

Optimization via Communication Networks

Matthew Andrews

Bell Laboratories

Murray Hill, NJ 07974

Email: andrews@research.bell-labs.com

Abstract—It has been known since the early 1990s that backpressure-type algorithms for communication networks (such as the Max-Weight algorithm) can be used to approximately solve static optimization problems such as Maximum Concurrent Flow problems. However, the running time of this approach is not fully understood. In this survey paper we shall describe the best currently known bounds and also compare the running time of the Max-Weight algorithm (which uses additive weights) with that of multiplicative weight schemes such as the Garg-Könemann algorithm [7].

I. INTRODUCTION

Since the seminal work of Tassioulas and Ephremides [15], [16] in the early 1990s, the backpressure algorithm for scheduling packets in multihop communication networks has been known to provide a method to keep queues stable as long as no queue is overloaded. However, the very fact that they are able to do this means that they are inherently solving an underlying multicommodity flow problem. Parallel work that also took place in the early 1990s due to Awerbuch and Leighton [4], [5] proposed this algorithm as a “local-control” method for solving multicommodity flow problems. Awerbuch and Leighton also analyzed the running time of this algorithm as a function of the problem size and the eventual error bound that we wish to achieve. One can view this algorithm as one in which the “link weights” (i.e. the queue sizes) are updated in an additive manner. An alternative approach that has been studied in a number of papers (including Garg-Könemann [7]) carries out the weight updates in a multiplicative manner. Multiplicative weight algorithms can also solve multicommodity flow problems and packing problems to within an arbitrarily small error.

The purpose of this note is to give an overview of these two types of algorithms for solving a large class of problems called packing Linear Programs. Secondly, we would like to study the running time of the algorithms as a function of the network size and the error parameter ε . Although the fact that the backpressure algorithm and others like it can solve packing LPs and multicommodity flow problems is well known in the networking community, we believe that the exact analysis of the running time of these algorithms is perhaps less well-known. Lastly, we would like to study whether the standard analysis of the running times is tight. Most of the traditional analyses show a running time whose dependence on the error term is of the form $O(1/\varepsilon^2)$. We study the extent to which this running time can actually occur.

For the most general form of a packing problem we are given an $m \times n$ matrix A , an $m \times 1$ vector b and a polytope

P . The matrix A satisfies $A_j x \geq 0$ for all rows A_j and all $x \in P$. We wish to find a point $x \in P$ for which the vector Ax can be “packed” into b to the greatest extent possible. More formally, we wish to solve,

$$\begin{aligned} & \max \lambda \\ & \text{subject to } \lambda Ax \leq b \\ & x \in P \end{aligned}$$

Throughout this paper we shall denote the value of the optimal solution by λ^* .

A special case of a packing Linear Program is the following Maximum Concurrent Flow (MCF) problem for routing flow in networks. We are given a graph with n vertices and m links in which links are indexed by (u, v) and have capacity c_{uv} . We also have a set of k demands. Each demand i has a size f_i and must be routed from a source s_i to a destination d_i . The formal definition of a Maximum Concurrent Flow problem is,

$$\begin{aligned} & \max \lambda \\ & \text{subject to } \sum_i x_{uv}^i \leq c_{uv} \quad \forall v, u \\ & \sum_u (x_{vu}^i - x_{uv}^i) = \begin{cases} \lambda f_i & v = s_i \\ -\lambda f_i & v = d_i \\ 0 & \text{o.w.} \end{cases} \quad \forall i. \end{aligned}$$

The Maximum Concurrent Flow problem can be cast in the form of a general packing LP if we let the polytope P represent the flow conservation constraints and the matrix-vector pair A, b represent the capacity constraints.

We remark that we could of course use a general LP algorithm such as the simplex algorithm or an interior-point method to solve these problems. However, in recent years a great deal of attention has been paid to algorithms of *Dantzig-Wolfe* type. These are combinatorial algorithms that can efficiently solve packet LPs to within a factor $1 - \varepsilon$ in time that is a small function of ε . These algorithms were first considered by Shahrokhi and Matula [14] and were later studied by many researchers including Klein et al. [11], Leighton et al. [12], Goldberg [8], Radzik [13], Grigoriadis and Khachiyan [9], and Young [17].

The general form of Dantzig-Wolfe algorithm is as follows. It operates iteratively and in each iteration t we maintain a vector $q(t)$ (that we sometimes call a *virtual queue* vector) that corresponds to variables in the associated dual linear program. A Dantzig-Wolfe algorithm operates by making calls to an oracle that answers queries of the form, “Given a vector q , find $x \in P$ that minimizes $q \cdot x$.” If we let $x(t)$ be the

corresponding vector found in iteration t , the eventual solution to the packing Linear Program is a suitable linear combination of all the vectors $x(t)$.

In this paper we survey two different types of Dantzig-Wolfe algorithms. The first is a generalization of the well-known Max-Weight algorithm for scheduling packets in communication networks. For this algorithm we update weights in an additive manner. The second type of algorithm is a multiplicative weights update algorithm. In particular we focus on the well-known algorithm due to Garg and Könemann. In this survey we give a basic description of these algorithms, after which we analyze the running time. We then show that the standard analysis of the running time of Garg-Könemann is tight in terms of its dependence on ε . Lastly we show that the running time of the Max-Weight algorithm can be improved in terms of ε at the expense of a worse dependence on network size. These last two results are described in more detail in the paper [2].

We remark that Dantzig-Wolfe algorithms with better running times than Garg-Könemann for the Maximum Concurrent Flow problem were subsequently obtained Fleischer [6] and Karakostas [10]. However, the improvements in running time were in the dependence on the network parameters rather than the dependence on ε . Since our primary focus is on the dependence on ε , we focus on the Garg-Könemann approach for simplicity. We also remark that a survey of multiplicative weights methods for many different types of problems may be found in Arora et al [3].

II. DEFINITION OF MAX-WEIGHT ALGORITHM

We begin with the definition of the Max-Weight algorithm for the special case of Maximum Concurrent Flow problems where we know the value of λ^* . In this case we can solve the problem via a communication network in which packets are injected into demand i at its source at a rate of $(1 - \frac{\varepsilon}{2})\lambda^* f_i$. Each node maintains a queue for packets from demand i . (It is possible to reduce the number of queues at a node by aggregating all the queues for demands that have the same destination. However, we will not consider that simplification here.) We let Q_v^i denote the queue at node v for demand i and we let $q_v^i(t)$ denote the size of this queue at time t . At time t each link uv calculates the maximum queue differential $\max_i (q_u^i - q_v^i(t))$. If the value of this maximum is non-negative then node v sends c_{uv} packets from queue Q_u^i to Q_v^i , otherwise no packets are transmitted across the link.¹ The fact that a large value of q_v^i will discourage q_u^i from sending packets to it is the reason that this version of the Max-Weight algorithm for Maximum Concurrent Flow problems is sometimes called the backpressure algorithm.

The standard analysis of the Max-Weight algorithm uses the Lyapunov function $\sum_i (q_v^i(t))^2$ to show that the queue sizes

¹We remark that strictly speaking this algorithm is not of Dantzig-Wolfe type since the packets that cross the links in a given time step do not necessarily correspond to a flow vector (i.e. an element of the flow polytope P). However, over the long run the movement of packets across the links will provide a flow vector.

always remain stable. In particular, by looking at how $L(t)$ changes in each time step it can be shown that $L(t)$ is never larger than $\frac{2}{\varepsilon}mC^2$ where $C = \max_{uv} c_{uv}$. Over T time steps the number of demand i packets injected into the network is at least $(1 - \frac{\varepsilon}{2})\lambda^* f_i T$. At all times at most $\frac{2}{\varepsilon}mC^2$ of these packets are yet to reach their destination. If $T \geq \frac{4mC^2}{\varepsilon^2 \lambda^* \min_i f_i}$ then this number is at most $\frac{\varepsilon}{2}\lambda^* f_i T$. Hence if we set x_{uv}^i to be the fraction of demand i packets that have crossed link uv and that have reached the destination, these values will define a solution to the Maximum Concurrent Flow problem that is within a factor $1 - \varepsilon$ of optimal. Note that the dependence of the running time on ε is $O(1/\varepsilon^2)$.

Note that the above description relies on knowledge of λ^* . If this value is not known the usual strategy is to find it via binary search. However, in the following we describe a Generalized Max Weight (GMW) algorithm that discovers the value of λ^* as the algorithm proceeds and can also be used for general packing Linear Programs.

The GMW algorithm is also an iterative algorithm. For iteration t we maintain a “virtual queue counter” $q_j(t)$ that corresponds to constraint j . We assume that we are able to apply a Dantzig-Wolfe algorithm, i.e. we have access to an oracle that answers queries of the form, “Given a vector q , find $x \in P$ that minimizes $q \cdot x$.”

For the GMW algorithm we define a length vector by,

$$\ell_i(t) = \sum_j A(j, i) q_j(t),$$

and then appeal to the oracle to find the $x \in P$ that minimizes $\ell(t) \cdot x$. We set $x(t)$ to be this x and define the eventual solution by $X(t) = \frac{1}{t} \sum_{t' \leq t} x(t')$. We also update our current estimate for the value of λ by

$$\lambda(t) = \sup\{\lambda : q(t) \cdot (-b + \lambda Ax(t)) \leq 0\}.$$

(It is shown in [2] that this λ is always be an upper bound on the optimal solution λ^* . Moreover, in an implementation we can restrict our attention to a discretized set of values.) Lastly we update the vector of virtual queue counters $q(t)$ by,

$$q(t+1) = \max\{0, q_j(t) - b_j + \sum_i \lambda(t) A(j, i) x_i(t)\}.$$

It is shown in [2] that the number of iterations required to obtain a solution that is within a factor $1 - \varepsilon$ of optimal is $O(1/\varepsilon^2)$ for fixed instances. More precisely, the running time is $O(\frac{1}{\varepsilon^2} \rho^2)$ where ρ is the *width* of the problem defined by $\rho = \max_{j,x} \lambda_0 A_j x / b_j$, where λ_0 is some easily computed upper bound on λ^* . (For example for the Maximum Concurrent Flow problem we can easily compute a λ_0 such that $\rho = k$, the number of demands.) However, [2] also shows using a geometric argument that for fixed packing problems the number of required iterations is actually $O(1/\varepsilon)$ for small values of ε . We give an overview of this analysis in Section V.

III. DEFINITION OF GARG-KÖNEMANN ALGORITHM

The Garg-Könemann algorithm is also an iterative algorithm that uses a weight $q_j(t)$ for each constraint. However, for this

algorithm that weights are updated in a multiplicative manner rather than an additive manner. For simplicity, in this paper we will focus on the Garg-Könemann algorithm when it is applied to the Maximum Concurrent Flow problem rather than a general packing LP. For the Maximum Concurrent Flow problem we have a weight q_{uv} associated with each link. The definition of the Garg-Könemann algorithm is parameterized by ε . We shall therefore refer to a particular instantiation of the algorithm as $GK(\varepsilon)$.

In each iteration of $GK(\varepsilon)$ we go through demands one by one. For each demand i we repeatedly find the shortest path from s_i to t_i with respect to the weights q_{uv} and route as much flow as possible subject to the minimum capacity link on the path. Whenever we route demand of size r through link uv then we update q_{uv} by $q_{uv} \leftarrow q_{uv}(1 + \varepsilon r/c_{uv})$. Once we have routed f_i units of demand i we move onto the next one. Initially, each q_{uv} is set to a common value $\delta = (m/(1 - \varepsilon))^{-1/\varepsilon}$. We terminate the full procedure after the first iteration for which $\sum_{uv} c_{uv} q_{uv} \geq 1$. This occurs for a value of T no greater than $\frac{\lambda^*}{\varepsilon} \log_{1+\varepsilon} \frac{m}{1-\varepsilon} = O((\lambda^* \log m)/\varepsilon^2)$ iterations. (Although the running time does depend on the value of λ^* , this dependence can be removed by appropriate scaling of the original capacities. The total number of iterations then becomes $O(\frac{\log k \log m}{\varepsilon^2})$.) Once the algorithm has terminated it is probable that many of the link capacities are violated. We hence scale down all components of the solution by a common factor until no remaining capacities are violated.

The analysis of $GK(\varepsilon)$ is sketched below. The idea is to first upper bound the amount that the dual solution increases in each iteration. We can then lower bound the amount that q_{uv} increases as flow of size c_{uv} is routed through link uv . By trading off these two quantities we can argue that after T iterations we have a solution that differs from the optimum by at most a factor $(1 - \varepsilon)^3 = 1 - O(\varepsilon)$.

Note that dependence of the running time of $GK(\varepsilon)$ on ε has the form $O(1/\varepsilon^2)$. Note also that unlike GMW the definition of the algorithm depends on the value of ε . Two interesting questions therefore are: Is this running time tight and is this dependence necessary. In Section IV-B below we describe an example from [2] which shows that unlike GMW:

- For some problems the quality of the solution obtained by $GK(\varepsilon)$ is never better than a $1 - \varepsilon$ factor from optimal.
- There is a fixed instance for which all instantiations of Garg-Könemann take time $\Omega(1/\varepsilon^2)$ to obtain a solution that is within a factor $1 - \varepsilon$ of optimal.

IV. ANALYSIS OF THE GARG-KÖNEMANN METHOD

In this section we sketch the analysis of the Garg-Könemann method that was presented in [7] and we describe an example presented in [2] which shows that their analysis is tight with respect to the dependence on ε .

A. Upper bound

At a high level, the analysis of the Garg-Könemann method presented in [7] proceeds as follows. First, the dual of the

Maximum Concurrent Flow problem is to minimize the unconstrained function $D(q)/\alpha(q)$ where $D(q) = \sum_{uv} c_{uv} q_{uv}$, $\alpha(q) = \sum_i f_i \text{dist}_i(q)$ and $\text{dist}_i(q)$ is the shortest path distance between s_i and t_i with respect to q . This can be seen by applying the complementary slackness condition to the regular Linear Programming formulation of the dual.

The performance of the algorithm can be derived from the manner in which this unconstrained dual function grows as the algorithm progresses. The first important fact is that since $D(t)/\alpha(t) \geq \lambda^*$, we can obtain a bound on the rate that $D(t)$ increases since we know that $D(t) \leq D(t - 1)/(1 - \varepsilon/\lambda^*)$. Hence we know that after T iterations we have $D(T) \leq \frac{m\delta}{1-\varepsilon} e^{\varepsilon(T-1)/\lambda^*(1-\varepsilon)}$. On the other hand we know that whenever we route flow of size at least c_{uv} through link uv , the weight q_{uv} increases by at least a factor $1 + \varepsilon$. Since the total increase in $D(t)$ through T iterations is at most $1/m\delta$, the total increase in q_{uv} through that number of iterations is at most $1/\delta$. Therefore the total amount of flow routed through any link uv is at most $\log_{1+\varepsilon} 1/\delta$. Hence after scaling the flow down the total flow routed for demand i is at least $f_i T / \log_{1+\varepsilon} 1/\delta$. Therefore for the chosen values of δ and T we have a solution of value at least $(1 - \varepsilon)\lambda^*$.

B. Lower Bound

We now show that the running time of Garg-Könemann of the form $O(1/\varepsilon^2)$ is actually tight. In particular, we present an example from [2] which for which the quality of the solution is never better than $(1 - \varepsilon)$ from optimal and for which in order to get that close we need to run for $\Omega(1/\varepsilon^2)$ iterations. The example consists of two columns of links. The left column has six links and the right column has three links. (See Figure 1.) For each link in the left column there are two demands of size two that wish to cross that link only (i.e. the demand is between the head of the link and the tail of the link). For each link in the right column there is a single demand of size three. We also have one demand of size one that wishes to go from the top node to the bottom node. (The two columns meet at the top and the bottom.)

We assume that each link is directed from top to bottom. Hence there is no choice regarding how to route any of the “one-hop” demands. They all must go through their associated one-hop path. The only choice remaining is with regards to how to route the longer demand. Clearly the optimum solution is to route it down the left path. However, Garg-Könemann parameterized by ε (which we refer to as $GK(\varepsilon)$, sometimes routes the long demand down the right path for the following reasons.

- At the beginning the weight of all links is the same. Therefore initially the algorithm views the right column as the shortest path. This behavior continues until the link weights get close to the optimum dual solution in which the weight of the right hand links is twice the weight of the left hand links.
- Even when the weights converge to something close to the dual optimum solution, the method of updates means that deviations can occur. In particular, suppose that the

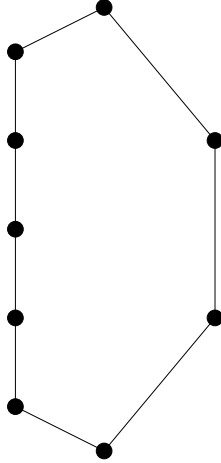


Fig. 1. The two column network.

algorithm makes the correct choice in an iteration and routes the long demand down the left path. In this case the weight on each link on the left path increases by a factor $(1 + \varepsilon)$ whereas the weights on the links on the right path increase by a factor $(1 + \varepsilon/3)^3$. The difference between these two quantities causes the weights to drift further and further away from the optimum dual solution. As some point the weights on the two possible paths become sufficiently different that GK(ε) once again routes down the right-hand column.

The basic behavior of GK(ε) on this example is as follows. For the first $O(1/\varepsilon)$ time steps the algorithm routes the long demand down the right path due to the first reason mentioned above. At that point the dual solution becomes sufficiently close to the correct values so that the algorithm starts to route mostly down the left-hand path (which is of course the correct solution). However, due to the second reason mentioned above, the algorithm does not use the correct solution indefinitely. Every $O(1/\varepsilon)$ iterations the link weights diverge sufficiently far from the optimal values for the algorithm to route the long demand down the wrong path.

Therefore, the algorithm is inherently running at a deficit of an $1 - \varepsilon$ factor. However, in order to get close to optimal it has to wipe out the influence of the first $O(1/\varepsilon)$ in steps in which the algorithm *always* makes the wrong choice. The eventual number of steps required to negate this influence is $O(1/\varepsilon^2)$.

V. GEOMETRIC ANALYSIS OF THE GENERALIZED MAXWEIGHT ALGORITHM

In this section we outline a geometric interpretation of the GMW algorithm that was derived in [2] and was based on an earlier analysis in [1] for the Max Weight algorithm in communication networks. It shows that for small values of ε , the discrete nature of the space in which the algorithm operates means that eventually the running time to get within a factor $(1 - \varepsilon)$ of optimal becomes $O(1/\varepsilon)$. We work with

two geometric spaces. The first is the space A of virtual queue vectors. The second is the space B of possible changes to the virtual queue counters. The space will actually change as λ changes since the updates to the virtual queue counter depend on λ .

The key to interpreting the GMW algorithm is that at each time step it updates the virtual queue counters using the vector $s(t)$ in the space B that minimizes $s(t) \cdot q(t)$. Then the queue vector is updated according to $q(t+1) = q(t) - s(t) + a(t)$ where $a(t)$ is a vector with bounded components at most c that represents an adjustment to the queue update process that occurs due to the fact the virtual queue counters cannot become negative.

Let \bar{A} be the set of queue vectors in A that do not make an acute angle with any vector in B . (If the current value of λ equals λ^* then \bar{A} is equivalent to the set of optimum solutions to the dual of the packing Linear Program.) Let $q'(t)$ be the vector in \bar{A} that is closest to $q(t)$. The $O(1/\varepsilon)$ running time of GMW follows from the following facts.

- For a fixed packing LP, the rate of growth of the function $q(t) \cdot q(t)$ is at most $O(1)$. This implies that the norm of the vector $q(t)$ is at most $O(\sqrt{t})$.
- The distance between vector $q(t)$ and $q'(t)$ is bounded.
- There exists a vector $w(t)$ in \bar{A} such that whenever $q'(t)$ changes, $w(t) \cdot q(t)$ increases by some value bounded away from zero.
- For any parameter S , there is sequence of S consecutive iterations during which $q'(t)$ is fixed. During this interval $q(t)$ stays in a fixed ball surrounding $q'(t)$. Moreover, the number of possible values for $q(t)$ in the ball is finite. Hence for S sufficiently large, $q(t)$ must repeat itself during this interval.

Due to the deterministic nature of the system once $q(t)$ has repeated itself once it will continue to loop indefinitely and $q(t) \cdot w(t)$ will never grow. From this we can obtain a bound on the norm of $q(t)$ that is independent of ε . At all times the error in our solution is proportional to the size of the largest component of $q(t)/t$. Hence the number of iterations required to obtain of solution that is within a factor $1 - \varepsilon$ of optimal is at most $O(1/\varepsilon)$. We remark however that a significant drawback of this analysis is that although the bound on the norm of $q(t)$ obtained from this analysis is independent of ε , it is exponential in m . Hence this linear running time is only an improvement over the earlier quadratic running times when ε is exponentially smaller than m . We believe that an extremely interesting open question is whether or not the analysis of GMW can be adapted to give a running time that is both linear in $1/\varepsilon$ and polynomial in m .

VI. CONCLUSIONS

In this article we have surveyed two popular methods for deriving approximate solutions to the class of packing Linear Programs (which includes Maximum Concurrent Flow problems). The first is an additive weights method which in the special case of Maximum Concurrent Flow problems is equivalent to the well-known backpressure method for

scheduling packets in communication networks. The second is a multiplicative weights method due to Garg and Könemann. The main features of the performance of these algorithms are that they both provide a solution that is within a factor $1 - \varepsilon$ in time $O(1/\varepsilon^2)$. The downside of the Garg-Könemann algorithm is that for the definition of the algorithm parameterized by ε , the solution that is obtained by the algorithm is never better than a factor $1 - \varepsilon$ from optimal. The downside for the Generalized Max-Weight algorithm is that the running time depends on the width of the problem ρ . The advantage of the Garg-Könemann algorithm is that its running time does not depend on the width. The advantage of GMW is that one single instantiation of the algorithm can achieve a solution arbitrarily close to optimal. Moreover, for small values of ε the running time actually takes the form $O(1/\varepsilon)$.

REFERENCES

- [1] M. Andrews, K. Jung, and A. Stolyar. Stability of the Max-Weight routing and scheduling protocol in dynamic networks and at critical loads. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, 2007.
- [2] M. Andrews. Approximately solving fixed packing problems in time linear in the error. Submitted, 2008.
- [3] S. Arora, E. Hazan and S. Kale. The multiplicative weights update method: A meta-algorithm and applications. Survey paper manuscript, 2005.
- [4] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 459–468, 1993.
- [5] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 487–496, 1994.
- [6] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 24–31, New York, NY, 1999.
- [7] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 300 – 309, Palo Alto, CA, November 1998.
- [8] A. Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. *Information Processing Letters*, 42:249–256, 1992.
- [9] M. Grigoriadis and L. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4(1):86–107, 1994.
- [10] G. Karakostas. Faster approximation schemes for fractional multicommodity flow problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 166–173, San Francisco, CA, 2002.
- [11] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23(3):466–487, 1994.
- [12] T. Leighton, F. Makedon, S. Plotkin, C. Stein, S. Tragoudas, and E. Tardos. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50:228 – 243, 1995.
- [13] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 486–492, 1995.
- [14] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, 1990.
- [15] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936 – 1948, December 1992.
- [16] L. Tassiulas and A. Ephremides. Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Transactions on Information Theory*, 30:466 – 478, 1993.
- [17] N. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.