S23, 2003.

[2] T. Inukai. An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications*, vol. 27, pp. 1449 – 1455, 1979.

[3] I. S. Gopal and C. K. Wong. Minimizing the number of switchings in an SS/TDMA system. *IEEE Transactions on Communications*, vol. 33, pp. 497 – 501, 1985.

[4] B. Towles and W. J. Dally. Guaranteed scheduling for switches with configuration overhead. *IEEE/ACM Transactions on Networking*, vol. 11, pp. 835 – 847, 2003.

[5] X. Li and M. Hamdi, "On scheduling optical packet switches with reconfiguration delay," *IEEE J. Select. Areas Commun.*, vol. 21, pp. 1156–1164, Sept. 2003.

[6] Z. Zhou, X. Li, and M. Hamdi. Fast scheduling for optical packet switches with minimum configurations. In Proceedings of *IEEE Workshop on High Performance Switching and Routing (HPSR05)*. Hong Kong, May 2005.

[7] A. Kesselman and K. Kogan, Non-Preemptive Scheduling of Optical Switches. In Proceedings of *2004 IEEE Global Telecommunications Conference (GLOBECOM'04)*, USA, November 2004.

[8] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 434 – 443, 1990.

[9] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, vol. 2(4), pp. 225 – 231, 1973

[10] P. Crescenzi, X. Deng, and C. H. Papadimitriou. On approximating a scheduling problem. *Journal of Combinatorial Optimization*, vol. 5, pp. 287 – 297, 2001.

[11] F. Afrati, T. Aslanidis, E. Bampis, and I. Milis. Scheduling in switching networks with set-up delays. *Journal of Combinatorial Optimization*, vol. 9, pp. 49-57, 2005.