Fast Matching Algorithms for Repetitive Optimization: An Application to Switch Scheduling

Supratim Deb
Bell Labs Research India
Bangalore-560017, INDIA
Email: supratim@lucent.com

Devavrat Shah
Massachusetts Inst. of Technology
Cambridge, MA-02139, USA
Email: devavrat@mit.edu

Sanjay Shakkottai
The Univ. of Texas at Austin
Austin, TX-78712 USA
E-mail: shakkott@ece.utexas.edu

***Abstract*— Scheduling in an input buffered switch can be viewed as repeated matching (corresponding to once every time slot) in a bipartite graph. It has been shown that scheduling algorithms based on *maximum weight matching* (MWM) with queue-lengths as the weights, leads to excellent performance in terms of throughput and delay. However, computing MWM using a strongly polynomial time algorithm requires $\mathbf{O(n^3)}$ operations in an $\mathbf{n \times n}$ switch.**

The main motivation for this paper comes from the following two observations: (1) The weights of edges (packets in buffer) change only a *little* between successive time slots, thus changing the weight of the MWM only by a small amount; (2) Under MWM algorithm, the average queue-sizes are *small*. The main difficulty in utilizing these properties comes from the fact that small changes in weights can change the matching arbitrarily, thus making it hard for current popular algorithms to compute an MWM quickly using the information from past (or memory).

In this paper, we develop an algorithm based on the algorithm of Cunningham and Marsh [1] that uses the above two properties in order to to find the new MWM quickly. Specifically, for an $\mathbf{n}$ port input-queued switch, i.e. a switch with $\mathbf{n}$ inputs and $\mathbf{n}$ outputs, our algorithm finds MWM in $\mathbf{O(n^2)}$ operations using past information. We believe that the incremental nature of our algorithm may be useful in the context of other applications.

I. Introduction

Study the past if you would divine the future.
- Confucius (c. 551-c. 479 BC)

Over the past few years the input-buffered switch architecture has become dominant in high speed switching. This is mainly due to the fact that the memory bandwidth of its packet buffers is very low compared to that of an output-queued or a shared-memory architecture. Furthermore, for an $n \times n$ switch, an output-buffered architecture requires a switch fabric with a processing speed of n times the line-rate, whereas an input-buffered switch requires a fabric with a processing speed as much as the line-rate.

Fig. 1 shows the logical structure for an input-queued (IQ) switch. Suppose that time is slotted so that at most one packet can arrive at each input in one time slot. Packets arriving at input i and destined for output j are buffered in a "virtual output queue" (VOQ), denoted here by VOQ_{ij}. The use of virtual output queues avoids performance degradation due to the head-of-line blocking phenomenon [2]. Let the average cell arrival rate at input i for output j be λ_{ij}. The incoming traffic is called *admissible* if $\sum_{i=1}^{n} \lambda_{ij} < 1$, and $\sum_{j=1}^{n} \lambda_{ij} < 1$. We assume that packets are switched from inputs to outputs by a crossbar fabric. When switching unicast traffic[1], this fabric imposes the following constraint: in each time slot, at most one packet may be removed from each input and at most one packet may be transferred to each output.

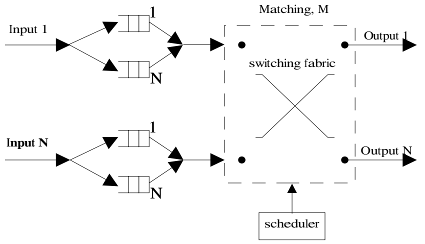

Fig. 1. Logical structure of an input-queued cell switch

To perform well, an $n \times n$ input-queued switch requires a good packet scheduling algorithm for determining which inputs to connect with which outputs in each time slot. It is well-known that the crossbar constraint makes the switch scheduling problem a matching problem in an $n \times n$ weighted bipartite graph. The weight of the edge connecting input i to output j is often chosen to be some quantity that indicates the level of congestion; for example, queue-lengths or the ages of packets.

A matching for this bipartite graph is a valid schedule for the switch. Note that a valid matching can be seen as a permutation of the n outputs. In this paper we will use the words *schedule*, *matching* and *permutation* interchangeably. A matching of particular importance for this paper is the maximum weight matching algorithm (MWM). Given a weighted bipartite graph, the MWM finds that matching whose weight is the highest. For example, Figure 2 shows a weighted bipartite graph and one valid schedule (or matching). We shall use $S(t)$ to denote the schedule used by the switch at time t. There are two main quantities for measuring the performance of a switch scheduling algorithm: throughput and delay. In the papers [3], [4], authors showed that under Bernoulli IID packet arrival processes the MWM is stable so long as no input or output is oversubscribed[2]. Further, MWM (with weights as function of queue-size), is known to perform optimally in terms of delay

[1] We do not consider multicast traffic in this paper.

[2] The weights were taken to be the length of Q_{ij} originally and later work [5] took the weights to be the age of the oldest packet in Q_{ij}.

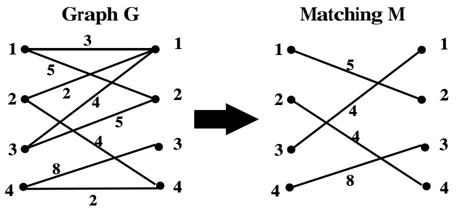

Fig. 2. Example of weighted bipartite graph and its maximum weight matching.

[6].

In addition to IQ switches, there are many other examples of scheduling in various problems arising in networks, that have throughput optimal policies as "maximum weight matching" type algorithms. For example, Max-pressure policy for network of queues [7], scheduling in Radio hop network [4], Generalized MWM [8] and scheduling in network of switches [9]. In summary, MWM type algorithms are the heart of good scheduling solutions in many network applications.

However, MWM is too complicated for implementation in its full generality in any of these applications. In particular, in case of $n \times n$ IQ switch, the best known strongly polynomial[3] time algorithm requires $O(n^3)$ operations. There are other known MWM algorithms that are not strongly polynomial but have better time-complexity when the weights are *small*. The best known among such algorithms (to the best of our knowledge) is by Gabow and Tarjan [10] that has performs as follows: Let weights of each edge be an integer between $\{1, \ldots, M\}$, V be number of nodes in the graph and E be number of edges in the graph. Then the algorithm takes $O(\sqrt{V \log V \alpha(E,V)} E \log(MV))$ time, where $\alpha(\cdot,\cdot)$ is the inverse Acremann's function[4]. This algorithm, while more efficient, is quite complicated for the purpose of implementation. These implementation related considerations have led to proposal of a number of practicable scheduling algorithms; notably, iSLIP [11], iLQF [11], RPA [12], MUCS [13] and WFA [14]; very little attention has been given to the complexity of MWM algorithm while taking into account the specifics of the application, e.g. switch dynamics.

Recently, [15] and [16], exploited the following observation to obtain simple-to-implement stable approximations of MWM: In each time slot, at most one packet arrives (departs) per input (output). This means that queue-lengths, taken to be the weights by MWM, change very little during successive time slots. Thus, a heavy matching will continue to be heavy over a few time slots, suggesting that carrying some information, or retaining memory, between iterations should help simplify the implementation while maintaining a high level of performance.

While the algorithms of [15], [16] are good approximations to MWM and are stable, they do not compute the MWM. Furthermore, none of the algorithms for switching take into account the dynamics of the queueing behavior in a switch. In this paper, we are motivated by the following questions:

(1) Is it possible to improve the computational complexity of the MWM provided the information of MWM from the previous time is utilized?

(2) If yes, how much improvement can be obtained?

We answer questions (1) in affirmative and quantify the improvement in the complexity using the observation that switch state (in terms of queue-size) changes "very little" in successive time-slots. We find that complexity improves significantly in various scenarios. The exact details are soon to follow.

A. Switch model

Let time be indexed by m and we will denote by queue (i,j) the queue for output j at input i. Initially, $m = 0$. Let the $n \times n$ integer valued matrix $Q(m) = [Q_{ij}(m)]$ denote the queue-sizes of the switch at the beginning of time-slot $m \geq 0$. We assume that the switch starts empty, i.e. $Q(0) = [0]$.

The arrival process to queue (i,j) at time m, denoted by $A_{ij}(m)$ [5], is exogenous while the service process depends on the scheduling algorithm. The arrival process is stationary and ergodic. We assume that line-rates are normalized to one as well as packets are of unit-size. Hence, at most one packet can arrive at each input. is, $\sum_{j=1}^{n} A_{ij} \in \{0,1\}$. Let the arrival rate-matrix be $\lambda = [\lambda_{ij}]$, where

$$E[A_{ij}(0)] \quad = \quad \lambda_{ij}. \tag{1}$$

Under Bernoulli IID distribution, $A_{ij}(\cdot)$ are IID random variables with $\Pr(A_{ij}(0) = 1) = \lambda_{ij}$ (note that IID refers to IID in time). We will use this specific distribution to obtain certain results. Due to constraints on arrivals and departures, we call an arrival rate-matrix λ as *admissible* if it is strictly doubly sub-stochastic, i.e.

$$\lambda_{i\cdot} < 1; \qquad \lambda_{\cdot j} < 1, \ \ \forall\ i,\ j. \tag{2}$$

where $\lambda_{i\cdot} = \sum_k \lambda_{ik}$ and $\lambda_{\cdot j} = \sum_k \lambda_{kj}$. We also define the load, ρ, as the quantity

$$\rho \overset{\triangle}{=} \max_{i,j}\{\lambda_{i\cdot}, \lambda_{\cdot j}\}\ .$$

Let $D_{ij}(m)$ denote the number of departures from $Q_{ij}(\cdot)$ under the scheduling algorithm[6]. Since schedule at each time has to be of matching form, by definition

$$\sum_k D_{ik}(m) \in \{0,1\}, \ \ \forall i; \quad \sum_k D_{kj}(m) \in \{0,1\}, \ \ \forall j.$$

Thus, each $Q_{ij}(\cdot)$ changes by at most 1 between successive time-slots. Further, there are at most $2n$ queues that can change between successive time-slots. This is the crucial property that we shall use to obtain lower-complexity MWM algorithms.

Now, the departures at time-slot m, $(D_{ij}(m))$, are decided by the scheduling algorithm. Let $x = (x_{ij})$ denote a matching.

[3] The algorithm's complexity scales only as a function of n, and is independent of the precise value of *weights* as long as each arithmetic operation can be performed over the weight in unit time.

[4] Ackremann's function is one of the fastest growing functions and its inverse is an extremely slowing growing function. It is safe to replace $\log(V+E)$ as an upper bound for $\alpha(E,V)$.

[5] We assume that packets always arrive at the end of a time-slot.

[6] We assume that packets always depart in the middle of a time-slot.

For an $n \times n$ switch, there are $n!$ possible such matchings. Let $W(x,m)$ denote the weight of x at time m, defined as

$$W(x,m) = \sum_{ij} x_{ij} Q_{ij}(m).$$

Then, MWM algorithm chooses, $x^*(m)$ as the schedule at time-slot m, where

$$x^*(m) = \arg\max_{x} W(x,m).$$

For a scheduling algorithm $\mathcal{A}$, denote the weight of the matching served at time m by $W_{\mathcal{A}}(m)$. Thus, for MWM algorithm, weight of matching served at time m is denoted by $W_{MWM}(m)$. We will also denote by $\pi_i(m)$ the input port i is mapped to at time m.

B. Main Results

We state our main results in this section. The corresponding algorithms are described in the later sections in detail.

Our results. Before describing our results, we first state a critical structural property of MWM that will be useful in describing our results.

(P1) In an MWM based scheduling in a switch, if E is the number of non-empty edges out of the possible n^2 edges, then $E = O(n)$ in expectation. The constant in the order can depend on the load in the switch. □

We will refer to the above as property **P1**. Note that, if $|\{(i,j) : \lambda_{ij} \neq 0\}| = O(n)$, then trivially **P1** is satisfied. In general, we state the following result.

Theorem 1.1: With MWM based scheduling,

$$E\left[\sum_{i,j} Q_{ij}\right] = O(n),$$

when $\rho < 0.5$. As a consequence, the property **P1** holds for load $\rho < 0.5$. □

Based on extensive simulations (see Figure 3 for illustration), we strongly believe the following.

Conjecture 1: The property **P1** holds for any $\rho < 1$.

Here we note that, a bound on net average queue-size of $O(n^2/(1-\rho))$ is known, for example see [6].

We now state the main result on repetitive computation of MWM.

Theorem 1.2: Given an MWM for time m, a new MWM can be computed with $O(nE + n^2)$ operations under the Switch model, where $E = |\{(i,j) : Q_{ij}(m) \neq 0\}|$ Thus, under property **P1**, we can compute a new MWM in $O(n^2)$ operations. □

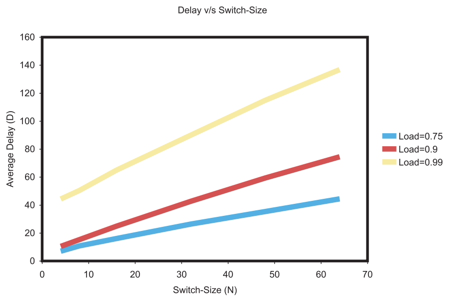

Fig. 3. Linearity of delay with respect to switch size for various loads

While a computational complexity of $O(n^2)$ is provably true for $\rho < 0.5$, we reiterate that, there are indications that property **P1** is true for arbitrary $\rho < 1$ rendering a computational complexity of $O(n^2)$ for arbitrary ρ.

Comparison with [10]. The algorithm of Gabow and Tarjan [10], under property **P1** takes $O(n^{1.5} \log^3 n)$ operations. The weaker bound of $O(n^2)$ on the net average queue-size under MWM suggests that the algorithm of Gabow and Tarjan should take $O(n^{2.5} \log^3 n)$.

The algorithm of [10], though has better analytical performance, is too complicated to implement and requires sophisticated data-structures. In contrast, as it will be clear to the reader, our algorithm requires simple operations and uses very light-weight data structures. These are attractive properties for implementation. Finally, we remark that our algorithm is by design *incremental*. For example, it takes only $O(E + n)$ operations to update MWM upon single arrivals. It is not clear if algorithm of [10] (or any other algorithm) can be simplified to make it *incremental*.

II. Incremental Algorithm for Computing MWM

This section presents an algorithm to compute MWM using information from previous time. This is based on Cunningham-Marsh algorithm [1], here we extend it for the specific instance of switch scheduling. We first present a linear programming formulation of the maximum weight matching. Then, we present algorithm and finally present proof of Theorem 1.2.

A. Linear Program: MWM

We present linear programming formulation for maximum weight matching. Let $i, j \in \{1, \ldots, n\}$ be generic indices for input and output ports of the switch respectively. Let $Q(m) = [Q_{ij}(m)]$ denote the queue-size matrix of a switch at time m. Then, the problem of finding the maximum weight matching can be described as the following integer program.

IP-MWM.

$$\begin{aligned} \text{max} \quad & \sum_{ij} Q_{ij}(m) x_{ij} \\ \text{Subject to} \quad & \sum_k x_{ik} = 1, \ \sum_k x_{kj} = 1 \\ & x_{ij} \in \{0,1\} \ \forall 1 \le i,j \le n. \end{aligned}$$

The above optimization problem is an integer program. However, it is well known that it is the linear programming (LP) relaxation of the problem LP-MWM, obtained by dropping the integrality constraints on $\{x_{ij}\}$ as described below automatically forces the solution to be integral (follows from the fact that the vertices of the polytope generated by the constraint set are integral). Hence, it is enough to solve the LP Thus, it is enough to solve the following problem.

LP-MWM.

$$\begin{aligned} \text{max} \quad & \sum_{ij} Q_{ij}(m) x_{ij} \\ \text{Subject to} \quad & \sum_k x_{ik} = 1, \ \sum_k x_{kj} = 1 \\ & x_{ij} \in [0,1] \ \forall 1 \le i,j \le n. \end{aligned}$$

We will denote the above LP-MWM as Primal LP (P-LP). Next, we consider the dual of the above problem which will be useful for describing the algorithm. Denote the r_i as the dual variable corresponding to the constraint $\{\sum_k x_{ik} = 1\}$ and p_j as the dual variable corresponding to the constraint $\{\sum_k x_{kj} = 1\}$. Then, the dual is as follows.

Dual-LP (D-LP).

$$\begin{aligned} \text{min} \quad & \sum_i r_i + \sum_j p_j \\ \text{Subject to} \quad & r_i + p_j \ge Q_{ij}(m), \ \forall 1 \le i,j \le n. \end{aligned}$$

Let $\{x^*_{ij}\}$ and $\{r^*_i, p^*_j\}$ be solutions to the above P-LP and D-LP. Then, due to the well-known fact that there is no duality gap for the above convex optimization problem, the following conditions are satisfied.

1) **Complementary slackness (CS):** $x^*_{ij} = 1 \Rightarrow r^*_i + p^*_j - Q_{ij}(m) = 0$
2) **Feasibility (F):** The $\{x^*_{ij}, r^*_i, p^*_j\}$ satisfy constraints of P-LP and D-LP.

The above discussion immediately implies the following key result that is the heart of our algorithm (as well as the algorithm of [1]).

Lemma 2.1: Let $\{x_{ij}, r_i, p_j\}$ be any tuple that satisfies conditions CS and F with $x_{ij} \in \{0,1\}, \forall i,j$. Then, $\{x_{ij}\}$ is a maximum weight matching.

In addition to Lemma 2.1, it can be shown that there exists r, p such that they are always integers when queue-size are integral (see [1]).

B. Algorithm

The Lemma 2.1 suggests that one way to find MWM is to obtain $\{x_{ij}, r_i, p_j\}$ that satisfy CS and F conditions simultaneously with $x_{ij} \in \{0,1\}$. This is precisely what our algorithm does using information from previous schedule in a clever manner. Next, we describe our algorithm, denoted by Inc-Alg.

Before formally describing Inc-Alg, we provide some intuition on its operation. Let us consider consider the case where an arrival (i.e, Q_{11} changes by '+1') occurs to Q_{11} at time m (the new queue length is $Q_{11} + 1$) and that we have a matching and the associated dual variables from time $m-1$ given by $\{x_{ij}, r_i, p_j\}$. Also suppose that edge $(1,1)$ was *not* in the matching in the previous time slot $m-1$ (if $x_{11} = 1$, clearly it will be in the matching for time slot m as well). Now, one of two cases can occur: *(i)* $(1,1)$ is not in the new MWM for time slot m, or *(ii)* $(1,1)$ is in the new MWM for time slot m.

Case (i): We first refer the reader to Figure 5. If edge $(1,1)$ is not in the new MWM, the dual variables need to be updated so that the CS and F conditions are satisfied. Due to the increased queue length, the Feasibility condition $r_1 + p_1 - (Q_{11} + 1) \ge 0$ could be violated. If so, we adopt the procedure where we add '+1' to r_1 in this case to "repair" the feasibility condition. This fix, however, will violate CS at the corresponding *matched* output port o_1, i.e. now we have $(r_1 + 1) + p_{o_1} - Q_{1o_1} < 0$. Thus, to repair this, we need to change p_{o_1} by '-1'. This in turn *could* (but not necessarily) cause a Feasibility violation at some input port, i.e, $r_{i_2} + p_{o_1} - Q_{i_2 o_2} < 0$ could occur for one (or more than one) input ports. Thus, we need to change r_{i_2} by '+1' to repair this. This process now alternates between input and output ports, and growing a *tree* in the process with alternate levels in the tree corresponding to input and output ports respectively. Alternate edges in the tree correspond to matched edges from the edges in the MWM from time $m-1$ (see Figure 4 and Fig. 5). Since we are considering *Case (i)* where the $(1,1)$ is not in the new matching, the tree will terminate *without visiting* p_1.

On the other-hand, in *Case (ii)* (see Figure 4, the tree construction will lead to the case where at some level, p_1 will need to be decremented. This clearly is feasibility violation. This is because we originally had $r_1 + p_1 - (Q_{11} + 1) < 0$ (which is why we repaired the dual variables in the above discussion). We had repaired this by updating r_1 to $(r_1 + 1)$. Now, changing $p-1$ by '-1' will again lead to a violation, i.e, $(r_1 + 1) + (p_1 - 1) - (Q_{11} + 1) < 0$. Thus, the conclusion is that the matching has changed and we should *not* have repaired all the dual variables, but instead, added edge $(1,1)$ to the new matching and simply change r_1 by '+1' from the value in the previous slot. Further, the new MWM corresponds to the 'alternate' edges in the tree construction that led to a loop in the procedure described above (see Figure 4 and Fig. 5). The above procedure is formally described in the algorithm below, both in the context of increase and decrease of the queue length.

Algorithm Inc-Alg. (computes a new matching using the matching from previous time)

- **Setup.**
  - Let $x(m) = [x_{ij}(m)]$ be the MWM at time m based on queue-size $Q(m)$ and $r(m) = [r_i(m)], p(m) = [p_j(m)]$ be solutions to D-LP such that $\{x(m), r(m), p(m)\}$ satisfy CS and F.
  - The departures happen from non-empty queues according to $x(m)$ and new arrival happens to at most n queues. Let $Q(m+1)$ be queue-size at time $m+1$.
  - Let $\ell \leq 2n$ be the total queues that have changed in their sizes by $+1$ or -1 at time $m+1$ compared to time m. Let they be denoted by $(i_1, j_1), \dots, (i_\ell, j_\ell)$.
  - Let $Q^0 = Q(m)$. Obtain $Q^k, 1 \leq k \leq \ell$ by adding the ℓ changes in the queue-sizes one by one. By construction, Q^k and Q^{k+1} differ in only one queue being different by $+1$ or -1. As per notation, $Q^\ell = Q(m+1)$.
  - Let $\{x^k, r^k, p^k\}$ be tuple satisfying CS and F for P-LP and D-LP for weight given by Q^k, for $0 \leq k \leq \ell$. Note that, $\{x^0, r^0, p^0\} = \{x(m), r(m), p(m)\}$.
- **Compute x(m+1).**
  - For $k = 1, \dots, \ell$, compute $\{x^k, r^k, p^k\}$ using ONE-STEP with inputs $\{x^{k-1}, r^{k-1}, p^{k-1}\}$, where routine ONE-STEP is described below.
  - The $x(m+1) = x^\ell$ by definition.

Now, we describe the ONE-STEP routine that computes the solution when only one queue changes by $+1$ or -1.

Algorithm ONE-STEP (*computes a new matching when a single queue changes by* $+1/-1$)

SETUP:

- Input $\{x, r, p\}$ satisfy CS and F for P-LP and D-LP for MWM with weight as queue-size matrix Q.
- Let one of the queues changes by $+1$ or -1 in Q to give $\hat{Q}$. Without loss of generality, let the queue that changes be Q_{11}, that is, $\hat{Q}_{11} = Q_{11} \pm 1$.
- Let the set of non-empty edges, $\mathcal{E} = \{(i,j) : Q_{ij} \neq 0\}$.
- Define $\mathcal{O}(i) = \{j : r_i + p_j - Q_{ij} = 0\}$ and $\mathcal{I}(j) = \{i : r_i + p_j - Q_{ij} = 0\}$.

CASE-1: $\mathbf{\hat{Q}_{11} = Q_{11} + 1}$.

- If $x_{11} = 1$, then set $r_1 = r_1 + 1$. Return this modified $\{x, r, p\}$ as the output.
- Else, if $x_{11} = 0$ and $r_1 + p_1 - Q_{11} \geq 1$. Return the same $\{x, r, p\}$ as the output.
- Else, $x_{11} = 0$ and $r_1 + p_1 - Q_{11} = 0$. In this case, the F condition for D-LP is violate for $(1,1)$. We need to compute new $\{\tilde{x}, \tilde{r}, \tilde{p}\}$ using the $\{x, r, p\}$ that will satisfy the CS and F conditions for $\hat{Q}$. We do so by growing appropriate "tree" structure described below.
- Initialize $\hat{r}_i = r_i$ and $\hat{p}_j = p_j$ for all $1 \leq i, j \leq n$. Set $\hat{r}_1 = \hat{r}_1 + 1$. The $\{\hat{r}, \hat{p}\}$ are intermediate variables.
- Create a tree T_0 that contains input 1 and output 1 and edge (1,1). Set input 1 as leaf of the tree and output 1 as root of the tree.
- **ITER.** For $k \geq 0$, do the following till one can not grow tree any more or required to go to AUGMENT step.
  - When k is *even*: let v be any of the leaf of tree T_k. By construction, each leaf of T_k is input vertex for even k. For each such v, do the following steps:
    1) Let u be such that $x_{vu} = 1$, that is (v, u) is an edge in MWM according to old x.
    2) If u is output 1, then go to AUGMENT. Else, set $\hat{p}_u = \hat{p}_u - 1$. This satisfies the CS for edge (v, u).
    3) Add u and edge (v, u) to the tree T_k.
  - Set modified tree T_k as T_{k+1} and $k = k + 1$. Go to the step ITER.
  - When k is *odd*: let u be any of the leaf of tree T_k. By construction, each leaf of T_k is output vertex for odd k. For each such u, do the following steps (1)-(2).

    (1) Consider any $v \in \mathcal{I}(u)$. If v is already in the T_k ignore it.

    (2) Else if v is not in T_k, then the F condition for D-LP is not satisfied for $(x, \hat{r}, \hat{p})$. Add v and (u, v) to the T_k and set $\hat{r}_v = \hat{r}_v + 1$.
  - Set modified tree T_k as T_{k+1} and $k = k + 1$. Go to the step ITER.
- Set $\tilde{x} = x$, $\tilde{r} = \hat{r}$ and $\tilde{p} = \hat{p}$. Return $\{\tilde{x}, \tilde{r}, \tilde{p}\}$ as the output.
- **AUGMENT.** Let at stage k (even), an input vertex v be such that $x_{v1} = 1$.
  - Let the path in tree T_k starting from input 1 to vertex v be $(v_0 = 1, u_1, v_2, u_3, \dots, u_{k-1}, v_k = v)$.
  - Now, consider a cycle, which is extension of this path, $(v_0 = 1, u_1, \dots, u_{k-1}, v_k = v, \text{(output)}1, v_0)$. In this cycle, the alternate edges belong to matching according to x, that is $x_{v_{2s}u_{2s+1}} = 1, 0 \leq s \leq k/2 - 1$ and $x_{v_k 1} = 1$.
  - Now, create a new matching $\tilde{x}$ as follows. Set $\tilde{x}_{v_0 1} = 1$ and $\tilde{x}_{v_{2s}u_{2s-1}} = 1$. For all other (i, j) such that $x_{ij} = 1$, set $\tilde{x}_{ij} = 1$ while set all remaining $\tilde{x}_{ij} = 0$.
  - Set $\tilde{r} = r$ and $\tilde{p} = p$ for all i, j but set $\tilde{r}_1 = r_1 + 1$.
  - Return $\{\tilde{x}, \tilde{r}, \tilde{p}\}$ as output.

CASE-2: $\mathbf{\hat{Q}_{11} = Q_{11} - 1}$.

- If $x_{11} = 0$ then return this modified $\{x, r, p\}$ as the output.
- Else, if $x_{11} = 1$ then by definition $r_1 + p_1 - Q_{11} = 0$. Thus, the F condition for D-LP is violate for $(1,1)$ as $\hat{Q}_1 1 = Q_{11} - 1$. We need to compute new $\{\tilde{x}, \tilde{r}, \tilde{p}\}$ using the $\{x, r, p\}$ that will satisfy the CS and F conditions for $\hat{Q}$. This is done in exactly the same manner as done for the case when $\hat{Q}_{11} = Q_{11} + 1$. We describe the first few

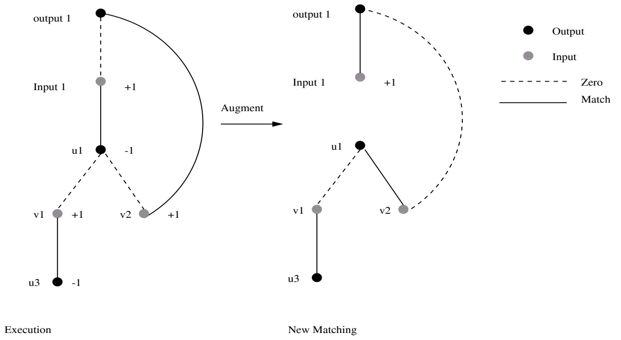

Fig. 4. Example of procedure ONE-STEP when Q_{11} increases by 1. Here the AUGMENT step is invoked and MWM changes as shown.

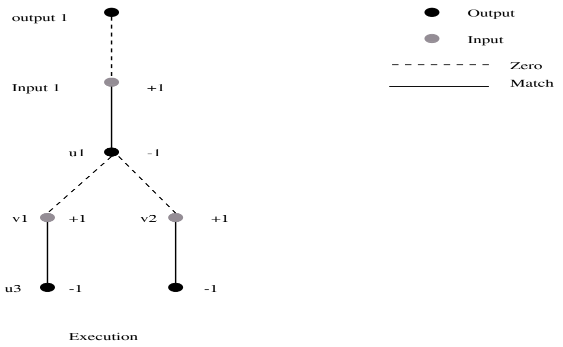

Fig. 5. Example of procedure ONE-STEP when Q_{11} increases by 1. Here the matching remains the same however dual variables are updated as shown.

steps as follows.

- Initialize $\hat{r}_i = r_i$ and $\hat{p}_j = p_j$ for all $1 \le i, j \le n$. Set $\hat{p}_1 = \hat{p}_1 - 1$. The $\{\hat{r}, \hat{p}\}$ are intermediate variables.
- Create a tree T_1 that contains input 1 and output 1 and edge (1,1). Set input 1 as root of the tree and output 1 as leaf of the tree. Now this is the same as T_1 of the previous case. The only difference is we will AUGMENT when there is an edge between some output vertex and input 1 that needs to be added to the tree. We skip further details.

The description of the algorithm seems rather complicated due to notation based description. It is better understood via an example with help of a figure. Consider case when $\hat{Q}_{11} = Q_{11} + 1$. There are mainly two cases in algorithm: (a) The MWM is changed when AUGMENT step is invoked and only r_1 increases by 1, or (b) the MWM remains the same but many dual variables, r, p change by $+1$ and -1 in an appropriate manner. The Figure 4 describes the case (a), while the Figure 5 describes the case (b).

C. Analysis of Algorithm

We state the following result about algorithm Inc-Alg. The Theorem 2.1, as stated below, immediately implies Theorem 1.2.

Theorem 2.1: Algorithm Inc-Alg produces solution $\{x(m+1), r(m+1), p(m+1)\}$ that satisfies CS and F for $Q(m+1)$ given $\{x(m), r(m), p(m)\}$ satisfying CS and F for $Q(m)$. Further, it can be implemented such that the total number of operations performed by algorithm is $O(nE+n^2)$, where $E = |\mathcal{E}|$, the number of non-empty edges with respect to $Q(m)$. Due to space constraints, we skip the proof details.

III. Conclusion

In this paper, we have demonstrated that, in an $n \times n$ switch, MWM can be computed using $O(n^2)$ operations if the MWM from the previous time slot is taken into account, and with suitable assumptions on system load. As the best known algorithm for ab-initio computing MWM has a complexity of $O(n^3)$, our work shows that there can substantial gains in complexity if we take the dynamics of the system into account. We believe that the average complexity results in this paper can be extended to a "high probability" kind of complexity.

Acknowledgments

This research was partially supported by NSF Grants ACI-0305644, CNS-0325788, CNS-0347400 and CNS-0519401.

References

[1] W. H. Cunningham and A. B. Marsh, "A primal algorithm for optimum matching," *Mathematical Programming*, pp. 50–72, 1978.

[2] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, pp. 1347–1356, Dec 1987.

[3] N. McKeown, V. Anantharan, and J. Walrand, "Achieving 100input-queued switch," in *Proceedings of IEEE INFOCOM*, 1996.

[4] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1949, 1992.

[5] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, August 1999.

[6] D. Shah, "Randomization and heavy traffic: new approaches for switch algorithms," Ph.D. dissertation, Computer Science Department, Stanford University, 2004.

[7] L. Tassiulas, "Adaptive back-pressure congestion control based on local information," *IEEE Transactions on Automatic Control*, vol. 40, no. x2, pp. 236–250, 1995.

[8] A. Stolyar, "Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic," *Annals of Applied Probability*, vol. 14, no. 1, pp. 1–53, 2004.

[9] M. Marsan, P. Giaccone, E. Leonardi, and F. Neri, "On the stability of local scheduling policies in networks of packet switches with input queues," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 642–655, May 2003.

[10] H. N. Gablow and R. E. Tarjan, "Faster scaling algorithms for general graph matching problems," *Journal of the ACM*, vol. 38, no. 4, pp. 815 – 853, 1991.

[11] N. McKeown, "Scheduling algorithms for input-queued switches," Ph.D. dissertation, Department of EECS, UC Berkeley, 1995.

[12] M. Marsan, M. Ajmone, A. Bianco, E. Leonardi, and L. Milia, "Rpa: A flexible scheduling algorithm for input buffered switches," *IEEE Transactions on Communications*, vol. 47, pp. 1921–1933, December 1999.

[13] H. Duan, J. W. Lockwood, S. M. Kang, and J. D. Will, "A high performance oc12/oc48 queue design prototype for input buffered atm switches," in *Proceedings of IEEE INFOCOM97*, vol. 1, 1997, pp. 20–28.

[14] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for vlsi communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 13–27, Jan 1993.

[15] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proceedings of IEEE INFOCOM*, 1998.

[16] P. Giaccone, B. Prabhakar, and D. Shah, "Towards simple, high-performance schedulers for high-aggregate bandwidth," in *Proceedings of IEEE INFOCOM*, 2002.