Network scheduling and message-passing

Devavrat Shah

Abstract Algorithms are operational building-blocks of a network. An important class of network algorithms deal with the scheduling of common resources among various entities such as packets or flows. In a generic setup, such algorithms operate under stringent hardware, time, power or energy constraints. Therefore, algorithms have to be extremely simple, lightweight in data-structure and distributed. Therefore, a network algorithm designer is usually faced with the task of resolving an acute tension between performance and implementability of the algorithm. In this chapter, we survey recent results on novel design and analysis methods for simple, distributed aka message-passing scheduling algorithms. We describe how the asymptotic analysis methods like fluid model and heavy traffic naturally come together with algorithm design methods such as randomization and belief-propagation (message-passing heuristic) in the context of network scheduling.

1 Introduction

We consider a queuing network in which there are constraints on which queues may be served simultaneously. Specifically, consider a collection of queues operating in discrete time. In each timeslot, queues are offered service according to a *service schedule* chosen from a specified set. For example, in a three-queue system, the set of allowed schedules might consist of "Serve 3 units of work each from queues A & B" and "Serve 1 unit of work each from queues A & C, and 2 units from queue B". New work may arrive in each timeslot; let each queue have a dedicated exogenous arrival process, with specified mean arrival rates. Once the work is served, it leaves the network.

This general model has been used to describe a wireless network in which radio interference limits the amount of service that can be given to each host. It has

Devavrat Shah
Massachusetts Institute of Technology, Cambridge MA 02139, e-mail: devavrat@mit.edu

been used to describe an input-queued switch, the device at the heart of high-end Internet routers, whose underlying silicon architecture imposes constraints on which traffic streams can be transmitted simultaneously. It has been used to describe the overall operation of congestion control in the Internet, whereby TCP assigns transmission rates to flows, subject to network constraints about the capacity on shared links. It can even describe road junctions, where the schedule of which lanes can move is controlled by traffic lights. Our general model is described in more detail in Section 2. We will use examples of switch scheduling and wireless scheduling throughout. We select these two examples, because (1) switch is the simplest non-trivial example of general scheduling problem and of great practical importance, (2) wireless network with scheduling constraints characterized by independent set on interference graph encapsulates a large class of scheduling problems.

We give the name *scheduling algorithm* to the procedure whereby a schedule is chosen for each timeslot. In such a setup, the basic question is about the characterization of an *optimal* scheduling algorithm. But before that, we need to understand the notion of *optimality*. In order to define *optimality*, we consider two important network performance metrics: throughput and delay or backlog (queue-size) on average[1]. Roughly speaking, throughput optimality corresponds to utilizing the network capacity to the fullest. Equivalently, an algorithm is called throughput optimal if whenever the network is *underloaded* the backlog in the network remains finite. The delay optimality of an algorithm is usually defined as the minimization of various norms of delay or queue-size. We will focus on minimization of the net queue-size in this chapter.

First, we will address the question of characterizing throughput and delay (queue-size) optimal algorithm. It should be noted that answering this question is quite non-trivial. The primary reason is that the algorithm has to be *online* (i.e. use only network-state information like queue-size or age of packet). However, the performance metrics like throughput and average queue-size are determined by the behavior of the network system over the entire time horizon (in principle infinite) it is operating. We will start with the description of the popular maximum-weight scheduling, called MW, introduced by Tassiulas and Ephremides [31]. It assigns a weight to each schedule, from summing the lengths of the queues that schedule proposes to serve, and choses the schedule with the largest weight. It was shown that the MW algorithm is throughput optimal for the general network model considered in this chapter (see Section 2). The MW algorithm is also known to induce reasonable (polynomial in network size) average queue-size for this model. But, it is not necessarily optimal in terms of average queue-size.

To understand the optimal algorithm both in terms of throughput and average queue-size, Keslassy and McKeown [14] considered a class of MW algorithms, called MW-α for $\alpha > 0$. Like MW algorithm, MW-α algorithm also uses the schedule with maximum weight. However, MW-α uses the queue-size to power α as weight instead of plain queue-size. Clearly, the MW algorithm is MW-α algorithm when $\alpha = 1$. The MW-α algorithm is throughput optimal for all $\alpha > 0$. The natural

[1] Due to general result like Little's law for stable system, we will use delay and queue-size or backlog interchangeably throughout.

question is: how does queue-size change with value of α? In [14], through an extensive empirical study in the context of input-queued switch, it was found that the average queue-size decreases monotonically as $\alpha \to 0^+$. This led them to conjecture that MW-0^+ is optimal in the class of MW-α algorithms, $\alpha > 0$. In Section 3, we will provide partial justification to this claim using the critical fluid model of the network. Specifically, an algorithm is called queue-size optimal if it induces minimal queue-size in its critical fluid model. The justification provided in this chapter shows that the MW-0^+ algorithm (i.e. limit of MW-α algorithm as $\alpha \to 0^+$) is queue-size optimal among all the scheduling algorithm, not just in the class of MW-α algorithm, in this critical fluid model. This justification holds for the general network model of this chapter, not just for input-queued switch. This result was recently established by Shah and Wischik [26].

The characterization of MW-0^+ as an optimal algorithm suggests that finding a good schedule requires solving a certain global (network-wide) combinatorial optimization problem every time. In order to be implementable, this necessitates the design of simple and distributed algorithms for such combinatorial optimization problems. In the second part of this chapter, we describe an extremely simple, randomized message-passing scheduling algorithm that is shown to be throughput optimal essentially for all known examples. This algorithm uses clever distributed summation algorithm along with a simple random sampler. The algorithm will be explained in detail through examples of switch scheduling and wireless scheduling in Section 4.

This randomized algorithm, though simple and throughput optimal, can induce very large (exponential in size of the network) average queue-size. Now, when scheduling constraints have simple structure (e.g. matching in switches), the algorithm performs very well even in terms of queue-size. However, when scheduling constraints have complex structure (e.g. independent set in wireless network), the algorithm induces exponentially (in network size) large queue-size. More generally, impossibility of any simple (polynomial time) centralized or distributed algorithm, that is throughput optimal and has low (polynomial size) delay, is established when scheduling constraints are complex enough (e.g. independent set) (see recent result by Shah, Tse and Tsitsiklis [24]).

Therefore, a pragmatic approach is to design simple, distributed algorithm that provides terrific performance when the underlying problem structure is simple and works as a reasonable heuristic when problem structure is hard. In the last part of this chapter in Section 5, we describe such an algorithm design method based on belief propagation (BP). The BP has recently emerged as a very successful heuristic for hard combinatorial problems. We present BP based scheduling algorithm for switches (matching) and wireless networks (independent set). These algorithms are exact when underlying problem have certain LP relaxation tight; and work as a reasonable heuristic otherwise.

We take note of the limitation of this chapter in that there is a lot of exciting work done in the past decade or two on the topic of network scheduling (e.g.[2, 11, 29, 20, 27]) and it is not discussed here for natural reasons. An interested reader is strongly encouraged to explore these results.

2 Model

This section describes an abstract model of the queuing network that we will consider. Though the model described here corresponds to *single-hop* network for convenience, most of the analytic and algorithmic results stated here should apply for general *multi-hop* network with appropriate changes. The examples of switch and wireless scheduling, which are special cases of the model, are described in detail and will be useful throughout the chapter.

2.1 Abstract formulation

Consider a collection of N queues. Let time be discrete, indexed by $\tau \in \{0,1,\dots\}$. Let $Q_n(\tau)$ be the size of queue n at the beginning of timeslot τ, and write $\mathbf{Q}(\tau)$ for the vector $[Q_n(\tau)]_{1\le n\le N}$. Let $\mathbf{Q}(0)$ be the prespecified vector of initial queue sizes.

In each timeslot, each queue is offered service subject to a *scheduling constraint* described below. If the amount of service offered to a queue is larger than the queue size, then we say that the queue has *idled*, otherwise it does not idle. Once work is served it leaves the network. New work may arrive in each timeslot; let each of the N queues have a dedicated exogenous arrival process.

The scheduling constraint is described by a finite set of *feasible schedules* $\mathscr{S} \subset \mathbb{R}_+^N$. In every timeslot a *schedule* $\pi \in \mathscr{S}$ is chosen; queue n is offered an amount of service π_n in that timeslot. For example, in the case of input-queued switch, $\mathscr{S}$ is the set of all matchings between input ports and output ports; in the case of wireless network, $\mathscr{S}$ is the set of independent sets in the interference graph. For simplicity, we will restrict ourselves to $\mathscr{S}$ such that $\mathscr{S} \subset \{0,1\}^N$; that is, for any $\pi \in \mathscr{S}$, $\pi_n = 1$ (queue n has received unit amount of service) or 0 (queue n receives no service). We will also assume that $\mathscr{S}$ is *monotone*: if $\pi \in \mathscr{S}$ then for any $\sigma \le \pi$ component-wise, i.e. $\sigma_n \le \pi_n$ for all n, $\sigma \in \mathscr{S}$.

Let $S_\pi(\tau) \in \mathbb{R}_+$ be the total length of time up to the end of timeslot τ in which schedule π has been chosen, and let $S_\pi(0) = 0$. Let $Z_n(\tau)$ be the total amount of idling at queue n up to the end of timeslot τ, and let $Z_n(0) = 0$. Let $A_n(\tau)$ be the total amount of work arriving to queue n up to the end of timeslot τ, and $A_n(0) = 0$. We will take $\mathbf{A}(\cdot)$ to be a random process. Define the arrival rate vector λ by

$$\lambda_n = \lim_{\tau\to\infty} \frac{1}{\tau} A_n(\tau) \tag{1}$$

and assume that this limit exists almost surely for each queue. For simplicity, we will assume the following about the arrival process: $A_n(\tau+1) - A_n(\tau)$ are independent across τ and n, identically distributed for a given n but different τ, have support on integer values only and are bounded. A simplest example of the above is $A_n(\cdot)$ being Bernoulli i. i. d. process with parameter λ_n.

We will use the convention that $\mathbf{Q}(\tau)$ is the vector of queue sizes at the beginning of timeslot τ, and then the schedule for timeslot τ is chosen and service happens, and then arrivals for timeslot τ happen. Define the cumulative offered service vector $\Sigma(\tau) = [\Sigma_n(\tau)]$ as $\Sigma(\tau) = \sum_{\pi \in \mathscr{S}} \pi S_\pi(\tau)$. Then,

$$Q_n(\tau) = Q_n(0) + A_n(\tau) - \Sigma_n(\tau) + Z_n(\tau) \tag{2}$$

$$Z_n(\tau) - Z_n(\tau - 1) = \max\Big(0, \Sigma_n(\tau) - \Sigma_n(\tau - 1) - Q_n(\tau - 1)\Big) \tag{3}$$

2.1.1 Notation

Finally, some notation. We will reserve bold letters for vectors in $\mathbb{R}^N$. Let $\mathbf{0}$ be the vector of all 0s, and $\mathbf{1}$ be the vector of all 1s. Let $1_{\{\cdot\}}$ be the indicator function, $1_{\text{true}} = 1$ and $1_{\text{false}} = 0$. Let $x \wedge y = \min(x,y)$ and $x \vee y = \max(x,y)$ and $[x]^+ = x \vee 0$. When x is a vector, the maximum is taken componentwise. Use the norm $|x| = \max_i |x_i|$ for vectors x. For vectors $\mathbf{u}$ and $\mathbf{v}$ and functions $f : \mathbb{R} \to \mathbb{R}$, let

$$\mathbf{u} \cdot \mathbf{v} = \sum_{n=1}^{N} u_n v_n, \quad \text{and} \quad f(\mathbf{u}) = \big[f(u_n)\big]_{1 \le n \le N}$$

Let $\mathbb{N}$ be the set of natural numbers $\{1, 2, \dots\}$, let $\mathbb{Z}_+ = \{0, 1, 2, \dots\}$, let $\mathbb{R}$ be the set of real numbers, and let $\mathbb{R}_+ = \{x \in \mathbb{R} : x \ge 0\}$.

2.2 Scheduling algorithms

For our purposes, one scheduling algorithm is particularly interesting: the Maximum Weight (MW) scheduling algorithm, which works as follows. Let $\mathbf{Q}(\tau)$ be the vector of queue sizes at the beginning of timeslot τ. Define the weight of a schedule $\pi \in \mathscr{S}$ to be $\pi \cdot \mathbf{Q}(\tau)$. The algorithm then chooses for timeslot τ a scheduling with the greatest weight (breaking ties arbitrarily). This algorithm can be generalized to choose a schedule which maximizes $\pi \cdot \mathbf{Q}(\tau)^\alpha$, where the exponent is taken componentwise for some $\alpha > 0$; call this the MW-α algorithm. In this paper, we will study the MW-α algorithm in detail. More generally, one could choose a schedule π such that

$$\pi \cdot f(\mathbf{Q}(\tau)) = \max_{\rho \in \mathscr{S}} \rho \cdot f(\mathbf{Q}(\tau)) \tag{4}$$

for some function $f : \mathbb{R}_+ \to \mathbb{R}_+$; call this the MW-$f$ algorithm. We will assume the following about f.

Assumption 1 *Assume f is differentiable and strictly increasing with $f(0) = 0$. Assume also that for any $\mathbf{q} \in \mathbb{R}_+^N$ and $\pi \in \mathscr{S}$, with $m(\mathbf{q}) = \max_{\rho \in \mathscr{S}} \rho \cdot f(\mathbf{q})$,*

$$\pi \cdot f(\mathbf{q}) = m(\mathbf{q}) \quad \Longrightarrow \quad \pi \cdot f(\kappa \mathbf{q}) = m(\kappa \mathbf{q}) \ \text{ for all } \kappa \in \mathbb{R}_+.$$

The MW-f algorithm is *myopic*, i.e. it chooses a schedule based only on the current queue sizes and doesn't need to try to learn traffic parameters etc. An important reason for the popularity of the MW algorithm is that MW-f is the only class of myopic scheduling algorithms known to have the largest possible stability region, for a large class of constrained scheduling problems. The MW algorithm was first proposed by Tassiulas and Ephremides [31]. Later, it was proposed by McKeown, Ananthram and Walrand in the context of switches [16]. The MW-f algorithm has been studied in detail by various researchers, including [22, 14, 1].

2.3 Input-queued switch

Here we describe input-queue switch as a special instance of the abstract network model. An Internet router has several input ports and output ports. A data transmission cable is attached to each of these ports. Packets arrive at the input ports. The function of the router is to work out which output port each packet should go to, and to transfer packets to the correct output ports. This last function is called *switching*. There are a number of possible switch architectures; we will consider the commercially popular input-queued switch architecture.

Figure 1 illustrates an input-queued switch with three input ports and three output ports. Packets arriving at input i destined for output j are stored at input port i, in queue $Q_{i,j}$. The switch operates in discrete time. In each timeslot, the switch fabric can transmit a number of packets from input ports to output ports, subject to the two constraints that each input can transmit at most one packet and that each output can receive at most one packet. In other words, at each timeslot the switch can choose a *matching* from inputs to outputs. Figure 1 shows two possible matchings. In the left hand figure, the matching allows a packet to be transmitted from input port 3 to output port 2, but since $Q_{3,2}$ is empty no packet is actually transmitted. The specific matching of inputs to outputs in each timeslot is chosen by the scheduling algorithm.

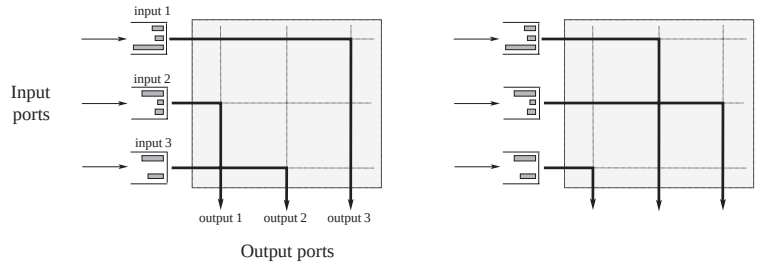

Fig. 1 An input-queued switch, and two example matching of inputs to outputs.

To connect back to the abstract formulation, note that an N-port switch has $n = N^2$ queues; in the context of switch, we will always use notation of Q_{ij} instead of

notation of Q_n so as to be clear in referencing the corresponding input and output for a queue. The set of all feasible schedules $\mathscr{S}$ correspond to the set of all complete matchings in an $N \times N$ complete bipartite graph. Formally,

$$\mathscr{S} = \left\{ \pi = [\pi_{ij}] \in \{0,1\}^{N\times N} : \sum_{k=1}^{N} \pi_{ik} = 1, 1 \le i \le N; \sum_{k=1}^{N} \pi_{kj} = 1, 1 \le j \le N \right\}.$$

The MW (or MW-1) scheduling algorithm, therefore chooses a matching as the schedule that is one of the possibly many solutions of the following combinatorial optimization problem: let $\mathbf{Q}(\tau) = [Q_{ij}(\tau)]$ be queue-sizes at timeslot τ, then the schedule $\pi(\tau)$ at timeslot τ solves:

$$\begin{aligned} &\text{maximize} \quad \sum_{i,j=1}^{N} \pi_{ij} Q_{ij}(\tau) \quad \text{over} \quad \pi_{ij} \in \{0,1\}, 1 \le i,j \le N \\ &\text{subject to} \quad \sum_{k=1}^{N} \pi_{ik} = 1,\ 1 \le i \le N; \quad \sum_{k=1}^{N} \pi_{kj} = 1,\ 1 \le j \le N. \end{aligned} \tag{5}$$

The above optimization problem is the well-known Maximum Weight Matching(MWM) problem. Therefore, designing MW scheduling algorithm will involve solving this optimization problem every timeslot.

Some notes on the input-queued switch architecture. (1) We have illustrated a switch with as many inputs as outputs. It may be that some of these do not actually carry any traffic; thus there is no loss of generality in assuming as many inputs as outputs. (2) In the Internet, packets may have different sizes. Before the packet reaches the switch fabric, it will be fragmented into a collection of smaller packets (called cells) of fixed size. (3) There will typically be a block of memory at each input port for the queues, and one packet's worth of memory at each output port to hold the packet as it is serialized onto the outgoing cable. Memory access speeds are a limiting factor, and indeed the time it takes to read or write a packet from memory is what determines the length of a timeslot. There are switches which perform several matchings per timeslot—but then the timeslots need to last several times longer, to give time for the extra reads and writes.

2.4 Wireless networks

Consider several wifi networks close to each other and sharing the same frequency. If two devices close to each other transmit at the same time, then there is interference and the data may be lost; whereas two devices far from each other may successfully transmit at the same time. A popular way to model this sort of interference is to draw a graph with a node for each device and an edge between two nodes if they can interfere with each other; in other words a transmission from a node is successful

only if none of its neighbors in the network graph is transmitting at the same time. (This is called the *independent set* model for interference.)

Figure 2 illustrates a wireless network of three nodes operating under interference constraints. Here, like switch packets arriving at node i and destined for node j are stored at node i in queue $Q_{i,j}$. For example, the queue $Q_{2,3}$ shown in the Figure 2 stores packet destined for node 3 arrived at node 2. The independent set constraint on scheduling, in our example, implies that at a given timeslot at most one of the three node can transmit. Figure 2 shows that node 3 is selected for transmission while other two nodes remain silent; and node 3 transmits packet to node 2.

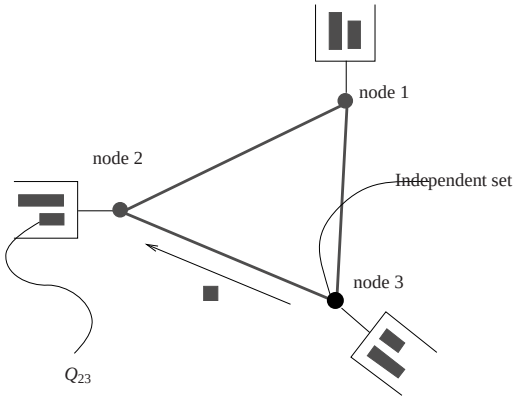

Fig. 2 A wireless network operating under interference constraint, and an example independent set (node 3) schedule for transmission from 3 to 2.

In a general wireless network, the interference network graph is represented as $G = (V,E)$ with vertices corresponding to nodes, $V = \{1,\ldots,N\}$ and edges corresponding to interference, that is

$$E = \{(i,j) : i \text{ and } j \text{ interfere with each other}\}.$$

An implicit assumption here is that any node $i \in V$ can transmit packets to node j only if $(i,j) \in E$. We will consider a single-hop wireless network, that is each packet arrives at one node, gets transmitted to one of its neighbors and then departs the network. In such setup, if a node i transmits (to any of its neighbors), then all of i's neighbors must be silent. Therefore, the real constraint lies in deciding whether i transmits or not; but not in which node it transmits. Therefore, for simplicity we will ignore the finer classification of queue-sizes as Q_{ij} but instead consider $Q_i = \sum_j Q_{ij}$ here for the purpose of scheduling. In this setup, the set of schedules is the set of independent sets in G. Formally,

$$\mathscr{S} = \left\{\sigma = [\sigma_i] \in \{0,1\}^N : \sigma_i + \sigma_j \leq 1, \text{ for all } (i,j) \in E\right\}.$$

The MW (or MW-1) scheduling algorithm, therefore chooses an independent set as the schedule that is one of the possibly many solutions of the following combinatorial optimization problem: let $\mathbf{Q}(\tau) = [Q_i(\tau)]$ be queue-sizes at timeslot τ, then the schedule $\sigma(\tau)$ at timeslot τ solves:

$$\begin{aligned} &\text{maximize} \quad \sum_{i}^{N} \sigma_i Q_i(\tau) \quad \text{over} \quad \sigma_i \in \{0,1\}, 1 \leq i \leq N, \\ &\text{subject to} \quad \sigma_i + \sigma_j \leq 1, \text{ for all } (i,j) \in E. \end{aligned} \tag{6}$$

The above optimization problem is the well-known Maximum Weight Independent Set(MWIS) problem. Therefore, designing MW scheduling algorithm will involve solving this optimization problem every timeslot.

Some notes on the described wireless network. (1) In the multi-hop network, the MW scheduling corresponds to solving MWIS problem with somewhat different weights (e.g. Back-pressure algorithm [31]). Therefore, for the purpose of scheduling algorithm design, the above model captures the essence. (2) The above describe model of interference is general in the following sense. Many of the combinatorial interference models such as 2-hop matching model (secondary interference model) can be represented as an independent set model by representing transmission edges as nodes; and such transformations are computationally equivalent (formally, they are reductions). (3) We note that there are other models of interference (e.g. SINR model) that can not be captures by *hard* constraints of the type of independent set model.

3 Characterization of optimal algorithm

This section presents characterization of optimal scheduling algorithms first in terms of throughput and then in terms of queue-size. The algorithms considered here are part of the maximum weight (MW) family. Interestingly enough, as explained in this section, considering this class of algorithm is sufficient for the purpose of finding throughput and queue-size optimal algorithm. This section will utilize fluid-model technique. The contents of this section can be found in a recent paper by Shah and Wischik [26]. We note that fluid model for MW scheduling algorithm was first introduced in the context of switch by Dai and Prabhakar [9] and later used by Andrews et. al. [1].

3.1 Throughput optimality

Here, we will establish that all the MW-f algorithms are throughput optimal as long as f satisfies Assumption 1. First, some necessary definitions.

Admissible arrival rates. At each timeslot, a schedule $\pi \in \mathscr{S}$ must be chosen. Let Θ be the convex hull of $\mathscr{S}$,

$$\Theta = \Big\{ \sum_{\pi\in\mathscr{S}} \alpha_\pi \pi : \sum_{\pi\in\mathscr{S}} \alpha_\pi = 1, \text{ and } \alpha_\pi \geq 0 \text{ for all } \pi \Big\}. \tag{7}$$

We say that an arrival rate vector λ is *admissible* if $\lambda \in \Lambda$ where

$$\Lambda = \Big\{ \lambda \in \mathbb{R}_+^N : \lambda \leq \sigma \text{ componentwise, for some } \sigma \in \Theta \Big\}. \tag{8}$$

Intuitively, this means that there is some combination of feasible schedules which permits all incoming work to be served. Also define

$$\Lambda^\circ = \Big\{ \lambda \in \Lambda : \lambda \leq \sum_{\pi\in\mathscr{S}} \alpha_\pi \pi, \text{ where } \sum_{\pi\in\mathscr{S}} \alpha_\pi < 1 \text{ and } \alpha_\pi \geq 0 \text{ for all } \pi \Big\}$$
$$\partial\Lambda = \Lambda \setminus \Lambda^\circ.$$

Say that λ is *strictly admissible* if $\lambda \in \Lambda^\circ$, and that λ is *critical* if $\lambda \in \partial\Lambda$.

Fluid model. The fluid model is essentially the first-order deterministic description of the network. To obtain this formally, we need to consider the fluid scaling of the original system. Next, we describe a scaling procedure to obtain a sequence of (network) systems from the given system, indexed by $r \in \mathbb{N}$. Write $X^r(\tau) = (\mathbf{Q}^r(\tau), \mathbf{A}^r(\tau), \mathbf{Z}^r(\tau), S^r(\tau))$, $\tau \in \mathbb{Z}_+$, for the rth system. Define the scaled system $x^r(t) = (\mathbf{q}^r(t), \mathbf{a}^r(t), \mathbf{z}^r(t), s^r(t))$ for $t \in \mathbb{R}_+$ by

$$\mathbf{q}^r(t) = \mathbf{Q}^r(rt)/r \qquad \mathbf{a}^r(t) = \mathbf{A}^r(rt)/r$$
$$\mathbf{z}^r(t) = \mathbf{Z}^r(rt)/r \qquad s_\pi^r(t) = S_\pi^r(r't)/r'$$

after extending the domain of $X^r(\cdot)$ to $\mathbb{R}_+$ by linear interpolation in each interval $(\tau-1, \tau)$. We describe fluid model equations, which are essentially satisfied by limiting system $X^r(\cdot)$ as $r \to \infty$. We say that the process $x(\cdot) = (\mathbf{q}(\cdot), \mathbf{a}(\cdot), \mathbf{z}(\cdot), s(\cdot))$ satisfies the fluid model for the MW-f scheduling algorithm if

$$\mathbf{a}(t) = \lambda t \tag{9}$$
$$\mathbf{q}(t) = \mathbf{q}(0) + \mathbf{a}(t) - \sum_\pi s_\pi(t)\pi + \mathbf{z}(t) \tag{10}$$
$$\sum_{\pi\in\mathscr{S}} s_\pi(t) = t \tag{11}$$
$$\text{each } s_\pi(\cdot) \text{ and } z_n(\cdot) \text{ is increasing (not necessarily strictly increasing)} \tag{12}$$
$$\text{all the components of } x(\cdot) \text{ are absolutely continuous} \tag{13}$$
$$\text{for almost all } t, \text{ all } n, \quad \dot{z}_n(t) = 0 \text{ if } q_n(t) > 0 \tag{14}$$
$$\text{for almost all } t, \text{ all } \pi \in \mathscr{S}, \quad \dot{s}_\pi(t) = 0 \text{ if } \pi \cdot f(\mathbf{q}(t)) < \max_{\rho\in\mathscr{S}} \rho \cdot f(\mathbf{q}(t)) \tag{15}$$

Our goal is to establish that the dynamics of $x^r(t)$, for t in a fixed interval $[0,T]$, as $r \to \infty$ satisfies the above stated fluid model equations. We make the following necessary assumption in addition to the setup described so far: the initial size is non-random and that it converges,

$$\mathbf{q}^r(0) \to \mathbf{q}_0 \quad \text{for some } \mathbf{q}_0 \in \mathbb{R}_+^N. \tag{16}$$

Theorem 1 (Theorem 5.1[26]). *Make assumption* (16)*. Let FMS*[2] *be the set of all processes $x(t)$ over $t \in [0,T]$ which satisfy the appropriate fluid model equations, namely*

- *equations* (9)–(14)*, for any scheduling algorithm,*
- *equation* (15) *in addition if the network is running MW-f and Condition 1 holds,*
- $\mathbf{q}(0) = \mathbf{q}_0$ *in addition, if* (16) *holds.*

Let FMS_ε be the ε-fattening

$$FMS_\varepsilon = \left\{ x : \sup_{t \in [0,T]} |x(t) - y(t)| < \varepsilon \text{ for some } y \in FMS \right\}.$$

Then for any $\varepsilon > 0$, $\mathbb{P}(x^r(\cdot) \in FMS_\varepsilon) = 1 - o(R(r))$, where $R(r) \to 0$ as $r \to \infty$.

Fluid stability and throughput optimality. A fluid model is said to be *stable* if there is some draining time $H \in \mathbb{R}_+$ such that every fluid model with bounded initial queue size $|\mathbf{q}(0)| \leq 1$ ends up with $\mathbf{q}(t) = \mathbf{0}$ for all $t \geq H$. It is said to be *weakly stable* if every fluid model with empty initial queues $\mathbf{q}(0) = \mathbf{0}$ remains at $\mathbf{q}(t) = \mathbf{0}$ for all $t \geq 0$. In lecture notes by Dai[8], Section 2.6 describes the relationship between stability of the fluid model and stability of the original (unscaled) stochastic process. Theorem 1 stated above suggests the spirit of the relationship. In our setup, the weakly stable fluid model implies that the system is *rate stable*. That is, let $\mathbf{D}(\tau) = [\mathbf{D}_n(\tau)]$ denote the vector of cumulative number of packets that has departed from from queues till timeslot τ. Then, weakly stable fluid model implies

$$\lim_{\tau \to \infty} \frac{\mathbf{D}(\tau)}{\tau} = \lambda, \text{ with probability 1.}$$

Here, we will seek weak stability only and we will call an algorithm *throughput optimal* if it is weakly stable. However, it should be noted that under our assumptions on arrival process, the strong stability[3] holds (and can be established either using the fluid model or discrete Foster-Lyapunov criteria). We will obtain weak stability for networks by considering the Lyapunov function

$$L(\mathbf{q}) = F(\mathbf{q}) \cdot \mathbf{1} \quad \text{where} \quad F(x) = \int_0^x f(y)\, dy. \tag{17}$$

[2] The abbreviation FMS is used for "fluid model solution".

[3] We call a network strongly stable if it is positive recurrent.

The first claim of Lemma 1, together with the fact that $L(\mathbf{q}) = 0 \iff \mathbf{q} = \mathbf{0}$, implies that the fluid model for MW-f is weakly stable for $\lambda \in \Lambda$. Further, it can be shown that the fluid model is stable for $\lambda \in \Lambda^\circ$; this can be proved by writing an explicit bound for $\dot{L}(\mathbf{q}(t))$ in terms of $\max_\pi \pi \cdot \mathbf{q}(t)$, then using the technique of [28].

Lemma 1. *For $\lambda \in \Lambda$, every fluid model solution satisfies $\frac{dL(\mathbf{q}(t))}{dt} \leq 0$. For $\lambda \in \Lambda^\circ$, every fluid model solution satisfies $\frac{dL(\mathbf{q}(t))}{dt} < 0$. Furthermore,*

$$\frac{dL(\mathbf{q}(t))}{dt} = \lambda \cdot f(\mathbf{q}(t)) - \max_{\pi \in \mathscr{S}} \pi \cdot f(\mathbf{q}(t))$$

and

$$\lambda \cdot f(\mathbf{q}) - \max_{\pi \in \mathscr{S}} \pi \cdot f(\mathbf{q}) \leq 0 \quad \textit{for all } \mathbf{q} \in \mathbb{R}_+^N.$$

Proof.

$$\begin{aligned}
\frac{d}{dt} L(\mathbf{q}(t)) &= \frac{d\mathbf{q}(t)}{dt} \cdot f(\mathbf{q}(t)) \\
&= \Big(\lambda - \sum_{\pi \in \mathscr{S}} \dot{s}_\pi(t)\pi + \dot{\mathbf{z}}(t)\Big) \cdot f(\mathbf{q}(t)) \quad \text{by differentiating (10)} \\
&= \Big(\lambda - \sum_{\pi} \dot{s}_\pi(t)\pi\Big) \cdot f(\mathbf{q}(t)) \quad \text{by (14), using } f(0) = 0 \\
&= \lambda \cdot f(\mathbf{q}(t)) - \max_{\rho \in \mathscr{S}} \rho \cdot f(\mathbf{q}(t)) \quad \text{by (15).}
\end{aligned}$$

When $\lambda \in \Lambda$, we can write $\lambda \leq \sigma$ componentwise for some $\sigma = \sum_\pi \alpha_\pi \pi$ with $\alpha_\pi \geq 0$ and $\sum \alpha_\pi = 1$. This yields $\frac{dL(\mathbf{q}(t))}{dt} \leq 0$. When $\lambda \in \Lambda^\circ$, the same holds except with $\sum \alpha_\pi < 1$, which yields $\frac{dL(\mathbf{q}(t))}{dt} < 0$. □

When we use this result, we almost always implicitly pair it with a standard fact which is worth stating here: if $f : \mathbb{R}_+ \to \mathbb{R}$ is an absolutely continuous function, and $\dot{f}(t) \leq 0$ at almost all t, then $f(t) \leq f(0)$ for all $t \geq 0$.

3.2 Queue-size optimality

The above results suggest that there is a large class of algorithms that are throughput optimal. Here, we will search for an algorithm that produces optimal performance in terms of queue-size as well. Clearly, identifying such an algorithm[4] in an absolute sense is a challenging question and is an open problem worth pursuing.

In this chapter, we will rely on fluid model characterization to seek an asymptotic answer to this question. We will restrict our search to special subclass of MW-f al-

[4] This algorithm should be online, i.e. utilize only the history of the network.

gorithms, the MW-α algorithms for $\alpha \in \mathbb{R}_+$. Recall that the MW-α algorithm uses $f(x) = x^\alpha$ as the weight function. The reason for this restriction is two fold. First, the average queue-size performance of MW-α scheduling algorithms was extensively studied (empirically) in the context of input-queued switch by Keslassy and McKeown [14]. They had observed that as $\alpha \to 0^+$, the average queue-size descreases. This led them to conjecture that MW-0^+ algorithm is an optimal (in the class of MW-α) algorithm with respect to its performance in terms of average queue-size. Second, the MW-α are a large class of algorithms and lend themselves to analytic tractability. Therefore, an optimist would take the conjecture of [14] one step ahead by hoping that MW-0^+ algorithm is optimal among all possible scheduling algorithms.

As we shall see, an optimist's perspective is indeed true; not only for switch, but for the general setup of network considered in this chapter. We again take note of the following limitation of the details provided in the remainder of this section. The justification of the optimality of MW-0^+ is *partial*, because the optimality is established for the fluid scaled queue-size. However, it is non-trivially important for two reasons. First, if MW-0^+ is optimal in terms of, say average (not fluid scaled) queue-size then it ought to be optimal with respect to fluid scaled queue-sizes as well. Second, optimality with respect to fluid scaling does imply[5] an approximate optimality in terms of average queue-size at the original scaling.

The presention in this section follows [26] very closely. In what follows, we will formally introduce critical fluid model and queue-size optimality. Then, we will indulge into a minor digression by introducing various definitions in order to be able to state the main result of this section formally. An impatient and curious reader may skip this digression in the first read and jump directly to the Theorem 2. Finally, we will discuss the implications of this result in terms of the structure of the optimal algorithm.

Critical fluid model. Consider the network operating under MW-α scheduling algorithm with arrival rate $\lambda \in \Lambda$. Let $\mathbf{q}(t)$ denote the fluid scaled queue-size vector. That is, $\mathbf{q}(t)$ satisfies the fluid model equations (9)–(15) (with $f(x) = x^\alpha$) starting with some finite initial queue-size vector $\mathbf{q}_0$ at $t = 0$. We are interested in the net queue-size of the network, that is $\sum_n \mathbf{q}_n(t)$, which we will denote by $\mathbf{1}\cdot\mathbf{q}(t)$. Now, suppose $\lambda \in \Lambda^\circ$. Then Lemma 1 suggests that as $t \to \infty$, the Lypanov function value $L(\mathbf{q}(t)) \to 0$ under MW-α algorithm for any $\alpha > 0$. Therefore, $\mathbf{1}\cdot\mathbf{q}(t) \to 0$ for MW-α algorithm for any $\alpha > 0$. Thus, using fluid model the performance of MW-α algorithms, in terms of the net queue-size at all time t, can not be differentiated if $\lambda \in \Lambda^\circ$. Therefore, in order to obtain any conclusive statement using fluid model, we need to restrict our attention to $\lambda \in \partial\Lambda$. We will call such a system *critically loaded* as λ is on the *boundary* of the capacity region of the system.

[5] This is not immediate and an interested reader will have to *dig through* the proof of Theorem 1. In particular, the approximation error introduced by fluid models in approximating the original system need to be quantified. Of course, it will lead to "probabilistically approximately correct" characterization which will depend on distributional characterization of arrival process in our setup.

The fluid model obtained for such a critically loaded system is called the critical fluid model. Apart from the use of critical fluid model for studying queue-size scaling as in this chapter, the critical fluid model has been utilized as an important technical tool to establish the so called *state-space collapse property* under heavy traffic asymptotic. This turns out to be an important intermediate step to obtain the heavy traffic characterization of networks (an interested reader is strongly recommended to check an excellent sequel of papers by Bramson and Williams [7, 33] to find out more about this).

Queue-size optimality: a formal definition. Now, we formally define the notion of *queue-size optimality* for the purpose of this chapter. Consider a scheduling algorithm $\mathscr{A}$. By Theorem 1, there exists vector of queue-sizes $\mathbf{q}^{\mathscr{A}}(t)$ satisfying (9)–(14) for any $\lambda \in \partial\Lambda$ with some initial queue-size vector $\mathbf{q}^{\mathscr{A}}(0) = \mathbf{q}_0 \in \mathbb{R}_+^N$. We call the algorithm $\mathscr{A}$ as a $(1+\phi)$-approximation algorithm, $\phi \geq 0$, if the following holds: for any other scheduling algorithm $\mathscr{B}$ with the same initial condition and arrival process, its (fluid scaled) queue-size vector $\mathbf{q}^{\mathscr{B}}(t)$ is such that for all $t \geq 0$,

$$\mathbf{1}\cdot\mathbf{q}^{\mathscr{A}}(t) \leq (1+\phi)\mathbf{1}\cdot\mathbf{q}^{\mathscr{B}}(t), \tag{18}$$

for all choices of $\lambda \in \partial\Lambda$ and all initial configuration $\mathbf{q}_0 \in \mathbb{R}_+^N$.

We call an algorithm *queue-size optimal*, if it is 1-approximation algorithm. In what follows, we will state that MW-α algorithm is $N^{\frac{\alpha}{1+\alpha}}$-approximation algorithm. Therefore, it is $(1+\delta)$ approximation when $\alpha = \ln(1+\delta)/\ln N \approx \delta/\ln N$ for $\delta > 0$. Thus, as $\alpha \to 0^+$ the MW-α algorithm becomes 1^+-approximation algorithm and hence essentially optimal.

Some necessary definitions. Here, we state some necessary definitions in order to state the main result about MW-α algorithms's approximate performance formally. Given $\lambda \in \Lambda$, first consider the optimization problem PRIMAL(λ):

minimize	$\sum_{\pi\in\mathscr{S}} \alpha_\pi$
over	$\alpha_\pi \in \mathbb{R}_+$ for all $\pi \in \mathscr{S}$
such that	$\lambda \leq \sum_{\pi\in\mathscr{S}} \alpha_\pi \pi$ componentwise

This problem asks whether it is possible to find a combination of schedules which can serve arrival rates λ; clearly λ is admissible if and only if the solution to the primal is ≤ 1. Now consider its dual problem DUAL(λ):

maximize	$\xi\cdot\lambda$
over	$\xi \in \mathbb{R}_+^N$
such that	$\max_{\pi\in\mathscr{S}} \xi\cdot\pi \leq 1$

The solution is clearly attained when the constraint is tight. Given a queue size vector $\mathbf{Q}$ and any dual-feasible ξ satisfying the constraint with equality, call $\xi\cdot\mathbf{Q}$ the *workload* at the *virtual resource* ξ. The virtual resource specifies a combination of several actual resources (namely the queues themselves). The long-run rate at which work arrives at the virtual resource is $\xi\cdot\lambda$, and the maximum rate at which it can be served is 1.

A concrete example. Consider a system with $N=2$ queues, A and B. Suppose the set $\mathscr{S}$ of possible schedules consists of "serve three packets from queue A" (schedule 1) and "serve one packet each from A and B" (schedule 2). Let λ_A and λ_B be the arrival rates at the two queues, measured in packets per second.
PRIMAL description. Schedule 2 is the only action which serves queue B, so we need to perform schedule 2 at least λ_B times per second. There's no point performing schedule 2 any more than this. This allows for serving λ_B packets per second from queue A, so we additionally need to perform schedule 1 at a rate of $[\lambda_A-\lambda_B]^+/3$ times per second. If we're only allowed to choose one schedule per second, we require $\lambda_B\le 1$ and $\lambda_A/3+2\lambda_B/3\le 1$.
DUAL description Define a virtual resource W as follows. Every time a packet arrives to queue A put $\zeta_A\ge 0$ tokens into W; every time a packet arrives to queue B put $\zeta_B\ge 0$ tokens into W. The most tokens that schedule 1 can remove from W is $3\zeta_A$, and the most tokens that schedule 2 can remove from W is $\zeta_A+\zeta_B$. We may as well normalize (ζ_A,ζ_B) so that the largest of these is 1. The total rate at which tokens arrive is $\lambda_A\zeta_A+\lambda_B\zeta_B$. If we're only allowed to choose one schedule per second, we need this to be ≤ 1.

Set $\zeta_A=1/3$ and $\zeta_B=2/3$, and we recover the PRIMAL constraint that $\lambda_A/3+2\lambda_B/3\le 1$. Set $\zeta_A=0$ and $\zeta_B=1$, and we recover the PRIMAL constraint that $\lambda_B\le 1$.

Critical workloads. Both problems are soluble so, by strong duality, the solutions to both problems are equal. Clearly the solutions to the optimization problems is ≤ 1 for any $\lambda\in\Lambda$. For $\lambda\in\Lambda^\circ$ it is <1, and for $\lambda\in\partial\Lambda$ it is $=1$. When this is so, we call the solutions to the dual problem the *critically-loaded virtual resources*.

Let $\mathscr{S}^*=\mathscr{S}^*(\lambda)$ be the set of all critically loaded virtual resources that are extreme points of the feasible region. Call these the *principal* critically-loaded virtual resources. Note that the feasible region is a polytope, therefore $\mathscr{S}^*$ is finite; and that the feasible region is convex, therefore any critically-loaded virtual resource ζ can be written

$$\zeta=\sum_{\xi\in\mathscr{S}^*}x_\xi\xi\quad\text{with}\quad\sum x_\xi=1\text{ and all }x_\xi\ge 0. \tag{19}$$

The critical workloads have a useful property. Suppose $\lambda\in\Lambda$, and $\lambda\le\sigma$ for some $\sigma\in\Sigma$, as per the definition of Λ. Then

$$\xi_n>0\text{ for some critically-loaded }\xi\implies\lambda_n=\sigma_n \tag{20}$$

In words, if queue n is critical, then it is not possible to reduce it without increasing some other queue. To see this, pick some critically-loaded ξ with $\xi_n>0$. Then

$\xi \cdot \sigma \geq \xi \cdot \lambda$ since $\sigma \geq \lambda$. Also $\xi \cdot \lambda = 1$ since ξ is critical, and $\xi \cdot \sigma \leq 1$ since ξ is feasible for DUAL(σ), and PRIMAL$(\sigma) \leq 1$. Therefore there is equality, therefore $\lambda_n = \sigma_n$.

Example: input-queued switch. Consider a switch with N input ports and N output ports. Let λ_{ij} be the arrival rate at the queue at input port i of packets destined for output port j, $\lambda \in \mathbb{R}_+^{N \times N}$. This means there are N^2 queues in total, not N. This fits with the notation used to describe the input-queued switch in the earlier section, and it is more convenient than the notation for the general network from Section 2. The set $\mathscr{S}$ is the set of all matching in $N \times N$ complete bipartite graph or $N \times N$ permutation matrices. The Birkhoff–von-Neumann decomposition result says that any doubly substochastic matrix is less than or equal to a convex combination of permutation matrices, which gives us

$$\Lambda = \left\{ \lambda \in [0,1]^{N \times N} \ : \ \sum_{j=1}^{N} \lambda_{\hat{i},j} \leq 1 \text{ and } \sum_{i=1}^{N} \lambda_{i,\hat{j}} \leq 1 \quad \text{for all } \hat{i}, \hat{j} \right\}.$$

It is easy to check that

$$\partial\Lambda = \left\{ \lambda \in \Lambda \ : \ \sum_{j=1}^{N} \lambda_{\hat{i},j} = 1 \text{ or } \sum_{i=1}^{N} \lambda_{i,\hat{j}} = 1 \quad \text{for at least one } \hat{i} \text{ or } \hat{j} \right\}$$

We propose the following set $\mathscr{S}^*$ of principal critically-loaded virtual resources. This set is obtained from the row and column indicators $\mathbf{r}_{\hat{i}}$ and $\mathbf{c}_{\hat{j}}$, defined by $(\mathbf{r}_{\hat{i}})_{i,j} = 1_{i=\hat{i}}$ and $(\mathbf{c}_{\hat{j}})_{i,j} = 1_{j=\hat{j}}$. We also need

$$\mathscr{N} = \left\{ \mathbf{n} \in \{0,1\}^{N \times N} : n_{i,j} = 1 \text{ if } \lambda_{i,j} > 0 \right\}$$

Then

$$\mathscr{S}^* = \left\{ \mathbf{r}_{\hat{i}}\mathbf{n} \text{ for } \mathbf{n} \in \mathscr{N} \text{ and } \hat{i} \text{ such that } \sum_{j} \lambda_{\hat{i},j} = 1 \right\} \cup \left\{ \mathbf{c}_{\hat{j}}\mathbf{n} \text{ for } \mathbf{n} \in \mathscr{N} \text{ and } \hat{j} \text{ such that } \sum_{i} \lambda_{i,\hat{j}} = 1 \right\}$$

The virtual resource $\mathbf{r}_1$, for example, corresponds to the constraint that at most one packet can be served from input port 1 in any timeslot, therefore the total arrival rate at input port 1 must be ≤ 1. If say $\lambda_{1,3} = 0$ then the total arrival rate to the remaining $N-1$ queues at input port 1 must also be ≤ 1, and this corresponds to the virtual resource $\mathbf{r}_1\mathbf{n}$ for $n_{i,j} = 1_{i>1 \text{ or } j \neq 3}$.

It is easy to see that every $\xi \in \mathscr{S}^*$ is a critically-loaded virtual resource, and it is not hard to check that they are all extreme as well. To show (19) requires some more work.

First, we remark upon a dual to the Birkhoff–von-Neumann decomposition. Let

$$\mathscr{D} = \{\mathbf{r}_{\hat{i}} \text{ for all } \hat{i}\} \cup \{\mathbf{c}_{\hat{j}} \text{ for all } \hat{j}\}.$$

Then, given any vector $\zeta \in \mathbb{R}_+^{N\times N}$ for which $\max_{\pi\in\mathscr{S}} \zeta\cdot\pi \le 1$, we can find some ζ' which is a convex combination of elements of $\mathscr{D}$ such that $\zeta \le \zeta'$ componentwise. This is because DUAL$(\zeta) \le 1$ when taken with respect to the schedule set $\mathscr{D}$, by the condition on ζ; and ζ' is then obtained from PRIMAL(ζ).

Now suppose that ζ is any critically-loaded virtual resource for DUAL(λ). We need to show that (19) holds. First, use the dual decomposition above to write

$$\zeta = \sum_{\hat{i}} x_{\hat{i}}\mathbf{r}_{\hat{i}} + \sum_{\hat{j}} y_{\hat{j}}\mathbf{c}_{\hat{j}} - \mathbf{z}.$$

Note that $\mathbf{r}_{\hat{i}}\cdot\lambda \le 1$ with equality only if $\mathbf{r}_{\hat{i}} \in \mathscr{S}^*$, and similarly for $\mathbf{c}_{\hat{j}}$. Since ζ is assumed to be critically loaded, $\zeta\cdot\lambda = 1$; it must therefore be that the coefficients $x_{\hat{i}}$ and $y_{\hat{j}}$ are 0 unless the corresponding virtual resource is in $\mathscr{S}^*$, and also that $z_{i,j} > 0$ only when $\lambda_{i,j} = 0$.

To recap, we have found

$$\zeta = \sum_{\xi\in\mathscr{S}^*} a_\xi \xi - \mathbf{z}$$

where $\sum a_\xi = 1$ and $a_\xi \ge 0$, and $z_{i,j} > 0$ only when $\lambda_{i,j} = 0$. It remains to dispose of $\mathbf{z}$. Suppose $z_{k,l} > 0$ for some k, l, and define $\mathbf{n}^{k,l}$ by $n^{k,l}_{i,j} = 1_{i\neq k \text{ or } j\neq l}$; note that $\mathbf{n}^{k,l} \in \mathscr{N}$ by the condition on $\mathbf{z}$. Also note that $\zeta \in \mathbb{R}_+^{N\times N}$, and so $\sum a_\xi \xi_{k,l} \ge z_{k,l}$. Now we can rewrite

$$\zeta = \frac{z_{k,l}}{\sum a_\xi \xi_{k,l}} \sum a_\xi \xi + \Big(1 - \frac{z_{k,l}}{\sum a_\xi \xi_{k,l}}\Big) \sum a_\xi \xi \mathbf{n}^{k,l} - \mathbf{z}\mathbf{n}^{k,l}.$$

Continuing in this way we can remove all non-zero elements of $\mathbf{z}$, until we are left with an expression of the form (19).

Main result: queue-size optimality of MW-0^+. The following result establishes the claim about queue-size optimality of MW-0^+ (see Theorem 10.2, [26] for detailed proof.)

Theorem 2. *Let $\lambda \in \partial\Lambda$ be such that there is a critically-loaded virtual resource which assigns equal wait to each queue (i.e. $\mathbf{1}/\max_\pi \mathbf{1}\cdot\pi$ is a critical virtual resource). Then, MW-α algorithm is $N^{\alpha/(1+\alpha)}$-approximation algorithm.*

Theorem 2 implies that for $\alpha = \ln(1+\delta)/\ln N$, the MW-$\alpha$ algorithm is $(1+\delta)$-approximation for any $\delta > 0$. Thus, as $\alpha \to 0^+$ the MW-0^+ becomes 1^+-approximation and hence optimal.

Remark. In the example of input-queued switch, an example of the requirement that there be a critically-loaded virtual resource which assigns equal weight to all queues is satisfied by the requirement that either there is some set of critically loaded input ports (i.e. $\sum_k \lambda_{ik} = 1$ for some collection of i) and $\lambda_{i,j} = 0$ for all input ports i which are not critical; or that there is some set of critically loaded output ports and $\lambda_{i,j} = 0$ for all output ports j which are not critical.

Discussion: optimal algorithm. The above is an evidence based on critical fluid model of the optimality of limiting algorithm MW-0^+. There a second piece of (intuitive) evidence based on the *structure* of effective state space of the algorithm in the case of switch. It essentially suggests that as $\alpha \to 0^+$, the effective state space becomes largest possible and hence the algorithm does *not idle* unless essentially required – thus, being least *wasteful* and hence optimal. Due to space constraint, we do not discuss this in further detail. An interested reader is encourage to read [26] for furter details.

Given these two pieces of evidence, it is tempting to speculate about a formal limit of MW-α as $\alpha \to 0$. Since MW-α chooses a schedule π to maximize $\pi \cdot \mathbf{q}^\alpha$, and since

$$x^\alpha \approx \begin{cases} 1 + \alpha \log x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases}$$

we make the following conjecture:

Conjecture 1. Consider the MW-0^+ scheduling algorithm, which at each timeslot looks at all maximum-size schedules (i.e. those $\pi \in \mathscr{S}$ for which $\sum_n \pi_n 1_{q_n>0}$ is maximal), and among these picks one which has maximal log-weight (i.e. for which $\sum_{n:q_n>0} \pi_n \log q_n$ is maximal), breaking ties arbitrarily. We conjecture that this algorithm is stable for $\lambda \in \Lambda^\circ$, and that it minimizes the total amount of idling in both the fluid limit and the heavy traffic limit for $\lambda \in \partial\Lambda$.

Key message on optimal algorithm. In [16], McKeown et. al. showed that maximum-size matching (without use of weights to break ties) is not stable for certain $\lambda \in \Lambda^\circ$, for an input-queued switch. However, the above conjecture (and MW-0^+ algorithm) suggests that the maximum-size schedule with (logarithmic) weights used to break ties is optimal. This suggests that the role of using *weight information* is in getting the *throughput maximized* while the role of *maximum-size* is in *minimizing delay*. This property is implicitly achieved by the MW-α algorithm for $\alpha \to 0^+$.

4 Message-passing: throughput optimality

The previous section argued that a maximum weight scheduling rule, with weights as an appropriate function of queue-size, leads to optimal performance both in terms of throughput and delay. Therefore, such a scheduling algorithm is required to solve a combinatorial optimization problem of finding a maximum weighted schedule, out of all possible choices, every timeslot.

Though the problem of finding maximum weight scheduling is a solvable (because number of scheduling choices are finite in our setup), if the scheduling con-

straints are complex (e.g. independent set in wireless network example), then designing efficient scheduling can become challenging (or may be impossible) in general. In most of scheduling applications, such as the input-queued switch and the wireless network considered here, the algorithms for finding schedule are highly constrained due to various engineering limitations: (a) they need to perform few logical operations either because of time-limitation or limited computational resources; (b) they need to operate in totally distributed manner while exchanging as little information as possible due to physical or architectural reasons; and (c) they need to maintain only little amount of data-structure. Such considerations lead to the fundamental question: is it possible to design *implementable* (as in satisfying the above stated qualitative requirements) scheduling algorithms that have good performance, both in terms of throughput and queue-size (delay), for any instance of the setup described here?

In this section, we will present an extremely simple, randomized distributed and hence implementable algorithm that is throughput optimal for any instance of the setup describe here. However, this algorithm can have very poor queue-size performance depending upon the complexity of the underlying problem structure. This algorithm (in centralized setup) was first proposed by Tassiulas [30] in the context of switch. The distributed implementation of this algorithm was discussed in [18, 13]. First, we present the generic algorithm and its throughput property. Then, we will explain it in the context of input-queued switch (matching) and wireless network (independent set). Finally, we will discuss its performance in terms of queue-size.

4.1 Throughput optimality through randomization and message-passing

The algorithm essentially uses the following key insights: (1) if a schedule $\pi \in \mathscr{S}$ has *high* weight (say, weight is equal to queue-size) at certain time-instance, then it is likely to have *high* weight at the next time-step as long as the weight is a linear or sub-linear function of queue-size; (2) in order to achieve throughput optimality, it is sufficient to have the weight of the schedule *close to* optimal, not necessarily optimal. Now, we describe the algorithm for any scheduling instance of our setup. But, before that we describe two sub-routines that the algorithm will utilize.

Sub-routine **RND**. It produces a random schedule $\sigma \in \mathscr{S}$ such that any schedule in $\mathscr{S}$ has strictly positive probability of being produced. That is, there exists $\omega > 0$ such that for any $\sigma \in \mathscr{S}$

$$\Pr(\textbf{RND} \text{ outputs } \sigma) > \omega. \tag{21}$$

Sub-routine **CNT**$(\sigma, \mathbf{w}, \varepsilon, \delta)$. Given parameters $\varepsilon, \delta > 0$ and schedule $\sigma \in \mathscr{S}$, with node-weights $\mathbf{w} = (w_i)$ (here, queue-size or function of queue-size), the algorithm returns a random number W such that

$$\Pr((1-\varepsilon)w(\sigma) \le W \le (1+\varepsilon)w(\sigma)) \ge 1-\delta, \tag{22}$$

where $w(\sigma) = \sum_i \sigma_i w_i$.

ALGO I(ε,δ). Here, we describe the generic form of the algorithm. The distributed implementation of this algorithm will follow by explaining the distributed implementation of the subroutines **RND** and **CNT**, which is done later.

0. Let $\sigma(\tau)$ be the schedule used at time $\tau \ge 0$ and $\mathbf{Q}(\tau)$ denote the queue-size vector at time τ.
1. Initially, at $\tau = 0$ we have $\mathbf{Q}(0) = \mathbf{0}$ and choose $\sigma(0)$ at random using **RND**.
2. The schedule $\sigma(\tau+1)$ at time $\tau+1$ is computed from $\sigma(\tau)$ as follows.
   (a) Produce a random schedule $R(\tau+1)$ using **RND**.
   (b) Compute weights,
   $$W(\sigma(\tau)) = \mathbf{CNT}(\sigma(\tau), \mathbf{Q}(\tau+1), \varepsilon/8, \delta), \text{ and}$$
   $$W(R(\tau+1)) = \mathbf{CNT}(R(\tau+1), \mathbf{Q}(\tau+1), \varepsilon/8, \delta).$$
   (c) If $W(R(\tau+1)) \ge \frac{(1+\varepsilon/8)}{(1-\varepsilon/8)} W(\sigma(\tau))$, then $\sigma(\tau+1) = R(\tau+1)$, else retain $\sigma(\tau+1) = \sigma(\tau)$.

Remark. The above algorithm is an *approximation* of MW-1 algorithm. The result stated next about its throughput optimality should not be affected if we were approximating MW-α algorithm for $\alpha \in (0,1)$. That is, if instead of weight being $\mathbf{Q}(\tau)$, it were $\mathbf{Q}^\alpha(\tau)$, then the above algorithm will still have good throughput property. This is primarily because $f(x) = x^\alpha$ is a Lipschitz continuous function (with Lipschitz constant ≤ 1) for all $\alpha \in (0,1]$.

Performance of ALGO I(ε,δ). Here, we state the result that establishes (essentially) throughput optimality of the **ALGO I**.

Theorem 3. *Consider any strictly admissible $\lambda \in \Lambda^\circ$. Then, there is a $\varepsilon > 0$ such that $(1-2\varepsilon)^{-1}\lambda \in \Lambda^\circ$. Then, under the algorithm **ALGO I($\varepsilon, \omega 3^{-N}$)**,*

$$\limsup_{\tau\to\infty} \mathbb{E}[\|\mathbf{Q}(\tau)\|_1] < \infty.$$

The proof of this result follows from Foster-Lyapunov criteria [17] by considering the quadratic Lyapunov function $L(\tau) = \sum_i Q_i^2(\tau)$ and establishing *negative drift* over large enough finite horizon. We skip details here. An interested reader can reconstruct the proof from [18] or [13].

Cost of ALGO I($\varepsilon, \omega 3^{-N}$). As Theorem 3 suggests, we need to design distributed algorithms **RND**and **CNT**$(\varepsilon, \omega 3^{-N})$ that are efficient (i.e. do computations in say *polynomial* in N distributed operations). We describe such distributed algorithms next. As we shall find, the **RND**, which is described for matching and indepen-

dent set, takes $O(N)$ total computation (or $O(1)$ *rounds*[6]); the **CNT**$(\varepsilon, \omega 3^{-N})$ takes $O(\varepsilon^{-2}N^2 \log 3^N/\omega)$ total computation (or $O(\varepsilon^{-2}N \log 3^N/\omega)$ *rounds*). Thus, net computation cost of the **ALGO I**$(\varepsilon, \omega 3^{-N})$ will boil down to $O(\varepsilon^{-2}N \log 3^N/\omega)$ rounds or $O(\varepsilon^{-2}N^2 \log 3^N/\omega)$ total distributed operations. It should be noted that the number of *message* exchanged scales in the same manner as the number of distributed operations. As we shall see, $\omega = 1/N! \approx 2^{-N \log N}$ for matching and $\omega = 2^{-N}$ – thus, the cost (in terms of rounds) in case of matching is $O(\varepsilon^{-2}N^2 \log N)$ and $O(\varepsilon^{-2}N^2)$ in case of independent set.

Remark. It should be noted that we can slow down our schedule computation algorithm by factor $O(\varepsilon^{-2}N \log 3^N/\omega)$ – thus, spending $O(1)$ computation per timeslot – and retain the throughput optimality as is. This fact follows directly from the Lyapunov-Foster's criteria. However, it increases the average queue-size and thus degrades performance. An interested reader will find a detailed study of this aspect of scheduling algorithms in the context of input-queued switch in [23].

Description of RND. Here, we describe the distributed algorithm **RND** for two examples: matching and independent set. The algorithm for any instance of our setup can be obtained as long as the constraints corresponding to the feasibility of a schedule is checkable *locally*; which happens to be the case for matching and independent set.

First, consider **RND** for finding a random matching or a random schedule in the case of input-queued switch. Each input node *i*, uniformly at random selects an output node, say $r(i)$, and sends a request to $r(i)$. An output node, say j, upon receiving multiple requests, selects one of the inputs at random and sends it notification of acceptance. Input node *i*, upon receiving accept matches the output and upon not receiving accept (i.e. receiving reject) does not connect to any output.

It can be easily checked that all complete matchings are likely to be chosen with probability at least $1/N!$, for switch of size N, under this **RND**. Further, it involves only two rounds of distributed computation. Thus, it satisfies our required condition (21).

Next, we describe a similar description for independent set. Here, in the network graph $G = (V, E)$, each node (vertex) chooses to become *active* with probability $1/2$. An *active* node, becomes *inactive* if any of its neighbor is *active*. All the remaining *active* nodes declare them to be part of the independent set while others (*inactive*) keep out.

Again, it is easy to check that the nodes that decide to be part of independent set, indeed form an independent set. Further, any independent set has probability at least $1/2^N$ of being chosen for network of size N. As description suggests, it is a simple two-round algorithm.

Description of CNT($\varepsilon, \omega 3^{-N}$). The purpose of algorithm is to compute summation of node weights (approximately) for a given schedule – equivalently, given N numbers in the network graph G, compute their summation approximately. The standard

[6] By a round, we mean an iteration of distributed computation where each node gets to perform $O(1)$ exchanges.

averaging algorithm (cf. [32, 6]) will not work here, because we need an algorithm that will produce exactly the *same* estimation at all the nodes so that local decisions (i.e. whether to choose new schedule $R(\tau+1)$ or to choose an old schedule $\sigma(\tau)$) are globally consistent. And, averaging algorithm does not posses this property. Here, we will describe an approximate summation procedure based [19]. The algorithm is based on the following probabilistic facts:

F1. Let $X_1,\ldots,X_k$ be independent random variables with exponential distribution and parameters $r_1,\ldots,r_k$. Then, $X_* = \min_{1\le i\le k} X_i$ has exponential distribution with parameter $\sum_{i=1}^k r_i$.
F2. Let $Y_1,\ldots,Y_m$ be independent exponential random variables with parameter r. Let $S_m = \frac{1}{m}\sum_{i=1}^m Y_i$. Then, for $\gamma \in (0,1/2)$

$$\Pr\left(S_m \notin (1-\gamma)r^{-1},(1+\gamma)r^{-1}\right) \le 2\exp\left(-\gamma^2 m/2\right).$$

F1 is well-known about exponential distribution; **F2** follows from Cramer's Theorem [10] about large deviation estimation for exponential distribution.

Now, the algorithm. Given node weights $W=[W_v]$, **F1** and **F2** can be used to compute $\bar{W} = \sum_v W_v$ approximately as follows: each node $v \in V$ draws an independent exponential random variable with parameter W_v (nodes with $W_v = 0$ do not participate in generating numbers); then all nodes together compute minimum, say X_* of these random numbers in distributed fashion by iteratively asking their neighbors for their estimates of minimum. Nodes should terminate this process after $\Theta(n)$ transmissions. Repeat this for m times to obtain minimums $X_*(i), 1 \le i \le m$. Now set $S_m = \frac{1}{m}\sum_{i=1}^m X_*(i)$ and declare $Z_m = 1/S_m$ as an estimate summation of $\bar{W}$.

Now, given small enough ε it follows from **F1**, **F2** that by selecting $m = O(\varepsilon^{-2}\log 3^N/\omega)$, we obtain estimate of summation, say $\hat{W}$ such that

$$\Pr\left(\hat{W} \notin ((1-\varepsilon)\bar{W},(1+\varepsilon)\bar{W})\right) \le \omega 3^{-N}. \tag{23}$$

Computation of a single minimum over the network can be done in a distributed manner in many ways. We skip the details here in interest of space. However, we refer an interested reader to see [19] for interesting account on such algorithms. The minimum computation takes total $O(N^2)$or $O(N)$ per node message exchanges. This completes the description of desired **CNT**$(\varepsilon,\omega 3^{-N})$ algorithm.

4.2 Performance in terms of queue-size

The **ALGO I** is a simple, randomized message-passing algorithm that is throughput optimal for almost all reasonable instances of our setup. Now, we consider its performance in terms of queue-size. We will consider the cases of matching (switch scheduling) and independent set (wireless scheduling) to establish existence of the following dichotomy: for some scheduling problem, it is possible to have simple algorithms that are throughput optimal and have small queue-size; for other schedul-

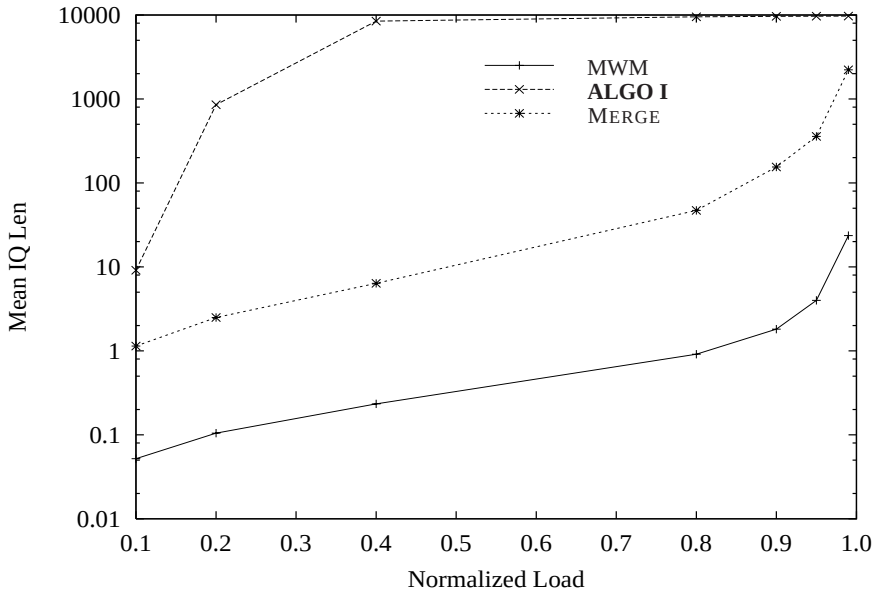

Fig. 3 An illustration of impact of MERGE on Performance.

ing problems we can only hope for throughput optimal simple algorithms with very large queue-size.

ALGO I for switches: a simple modification. The basic version of the **ALGO I** described above is likely to induce very large queue-size. However, a simple modification of the **ALGO I** can lead to smaller queue-size[7] as described here. To exemplify this, we present a sample simulation in Figure 3 which plots average queue-size (on Y-axis) with respect to the varying load (on X-axis) for three algorithms: MW scheduling (MWM), the **ALGO I**, and MERGE (which is the modification of **ALGO I** for matching). The figure shows that the queue-sizes are very large under **ALGO I** , but MERGE and MWM have very small and comparable queue-sizes. The modification presented here, the MERGE algorithm, is based on the results described in [12] and a later adaption of it for distributed algorithm design in [18].

The main insight is as follows. In **ALGO I**, every time either we choose schedule $R(\tau+1)$ or $\sigma(\tau)$ entirely. However, some parts of $R(\tau+1)$ are likely to be higher weights while some other parts of $\sigma(\tau)$ are likely to be of higher weights. Therefore, a better approach towards designing such algorithm would be to choose *a mixture* of the best parts of these two schedules. In general, not all scheduling constraint

[7] We make note of the following: while the modification presented here seem to reduce queue-size drastically (see [12] for detailed simulations) and there are arguments based on toy-model (again, see [12]) to justify this, the problem of establishing average queue-size being *polynomial* in size of switch N, under algorithm using MERGE remains an important open problem.

structures allow for this. However, matching constraint allows for this possibility. Below, we describe this formally as the Merge procedure.

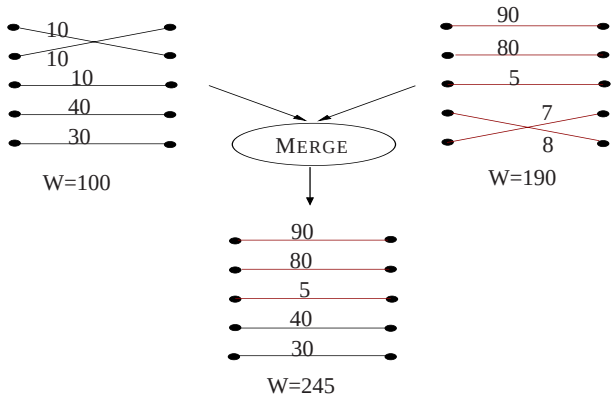

Fig. 4 An example of MERGE procedure.

Consider a switch bipartite graph with **Q** matrix as its edge weights. Given two matchings π^1 and π^2, define

$$\mathscr{S}(\pi^1,\pi^2) = \{\pi \in \mathscr{S} : \pi_{ij} = 1 \text{ only if } \pi^1_{ij} = 1 \text{ or } \pi^2_{ij} = 1\}.$$

The MERGE procedure, when applied to π^1 and π^2 with weights given by **Q**, returns a matching $\tilde{\pi}$ such that

$$\tilde{\pi} = \arg \max_{\pi \in \mathscr{S}(\pi(1),\pi(2))} \{\sum_{ij} \pi_{ij} Q_{ij}\}. \tag{24}$$

The MERGE finds such matching using only $2n$ addition and subtraction. It is described as follows: Color the edges of π^1 as red and the edges of π^2 as green. Start at output node j_1 and follow the red edge to an input node, say i_1. From input node i_1 follow the (only) green edge to its output node, say j_2. If $j_2 = j_1$, stop. Else continue to trace a path of alternating red and green edges until j_1 is visited again. This gives a "cycle" in the subgraph of red and green edges.

Suppose the above cycle does not cover all the red and green edges. Then there exists an output j outside this cycle. Starting from j repeat the above procedure to find another cycle. In this fashion find all cycles of red and green edges. Suppose there are ℓ cycles, $C_1,...,C_\ell$ at the end. Then each cycle, C_i, contains two matchings: G_i which has only green edges, and R_i which has only red edges. For each cycle C_i, the MERGE chooses R_i if the sum of the queue-size corresponding to these edges is higher than that of the G_i. Else, MERGE chooses G_i. It is easy to show that the final matching as chosen above is precisely the one claimed in (24). Figure 4 illustrates the MERGE procedure.

Finally, the MERGE is used in place of choosing $R(\tau+1)$ or $\sigma(\tau)$ entirely. Note that, in order to construct the schedule for MERGE, essentially we need to compute the weights of schedules only restricted to the "cycles" as described above. However, this can be done in a totally distributed manner using the same **CNT** procedure since the membership to a cycle (or paths) is by definition defined locally. Note that this modification *does not* increase the (bound on the) cost of the algorithm. We refer interested readers to [18] for details.

ALGO I for independent set: impossibility of low queue-size. The above simple modification for matching reduces queue-sizes drastically and makes the algorithms comparable to MW scheduling. However, it utilizes the structure of matching crucially. Therefore, question remains whether it is possible to modify **ALGO I** to obtain small queue-size for any scheduling problem. Here, we will state an impossibility result in the context of independent set based scheduling which implies that, (a) **ALGO I** has exponentially large, in problem size N, average queue-size and (b) it is not possible to have simple modification of **ALGO I** to obtain smaller queue-sizes. This is based on a recent work [24].

To this end, we consider a wireless network operating under independent set constrained model. We assume that the network graph G can be arbitrary. Let Λ be its admissible arrival rate region and $c\Lambda = \{c\lambda : \lambda \in \Lambda\}$ for $c > 0$. Here, $c\Lambda$ means fraction c of the capacity region: e.g. for $c = 0.1$, it will be 10% of the capacity region. The following impossibility result implies that for any $\varepsilon > 0$, there is no simple algorithm that can achieve small average queue-size for all network instances. Thus, the problem of scheduling for low queue-size is inherently hard !

Theorem 4. *Consider any $\varepsilon > 0$. Then, there is no (centralized or distributed, deterministic or randomized) algorithm that runs in polynomial (in N) time and induces polynomial (in N) average queue-size for all $\lambda \in \varepsilon\Lambda$ unless* certain computational hypothesis[8] *is false.*

Key message on simple, randomized message-passing algorithm. The randomized algorithm **ALGO I** described here is a simple, message-passing mechanism that is essentially throughput optimal for any scheduling instance that allows for checking feasibility of a schedule through local constraints (e.g. matching, independent set). However, the queue-sizes induced are very large. The simpler problem structures, like matching allow for minor (problem dependent) modification of the **ALGO I**, to obtain lower queue-sizes while retaining the high throughput. However, for hard problem structures, like independent set it is impossible to obtain simultaneously high-throughput and low queue-sizes under arbitrary setup. Thus, *obtaining high-throughput is relatively simple, and meanwhile maintaining low queue-size is quite hard*.

[8] The precise computational hypothesis is NP $\not\subset$ BPP.

5 Message-passing: low queue-size or delay

In essence, we have learnt so far that in order to retain small queue-size a known effective way is to design excellent approximation algorithm of maximum weight scheduling – in case of matching, we could do it since it is an easy problem, but in case of independent set we could not since it is a hard problem. The modification of randomized algorithm **ALGO I** to obtain small queue-size for matching is very problem specific. Ideally, we would like to have a generic method. Specifically, in this section we would like to design general message-passing algorithmic method that has the following properties: (a) for easy problem, allows for fine-control to trade-off performance with implementation cost; and (b) for hard problem, works well when problem posses special structure (like solvable through linear program) and gives a reasonable heuristic otherwise.

We will present two, somewhat surprisingly very related, approaches for algorithm design: (a) the classical optimization based method of co-ordinate descent algorithm and (b) the recently emerging heuristic from Statistical Physics and Artificial Intelligence, called belief propagation (also known as max-product for optimization problem). The presentation here is based on [3], [4] and [21]. We will explain these two methods in the context of input-queued switch (matching) and wireless network (independent set).

5.1 Input-queued switch: message-passing algorithm

Here, we describe two algorithms for input-queued switch. The first algorithm is a direct adaptation of the Auction algorithm by Bertsekas [5]. The second algorithm is based on belief propagation (max-product).

Auction algorithm. For ease of explanation, we introduce some notation. Consider an N port input-queued switch with N input ports and N output ports. Denote the N input ports by $\alpha_1,\ldots,\alpha_N$ and the N output ports by $\beta_1,\ldots,\beta_N$. As described earlier in Section 2, there are N^2 queues, one per distinct input-output pair. Let

At time τ the weight of an edge (α_i,β_j) will be $Q_{ij}(\tau-1)$ and the weight of the matching π is $\sum_{i=1}^{n} Q_{i\pi(i)}(\tau-1)$. Recall that the Maximum Weight Matching $\pi^*(\tau)$ at time τ is such that

$$\pi^*(\tau) \in \arg\max_{\pi\in\mathcal{J}} \sum_{i=1}^{n} Q_{i\pi(i)}(\tau-1).$$

Now we describe the auction algorithm with parameter $\varepsilon > 0$. In the description of the algorithm, we drop reference to time τ for the queue-size. Readers familiar with the iSLIP [15] algorithm may notice a striking syntactic similarity between the iSLIP and the auction algorithms: both algorithms iterate between inputs proposing

and outputs accepting/refusing. This similarity suggests that the auction algorithm is likely to very close to be implementable.

- *Phase 0: Initialization.* Given queue-size matrix $\mathbf{Q}$, let $Q^* = \max_{ij} Q_{ij}$ which is determined as follows:
  - Each output β_j computes $Q^*_{\cdot j} = \max_{k=1}^{n} Q_{kj}$.
  - Each input α_i obtains $Q^*_{\cdot j}$ from all outputs β_j and computes $Q^* = \max_j Q^*_{\cdot j}$.
  - Each output β_j contacts input α_j to obtain Q^*.
  - Set $\delta = \varepsilon Q^*/n$.
  - Initially, the set of matched inputs-outputs $S = \emptyset$; the set of unassigned inputs $I = \{\alpha_1, \ldots, \alpha_n\}$, and parameters $p_j = 0$ for $1 \leq j \leq n$.
  - Algorithm finds matching of interest in two phases, described next.
- *Phase 1: Bidding* For all $\alpha_i \in I$,

  (1)Find the 'weight' maximizing output β_j. Let,

$$j_i = \text{argmax}_j \{Q_{ij} - p_j\},\ v_i = \max_j \{Q_{ij} - p_j\}, \tag{25}$$

$$\text{and } u_i = \max_{j \neq j_i} \{Q_{ij} - p_j\}. \tag{26}$$

  (2)Compute the 'proposal' of input α_i, denoted by $b_{\alpha_i \to \beta_j}$ as follows:

$$b_{\alpha_i \to \beta_{j_i}} = Q_{ij_i} - u_i + \delta.$$

- *Phase 2: Assignment.* For each output β_j,

  (3)Let $P(j)$ be the set of inputs from which β_j received a 'proposal'. If $P(j) \neq \emptyset$, increase p_j to the highest bid, i.e.

$$p_j = \max_{\alpha_i \in P(j)} b_{\alpha_i \to \beta_j}.$$

  (4)Remove the maximum proposing input α_{i_j} from I and add (α_{i_j}, β_j) to S. If $(\alpha_k, \beta_j) \in S,\ k \neq i_j$, then put α_k back in I.

Performance of Auction algorithm. The auction algorithm described above is a slight variant of Bertsekas' auction algorithm. Given a fixed weighted bipartite graph, the behavior of the auction algorithm is well understood. However, the algorithm converges only if all the weights are finite. In our setup, weights are given by $\mathbf{Q}(\cdot)$. Hence, it is not clear if the above described algorithm will maintain finite queue-sizes $Q^*(\cdot)$ with probability 1. Specifically, the size of $Q^*(\cdot)$ directly affects the number of iterations required by the algorithm to converge. We state the following result (Theorem 1, [4]).

Theorem 5. *Given $\varepsilon > 0$, let $\lambda = \sum_k \alpha_k \pi_k$ be such that $\sum_k \alpha_k \leq 1 - 2\varepsilon$. Then, for a switch operating under the Auction algorithm with parameter ε,*

$$\limsup_{\tau\to\infty} \mathbb{E}\left[\sum_{ij} Q_{ij}(\tau)\right] = O(n^2/\varepsilon).$$

Further, the algorithm takes $O(n^2/\varepsilon)$ iterations to compute the schedule.

Now, we consider a natural variant of the Auction algorithm to utilize the slowly varying nature of the switch. Specifically, at any time slot $\tau+1$ the parameter p_j for $1 \le j \le n$ instead of being initiated with zero starts with its final value from the time slot τ. The intuition behind this modification is the following. At the end of the time slot τ the parameters p_j are optimal for the queue sizes $Q_{ij}(\tau-1)$. Since, queue-size only changes by ± 1 in a time slot, one expects the parameters p_j to be near optimal at time $\tau+1$ as well. Therefore, we expect algorithm to converge quickly starting from thus chosen new initial condition – this is confirmed by simulations presented next.

Representative simulation results. We describe simulation results for an 8×8 input queued switch with a non-uniform admissible arrival matrix. The traffic load takes one of the values from the set $\{.65, 8, .9, .95, .98\}$. All simulations are done for one million time slots. Here, "auction(c)" denotes the auction algorithm with $\delta = c$ where c is a constant. For the ε-Auction algorithm we use $\varepsilon = 1$ and hence denote it by 1-Auction. We compare the performance of auction algorithm with MWM and iSLIP algorithm – iSLIP is the popular heuristic used in practice. When the number of iterations of the iSLIP algorithm is not mentioned it is understood to have run all the way to the end, i.e. it runs $n = 8$ iterations.

Figure 5 shows that the 1-Auction algorithm performs much better than the iSLIP algorithm and is as good as MWM. The next plot, Figure 6, shows that 1-Auction with memory has better performance than 1-Auction.

Figures 7 and 8 show the trade-off achieved by tuning the parameter δ: the higher the value of δ, the poorer the performance and the fewer iterations required to find solution. Here the value $\delta = B/N$ takes one of the values $1, 10, 50$. As mentioned before larger values of δ yield less number of iterations but at the expense of greater queue sizes. Figure 9 shows a comparison between 1-Auction and iSLIP when both run only three iterations in each time slot. In practice sometimes only a few iterations of the iSLIP algorithm are used instead of the full iSLIP. This figure shows that the 1-Auction algorithm can also be used for a fewer number of iterations and it still outperforms iSLIP.

Belief propagation(BP) for matching. In a switch, the partition in the bipartite graph is well known. However, in general graph, even if it is bipartite such partition may not be known – for example, wireless ad-hoc network operating under *primary interference constraints*. In such cases, Auction algorithm, which requires prior partitioning and is not symmetric in its response to partitioning, is not very attractive. Instead, we would like to have a scalable approach that *does not* require prior knowledge of the bipartition and operates symmetrically. That is, we need an algorithm, that treats nodes of the two partitions in the same manner. Next, we describe such an algorithm based on the (Max-Product) Belief propagation algorithm.

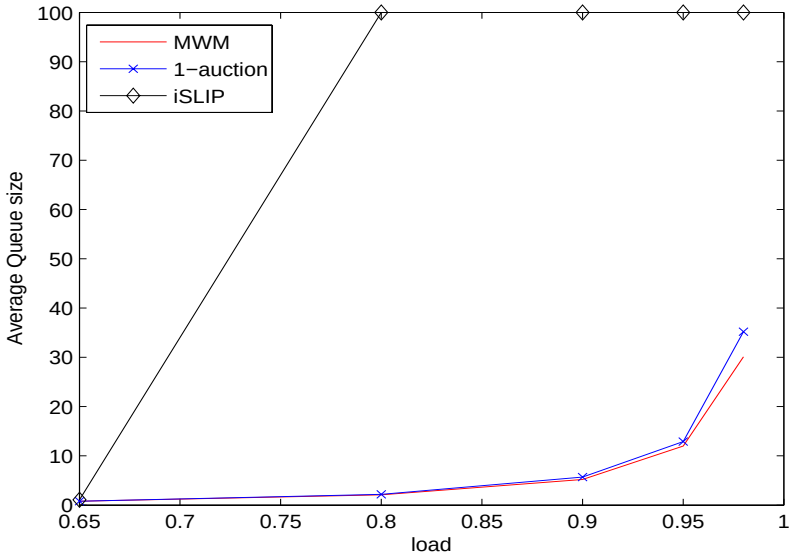

Fig. 5 Average queue sizes for MWM, 1-Auction and iSLIP.

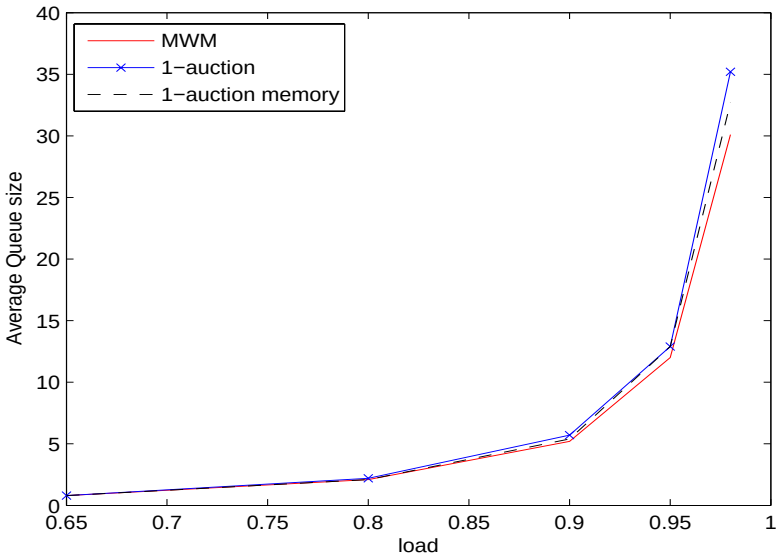

Fig. 6 Average queue sizes MWM, 1-Auction and 1-Auction with memory.

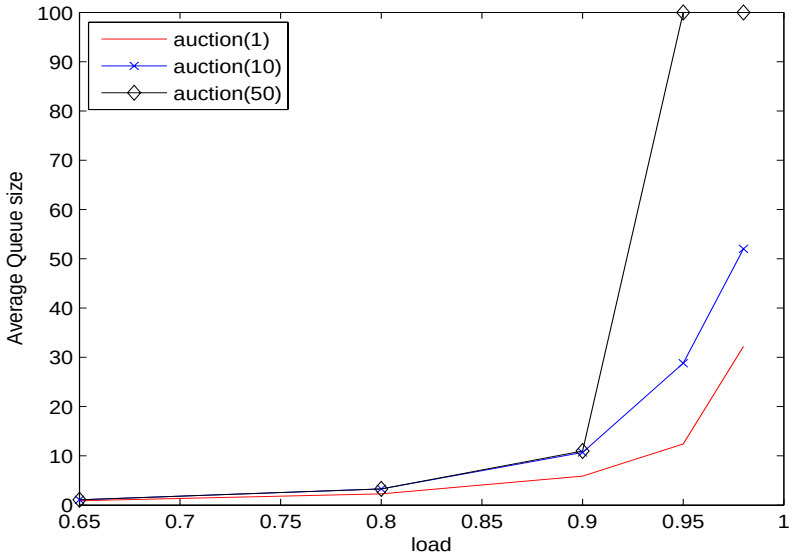

Fig. 7 Average queue sizes for auction(1), auction(10), auction (50)

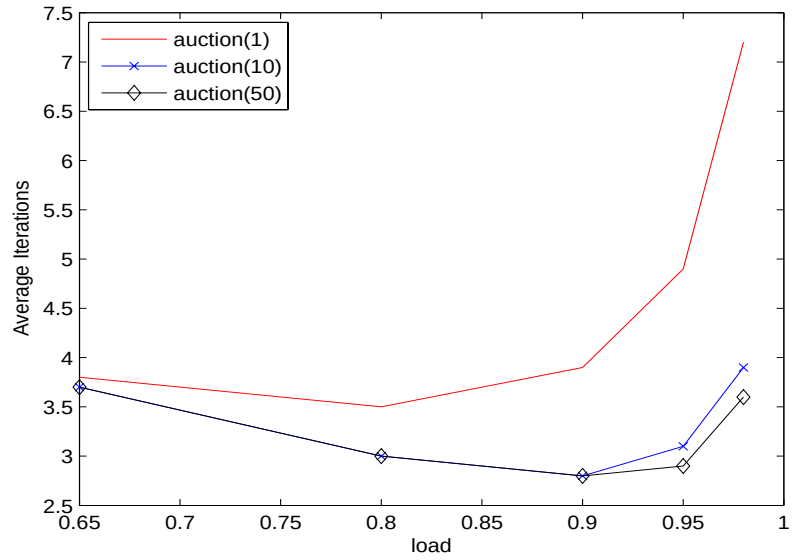

Fig. 8 Average iterations to converge for auction(1), auction(10), auction(50)

The following algorithm is an adaption of the (Max-Product) Belief Propagation algorithm described in [3] that operates very similarly to the auction algorithm.

- Let $Q^* = \max_{ij} Q_{ij}$, which can be quickly computed in a distributed manner. Set $\delta = \varepsilon Q^*/n$.
- Given queue-size matrix $\mathbf{Q}$, define a symmetric weight matrix $\mathbf{W} = [W_{ij}]$ as follows: for all $(i,j) \notin E$, set $W_{ij} = 0$ and for all $(i,j) \in E$ set $W_{ij} = \max\{Q_{ij}, Q_{ji}\} + \delta_{ij}$. Where δ_{ij} is a randomly chosen number from the interval $(0,\delta)$ and can be selected by one communication between i,j.
- The algorithm variables are messages that are exchanged between neighboring nodes. Let $\hat{m}^k_{i\to j} \in \mathbb{R}$ denote message from node i to node j in iteration k.
- Initialize $k = 0$ and set the messages as follows: $\hat{m}^0_{i\to j} = W_{ij}$; $\hat{m}^0_{j\to i} = W_{ij}$.
- Algorithm is iterative, as described next.
- For $k \geq 1$, iterate as follows:

  (a) Update messages as follows:

$$\hat{m}^k_{\alpha_i\to\beta_j} = W_{ij} - \max_{\ell\neq j} \hat{m}^{k-1}_{\beta_\ell\to\alpha_i},$$
$$\hat{m}^k_{\beta_j\to\alpha_i} = W_{ij} - \max_{\ell\neq i} \hat{m}^{k-1}_{\alpha_\ell\to\beta_j}. \quad (27)$$

  (b) The estimated MWM at the end of iteration k is π^k, where $\pi^k(i) = \arg\max_{j\in\mathcal{N}(i)}\{\hat{m}^k_{\beta_j\to\alpha_i}\}$ for $1 \leq i \leq n$. But when $\max_{j\in\mathcal{N}(i)}\{\hat{m}^k_{\beta_j\to\alpha_i}\} < 0$ then let $\pi^k(i) =$ "null" which means node i chooses not to connect to any of its neighbors.

  (c) Repeat (a)-(b) till $\pi^k(i)$ converges, i.e. for each $1 \leq i \leq n$, $\pi^k(\pi^k(i)) = i$ or $\pi^k(i) =$ "null" for all k large enough.

Performance of Belief Propagation(BP). Let $\mathbf{Q}' = [Q'_{ij}]$ be a symmetric matrix of queue seizes defined by $Q'_{ij} = \max\{Q_{ij}, Q_{ji}\}$. Also, let π^* denote the MWM of matrix $\mathbf{Q}'$ and let W^* denote weight of π^*. We will prove the following result.

Theorem 6. *Given $\varepsilon > 0$, with probability one BP will converge to a matching with weight at least $W^* - \varepsilon Q^*$. The algorithm takes $O(n^2/\varepsilon\rho)$ iterations to converge with high probability, where ρ is some function of n.*

The BP algorithm performs very similar to the auction algorithm. Here, we provide a simulation run that confirms this as shown in Figure 10.

5.2 Wireless scheduling: message-passing scheduling

Here we describe message-passing algorithm for finding maximum weight independent set in order to obtain schedule in wireless networks. Clearly, it is not possible to

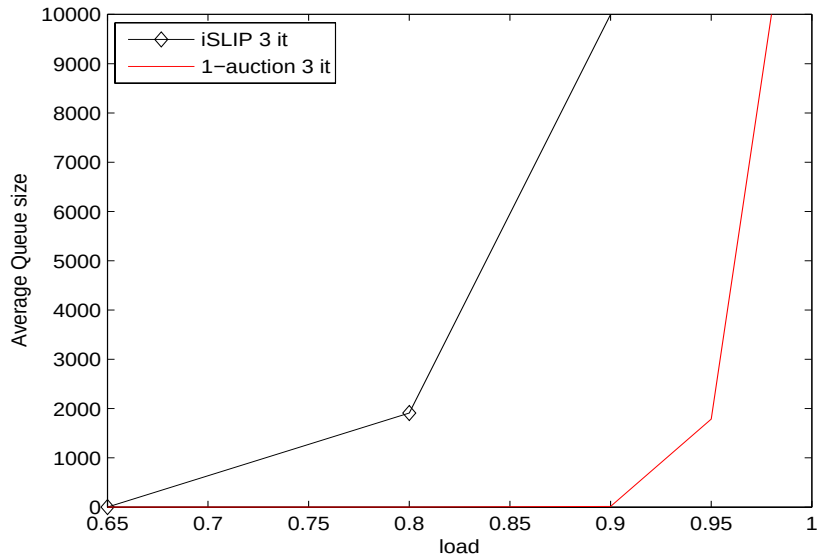

Fig. 9 Average queue sizes for iSLIP with 3 iterations and 1-Auction with 3 iterations

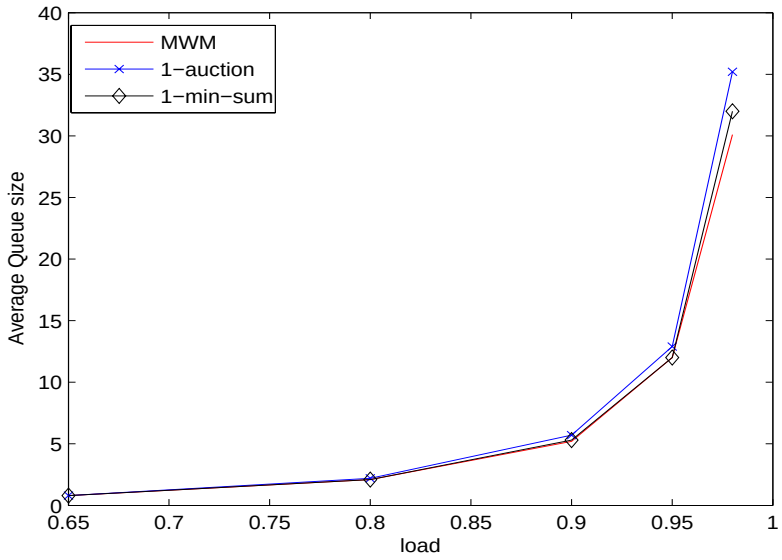

Fig. 10 Average queue sizes for MWM, 1-Auction and 1-min-sum

find such an algorithm for all possible graph structures. We describe an algorithm, based on co-ordinate descent along with a combinatorial method, that works perfectly for bipartite graphs. We will end with a heuristic based on BP, very similar to the first algorithm, that is likely to provide very good performance on other graphs. The most of the results presented in this section are from [21].

Exact algorithm: bipartite graph. Here, we describe an algorithm for finding maximum weight independent set in a given graph $G = (V,E)$ with node weights represented by $w = [w_i]$. The algorithm is essentially a modification of the standard co-ordinate descent algorithm for a "dual" of an appropriate linear programming relaxation of the integer program of maximum weight independent set. We refer an interested reader to [21] for further details. The basic algorithm, described below, invokes two sub-routines which are described next.

(o)Given (small enough) positive parameter ε, δ, run sub-routine DESCENT(ε, δ) results in an output $\lambda^{\varepsilon,\delta} = (\lambda_{ij}^{\varepsilon,\delta})_{(i,j)\in E}$ upon convergence (or close to convergence).
(i) Next, using (small enough) $\delta_1 > 0$, use EST($\lambda^{\varepsilon,\delta}, \delta_1$), to produce an estimate for the MWIS as an output of the algorithm.

Algorithm DESCENT**.** Here, we describe the DESCENT algorithm.

(o)The parameters are variables λ_{ij}, one for each edge $(i,j) \in E$. We will use notation that $\lambda_{ij}^t = \lambda_{ji}^t$. The vector λ is iteratively updated, with t denoting the iteration number.

- Initially, set $t = 0$ and $\lambda_{ij}^0 = \max\{w_i, w_j\}$ for all $(i,j) \in E$.

(i) In iteration $t+1$, update parameters as follows:

- Pick an edge $(i,j) \in E$. The edge selection is done in a round-robin manner over all edges.
- For all $(i',j') \in E, (i',j') \neq (i,j)$ do nothing, i.e. $\lambda_{i'j'}^{t+1} = \lambda_{i'j'}^t$.
- For edge (i,j), nodes i and j exchange messages as follows:

$$\gamma_{i\to j}^{t+1} = \left(w_i - \sum_{k\neq j, k\in\mathscr{N}(i)} \lambda_{ki}^t \right)_+, \quad \gamma_{j\to i}^{t+1} = \left(w_j - \sum_{k'\neq i, k'\in\mathscr{N}(j)} \lambda_{k'j}^t \right)_+.$$

- Update λ_{ij}^{t+1} as follows: with $a = \gamma_{i\to j}^{t+1}$ and $b = \gamma_{j\to i}^{t+1}$,

$$\lambda_{ij}^{t+1} = \left(\frac{a+b+2\varepsilon+\sqrt{(a-b)^2+4\varepsilon^2}}{2} \right)_+. \tag{28}$$

(ii) Update $t = t+1$ and repeat till algorithm converges within δ for each component.
(iii) Output the vector λ, denoted by $\lambda^{\varepsilon,\delta}$, when the algorithm stops.

Remark. It can be established that the update (28) turns out to be

$$\lambda_{ij}^{t+1} = \beta\varepsilon + \max\left\{-\beta\varepsilon, \left(w_i - \sum_{k\in\mathcal{N}(i)\setminus j}\lambda_{ik}^t\right), \left(w_j - \sum_{k\in\mathcal{N}(j)\setminus i}\lambda_{kj}^t\right)\right\},$$

where for some $\beta \in (1,2]$ with its precise value dependent on $\gamma_{i\to j}^{t+1}, \gamma_{j\to i}^{t+1}$.

Algorithm EST**.** The algorithm EST estimates the assignment of nodes in the maximum weight independent set based on the converged messages of the DESCENT algorithm.

(o) The algorithm iteratively estimates $\mathbf{x} = (x_i)$ given λ (expected to be a dual optimal solution).
(i) Initially, color a node i *gray* and set $x_i = 0$ if $\sum_{j\in\mathcal{N}(i)}\lambda_{ij} > w_i$. Color all other nodes with *green* and leave their values unspecified. The condition $\sum_{j\in\mathcal{N}(i)}\lambda_{ij} > w_i$ is checked as whether $\sum_{j\in\mathcal{N}(i)}\lambda_{ij} \geq w_i + \delta_1$ or not.
(ii) Repeat the following steps (in any order) till no more changes can happen:

- if i is *green* and there exists a *gray* node $j \in \mathcal{N}(i)$ with $\lambda_{ij} > 0$, then set $x_i = 1$ and color it *orange*. The condition $\lambda_{ij} > 0$ is checked as whether $\lambda_{ij} \geq \delta_1$ or not.
- if i is *green* and some *orange* node $j \in \mathcal{N}(i)$, then set $x_i = 0$ and color it *gray*.

(iii)If any node is *green*, say i, set $x_i = 1$ and color it *red*.
(iv)Produce the output $\mathbf{x}$ as an estimation.

Overall performance of algorithm ALGO**.** Here, we state the convergence, correctness and bound on convergence time of the ALGO using parameters $(\varepsilon, \delta, \delta_1)$.

Theorem 7. *The algorithm* ALGO *with parameters ε, δ converges for any choice of $\varepsilon, \delta > 0$ and for any G. The solution obtained by it is correct if G is bipartite with unique maximum weight independent set solution and $\varepsilon, \delta > 0, \delta_1$ are small enough. The convergence happens exponentially fast (constant dependent on problem size and weights through* reasonable *function).*

BP heuristic. We end this section, with brief description of the BP heuristic for maximum weight independent set. Many interesting properties of BP are known (see [21] for details). A reader is suggested to observe the extreme similarity between BP and the DESCENT algorithm.

(o)The parameters are variables $\gamma^t_{i\to j}, \gamma^t_{j\to i}$, for each $(i,j) \in E$ and iteration t. Initially, all of them are set to 0.
(i) In iteration $t+1$, update parameters as follows: for each $(i,j) \in E$,

$$\gamma^{t+1}_{i\to j} = \left(w_i - \sum_{k\neq j, k\in\mathcal{N}(i)} \gamma^t_{k\to i} \right)_+ ,$$

$$\gamma^{t+1}_{j\to i} = \left(w_j - \sum_{k'\neq i, k'\in\mathcal{N}(j)} \gamma^t_{k'\to j} \right)_+ .$$

(ii) In iteration $t+1$, estimate assignment for $i \in V$ in independent set as $\hat{x}^{t+1}_i = 1$, if $w_i > \sum_{k\in\mathcal{N}(i)} \gamma^t_{k\to i}$, and $\hat{x}^{t+1}_i = 0$, otherwise.
(iii)Update $t = t+1$ and repeat till convergence.

Key message on message-passing algorithms. The algorithmic method based on co-ordinate descent and Belief Propagation provides extremely simple, message-passing algorithms that require very little data structure (few numbers at each node) and perform few simple (addition, maximum) logical operations. Such algorithms naturally allow trade-off between performance and implementation cost by varying the number of iterations and the tuning algorithm parameters. In that sense, this provides a *universal algorithmic architecture* for a large class of scheduling problems. One may imagine running a message-passing scheduler all the time and when required network can read-off schedule based on the current message-values – thus, providing excellent pipelineability along with simple, distributed and parallel implementation.

6 Discussion and future direction

We surveyed the current state-of-art in the field of scheduling algorithms for networks with input-queued switch and wireless networks as running examples. In summary, we note three important points: (1) optimal scheduling algorithm is the MW-0^+ which can be interpreted as maximum weight maximum size scheduling; (2) designing throughput optimal simple, distributed algorithm for any problem instance is easy but obtaining small queue-size in addition is impossible for all problems; and (3) belief propagation and co-ordinate descent provide very attractive message-passing algorithmic architecture for scheduling problems.

The future work involves progress in the direction of design and analysis of scheduling algorithms. An important question in terms of analysis lies in identifying optimal algorithm beyond the fluid model scaling. In terms of design, the question lies in designing better algorithms and heuristic methods for message-passing and

simple implementation. Specifically, algorithms with easily tunable performance in various dimensions would be extremely useful.

Acknowledgements I am very grateful to all my collaborators on the topic of scheduling; without those collaborations it would not be possible for me to write this survy. I would particularly like to thank Balaji Prabhakar and Damon Wischik for numerous enlightening conversations on the topic of scheduling and network algorithms over many years: in person, on phone and more recently through skype. Finally, I would like to thank Cathy Xia and Zhen Liu for carefully reading the chapter and providing feedback to improve the readability of this chapter. I would like to acknowledge support by NSF CAREER from CNS division (on scheduling algorithms) and NSF Theoretical Foundation grant (on flow-level models) while preparing this manuscript.

References

1. Andrews, M. and Kumaran, M. and Ramanan, K. and Stolyar, A. and Vijayakumar, R. and Whiting, P. : Scheduling in a queueing system with asynchronously varying service rates. *Probability in the Engineering and Informational Sciences*, Vol. 18 (2): 191–217, (2004).
2. Bambos, N. and Walrand, J. : Scheduling and stability aspects of a general class of parallel processing systems. *Advances in Applied Probability*, Vol. 25(1) :176–202, (1993).
3. Bayati, M. and Shah, D. and Sharma, M. : Max-product for maximum weight matching: convergence, correctness and LP duality. *IEEE Information Theory Transactions*, Vol. 54 (3): 1241–1251, (2008). Preliminary versions appeareared in IEEE ISIT, (2005) and (2006).
4. Bayati, M. and Prabhakar, B. and Shah, D. and Sharma, M. : Iterative scheduling algorithms. IEEE Infocom, (2007).
5. Betsekas, D. : The auction algorithm: a distributed relaxation method for the assignment problem. *Annals of operations research*, Vol. 14: 105–123., (1988).
6. Boyd, S. and Ghosh, A. and Prabhakar, B. and Shah, D. : Gossip algorithms: design, analysis and application. In proceedings of IEEE Infocom, (2005).
7. Bramson, M. : State space collapse with application to heavy traffic limits for multiclass queueing networks. Queueing Systems **30** 89–148, (1998).
8. Dai, J. G. "Jim" : Stability of fluid and stochastic processing networks. MaPhySto Lecture Notes, (1999). `http://www.maphysto.dk/cgi-bin/gp.cgi?publ=70`
9. Dai, J. and Prabhakar, B. : The throughput of switches with and without speed-up. In proceedings of IEEE Infocom, (2000).
10. Dembo, A. and Zeitouni, O. : Large Deviations Techniques and Applications, 2nd edition, Springer, (1998).
11. Eryilmaz, A. and Srikant, R. and Perkins, J. R. : Stable scheduling policies for fading wireless channels. *IEEE/ACM Trans. Networking*, Vol. 13(2):411–424, (2005).
12. Giaccone, P. and Prabhakar, B and Shah, D. : Randomized scheduling algorithms for high-aggregate bandwidth switches. IEEE J. Sel. Areas Commun., **21(4)**, 546- 559, (2003).
13. Jung, K. and Shah, D. : Low Delay Scheduling in Wireless Network. In Proceedings of IEEE ISIT, (2007).
14. Keslassy, I. and McKeown, N. : Analysis of Scheduling Algorithms That Provide 100% Throughput in Input-Queued Switches. In proceedings of Allerton Conference on Communication, Control and Computing, (2001).
15. McKeown, N. : The iSLIP scheduling algorithm for input-queued switches. IEEE/ACM Transactions on Networking, **7(2)**, 188 – 201, (1999).
16. McKeown, N. and Anantharam, V. and Walrand, J.: Achieving 100% throughput in an input-queued switch. In Proceedings of IEEE Infocom, 296–302 (1996).
17. Meyn, S. P. and Tweedie, R. L. : Markov Chains and Stochastic Stability. Springer-Verlag, London, (1993). `http://probability.ca/MT/`

18. Modiano, E. and Shah, D. Zussman, G. : Maximizing Throughput in Wireless Network via Gossiping. In Proceedings of ACM SIGMETRIC/Performance, (2006).
19. Mosk-Aoyama, D. and Shah, D. Computing separable functions via gossip. In Proceedings of ACM PODC, (2006). Longer version to appear in *IEEE Transaction on Information Theory*, (2008).
20. Tassiulas, L. and Ephremides, A.: Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Transactions on Information Theory*, Vol. 39(2), 466-478, (1993).
21. Sanghavi, S. and Shah, D. and Willsky, A. : Message-passing for Maximum Weight Independent Set. Submitted. In Proceedings of NIPS, (2007).
22. Shah, D. : Stable algorithms for Input Queued Switches. In Proceedings of Allerton Conference on Communication, Control and Computing, (2001).
23. Shah, D. and Kopikare, M. : Delay bounds for the approximate Maximum Weight matching algorithm for input queued switches. In Proceedings of IEEE Infocom, (2002).
24. Shah, D. and Tse, D. and Tsitsiklis, J. N. : On hardness of low delay scheduling. Pre-print, (2008).
25. Shah, D. and Wischik, D. J. : Optimal scheduling algorithms for input-queued switches. In Proceedings of IEEE Infocom, (2006).
26. Shah, D. and Wischik, D. J. : Heavy traffic analysis of optimal scheduling algorithms for switches networks. Submitted. Preliminary version appeared in proceedings of IEEE Infocom, (2006). `http://www.cs.ucl.ac.uk/staff/D.Wischik/Research/netsched.html`
27. Shakkottai, S. and Srikant, R. and Stolyar, A. L. : Pathwise Optimality of the Exponential Scheduling Rule for Wireless Channels. *Advances in Applied Probability*, Vol. 36(4), 1021–1045, (2004).
28. Stolyar, A. L. : On the stability of multiclass queueing networks: A relaxed sufficient condition via limiting fluid processes. Markov Processes and Related Fields, 491–512, (1995).`http://cm.bell-labs.com/who/stolyar/stabil_mprf.pdf`
29. Stolyar, A. L. : Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Annals of Applied Probability*, Vol. 14(1), 1–53, (2004).
30. Tassiulas, L. : Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In Proceedings of IEEE INFOCOM'98, (1998).
31. Tassiulas, L. and Ephremides, A. : Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. IEEE Transactions on Automatic Control, **37**, 1936–1948 (1992).
32. Tsitsiklis, J. N. : Problems in decentralized decision making and computation. Ph.D. Thesis, Department of EECS, MIT, (1984).
33. Williams, R. : iffusion approximations for open multiclass queueing networks: sufficient conditions involving state space collapse. Queueing Systems **30** 27–88, (1998).

Index