Maximizing throughput in queueing networks with limited flexibility[*]

Douglas G. Down[†] George Karakostas[‡]

March 16, 2007

Abstract

We study a queueing network where customers go through several stages of processing, with the *class* of a customer used to indicate the stage of processing. The customers are serviced by a set of *flexible* servers, i.e., a server is capable of serving more than one class of customers and the sets of classes that the servers are capable of serving may overlap. We would like to choose an assignment of servers that achieves the maximal capacity of the given queueing network, where the maximal capacity is λ^* if the network can be stabilized for all arrival rates $\lambda < \lambda^*$ and cannot possibly be stabilized for all $\lambda > \lambda^*$. We examine the situation where there is a restriction on the number of servers that are able to serve a class, and reduce the maximal capacity objective to a maximum throughput allocation problem of independent interest: the TOTAL DISCRETE CAPACITY CONSTRAINED PROBLEM *(TDCCP)*. We prove that solving *TDCCP* is in general NP-complete, but we also give exact or approximation algorithms for several important special cases and discuss the implications for building limited flexibility into a system.

Keywords: Queueing, scheduling, approximation algorithms, flexible servers.

1 Introduction

Suppose that one is presented with a queueing network with flexible servers. By flexibility, we mean that if the *class* of a customer is used to indicate the stage of processing, then a server may be capable of serving more than one class of customer and the sets of classes that the servers are capable of serving may overlap. Such models have arisen in areas that include production scheduling (Hillier and So [15] is but one example), power control for wireless networks (Armony and Bambos [5]), and parallel computer systems (Squillante et al. [27]).

The performance measure of interest in our work is the maximal capacity of a given network, defined to be λ^* if the network can be stabilized for all arrival rates $\lambda < \lambda^*$ and cannot possibly

[*]An extended abstract of this paper appeared in the Proceedings of LATIN'06 [11].

[†]Corresponding Author. Address: Dept. of Computing & Software, McMaster University, 1280 Main Street West, Hamilton, ON, Canada L8S 4L7, Tel: +1 905 525 9140, Fax: +1 905 524 0340. e-mail:`downd@mcmaster.ca`. Research supported by an NSERC Discovery grant.

[‡]Address: Dept. of Computing & Software, McMaster University, 1280 Main Street West, Hamilton, ON, Canada L8S 4L7, Tel: +1 905 525 9140, Fax: +1 905 524 0340. e-mail:`karakos@mcmaster.ca`. Research supported by an NSERC Discovery grant.

be stabilized for all $\lambda > \lambda^*$. A number of authors have examined flexible server systems with throughput as a performance measure. These include Tassiulas and Ephremides [29], Tassiulas and Bhattacharya [28], Andradóttir, Ayhan, and Down [3, 4], Andradóttir and Ayhan [2], Bischak [7], Dai and Lin [9], Zavadlav, McClain, and Thomas [31], and Ostolaza, McClain, and Thomas [21]. None of these papers study the problem when there is a constraint on the number of servers that may be at a class. We believe that this is the first attempt to address such a problem. For a more extensive overview of the flexible server literature (including works that look at other performance measures), see [4] and Hopp and van Oyen [17].

In the area of queueing networks, the use of fluid limits to characterize stability is a standard technique, originating in the work of Rybko and Stolyar [23] and Dai [8]. The central idea in this approach is that one can equate stability of a (stochastic) queueing network with that of a related deterministic model (the fluid model). Most of the work in this paper addresses the deterministic model that arises from the stochastic network. We stress that this deterministic model is the result of a formal limiting process and not an approximation, so results for the deterministic model have direct implications for that of the stochastic network.

For the flexible server setting, the fluid limit methodology has been used to break down the determination of maximal capacity to two steps (see [4] and [9]).

1. Determine the maximal capacity of the fluid model and an optimal allocation of each server's effort.

2. Use the allocation to construct a scheduling policy for the original network.

The framework in [4] gives a standard means by which to perform the second step. Also, if there is no constraint on the number of servers that can be at a class at any one time, it is shown in [4] that for the problem that we consider in this paper, the first step reduces to solving a linear programming problem.

In this work, we wish to examine what happens when there is a constraint on the number of servers that may ever be allowed to serve a class. There are several reasons for studying such a model. A designer can explore the impact of less flexibility (and hence reduced training costs) by considering reduced training levels at each class. It also allows for the design of decentralized scheduling policies that respect a constraint on the number of servers at a class at any point in time. (Note that the problem of designing polices that require server coordination to satisfy such a constraint should lead to better performance. The resulting required server synchronization makes the problem significantly more difficult and is a topic for future consideration.) There is a negative side to our result, in that a very simple means of limiting flexibility results in a computationally challenging problem, so it can only be expected that related problems will provide similar challenges. There has been much recent work on designing and evaluating stochastic systems with limited flexibility, see for example (the list is not exhaustive) Akşin and Karaesmen [1], Gurumurthi and Benjaafar [14], Hopp, Tekin, and van Oyen [16], Sheikhzadeh, Benjaafar, and Gupta [24], Jordan and Graves [18], and Wallace and Whitt [30]. The specific problem that we will examine we will call the Total Discrete Capacity Constrained Problem *(TDCCP)*. For this problem, the second step in the above procedure is unchanged. It is the first step in the procedure which sees the most significant change, and the resulting optimization problem is the focus of this paper. We show that this problem is NP-complete even for special cases. Hence we look for approximation algorithms for such hard special cases and for the general problem. We achieve an approximation factor of 1/10 for the important case of service rates that depend only on the servers (and not on the classes)[1], and these approximation techniques extend also to the general case (but with a worse approximation factor). Even more importantly, some of these techniques give exactly the same approximation factors for the *budgetary constraint* version of the problem. In this generalization, a *per service unit cost* of assigning a particular server to a particular class is given, as well as a budget that our total assignment cost should not exceed. Some of our approximation algorithms produce solutions that are within the previous approximation factors without violating the budget.

With the ability to evaluate the maximal capacity of a network under a given set of constraints, our framework is a step towards allowing one to study the design problem of choosing the number of servers that are able to serve a class to satisfy some optimality criterion (which would typically trade off network performance and costs).

The structure of the paper is as follows. Section 2 provides details of the model under study. Section 3 describes the deterministic optimization problem whose solution characterizes the maximal capacity of the network. Section 4 looks at special cases of the problem, in particular, when the service rates depend only on the class or only on the server. The general case is discussed in Section 5. The proof of NP-completeness for the problem with heterogeneous servers is shown in Section 6. Section 7 contains three numerical examples, which serve to show that our proposed algorithm's performance can be well above its worst-case guarantee as well as providing some insight into the issue of the difficulty of designing flexibility structures for networks with heterogeneous servers. Finally, Section 8 provides concluding remarks.

2 Queueing Network Model

The model we consider is a generalization of that in Andradóttir, Ayhan, and Down [4]. For completeness, we present the model in its entirety.

2.1 Network topology

Consider a network where the location of a customer is given by its class k. We assume that there are K distinct classes, with a buffer of infinite size for each class. Arrivals to a class may occur from inside or outside of the network. Customers arriving from outside of the network do so according to an arrival process with independent and identically distributed (i.i.d.) interarrival times $\{\xi(n)\}$. The associated arrival rate is $\lambda = 1/E[\xi(1)]$. An arrival from outside of the network is routed to class k with probability $p_{0,k}$, with $\sum_{k=1}^{K} p_{0,k} = 1$. Within the network, customers circulate as follows. Upon completion of service at class i, a customer becomes one of class k with probability $p_{i,k}$. The customer leaves the network with probability $1 - \sum_{k=1}^{K} p_{i,k}$. We define the routing matrix P to have (i, k) entry $p_{i,k}$ for $i, k = 1, \ldots, K$ and I to be the $K \times K$ identity matrix. We assume that all customers eventually leave the network, which is equivalent to $(I - P')$ being invertible. (Note that the (i, k) entry of $(I - P')^{-1}$ is the expected number of future visits to class k of a class i customer.)

[1]In Section 4.1 we show that the special case of service rates which depend only on the classes can be solved optimally.

For technical reasons, we assume that the interarrival times are unbounded and spread out. In other words, there exists some non-negative function $q(x)$ on $\mathbb{R}_+$ and some integer ℓ such that

$$\begin{aligned} P(\xi(1) > x) &> 0 \text{ for all } x > 0, \\ P(\xi(1) + \cdots + \xi(\ell) \in dx) &\geq q(x)dx \text{ and } \textstyle\int_0^\infty q(x)dx > 0. \end{aligned}$$

These assumptions imply that the origin is reachable for the Markov process that describes the queueing network. These could be replaced by the assumption that a reachable compact set exists for the Markov process (see Meyn and Down [20] and Sigman [26]). Note that these assumptions will not be mentioned explicitly in the remainder of the paper. They arise in the proof of Theorem 1, see [4].

2.2 Service mechanism

The network is populated by M servers which service customers within a class according to First Come, First Served order. When switching from class i to class k for the nth time, server j incurs a (possibly zero) switching time of $\zeta^j_{i,k}(n)$. It is assumed that the sequence $\{\zeta^j_{i,k}(n)\}$ is i.i.d. for every $j = 1, \ldots, M$ and $i, k = 1, \ldots, K$. Further, we assume that $\{\zeta^j_{i,i}(n)\}$ is identically zero for all i and j.

Several servers may be simultaneously at a class, in which case they work in parallel (on different customers). If server j is capable of working at class k, the service time of the nth customer served by server j at class k is given by $\eta_{j,k}(n)$, where the sequence $\{\eta_{j,k}(n)\}$ is assumed to be i.i.d. for each j and k. The associated mean service time for server j at class k is $m_{j,k} = E[\eta_{j,k}(1)]$, with associated service rate $\mu_{j,k} = 1/m_{j,k}$. If server j cannot work at class k, we set $\mu_{j,k} = 0$. We only consider nonpreemptive policies.

The difference between the model in [4] and that considered here is that we put an upper limit, $c_k \leq M$, on the number of servers that can be assigned to a class (a server is assigned to a class if it spends any time at class k).

3 Total Discrete Capacity Constrained Problem

We are first interested in computing the *capacity*. A network operating under a service policy π is said to have capacity λ^π if the system is stable for all values of the arrival rate $\lambda < \lambda^\pi$. We wish to calculate a tight upper bound on the capacity that a given system can achieve (called the *maximal capacity*). In the course of doing so, we identify a means to construct server assignment policies that have capacity that is arbitrarily close to the maximal capacity.

3.1 The Allocation Program

First, we solve the traffic equations for the network, which give the expected number of visits to class k, a_k. Here we have

$$a_k = p_{0,k} + \sum_{i=1}^{K} p_{i,k} a_i,$$

for $k = 1, \dots, K$. This system of equations is known to have a unique solution if $(I - P')$ is invertible. If the system is stable, then $\lambda_k = \lambda a_k$ is the total arrival rate to class k.

Let $\delta_{j,k}$ be the proportion of time that server j is working at class k. The resulting optimization problem (with variables $\delta_{j,k}$ and λ) that will give us the assignment of servers to classes is:

$$\max \lambda \tag{AP}$$

$$\text{s.t.} \sum_{j=1}^{M} \mu_{j,k}\delta_{j,k} \geq \lambda a_k, \;\; k = 1, \dots, K \tag{1}$$

$$\sum_{k=1}^{K} \delta_{j,k} \leq 1, \;\; j = 1, \dots, M, \tag{2}$$

$$\delta_{j,k} \geq 0, \;\; k = 1, \dots, K, j = 1, \dots, M, \tag{3}$$

$$\sum_{j=1}^{M} \mathbf{1}\{\delta_{j,k} > 0\} \leq c_k, \;\; k = 1, \dots, K, \tag{4}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. The constraints in (AP) have the following interpretations. The first, (1), says that the service rate allocated to class k must be at least as large as the arrival rate. The second and third constraints, (2) and (3), prevent over allocations and negative allocations of a server, respectively. Finally, the constraint (4) limits the flexibility, by only allowing c_k servers to be assigned to work at class k. Let a solution of (AP) be given by λ^*, $\{\delta^*_{j,k}\}$. We will see that λ^* is the desired maximal capacity and $\{\delta^*_{j,k}\}$ is the set of proportional allocations of server j to classes k required to achieve λ^*. The solution of (AP) will allow us to construct a dynamic control for the original queueing network that can achieve a capacity λ^π arbitrarily close to the maximal capacity.

Obviously, the difficulty in solving (AP) comes from the integral constraints (4). (If (4) is removed, then (AP) reduces to a linear program, studied in [4].) Note that in these constraints, although the allocation variables $\delta_{j,k}$ are fractional, the capacity each one is allocated is either 0 or 1 (depending on whether $\delta_{j,k}$ is 0 or not). To the best of our knowledge, we are not aware of other scheduling problems with such constraints. In Section 6 we show that even a simpler variant of the problem is NP-complete. First we consider special cases in Section 4: If the $\mu_{j,k}$'s are independent of j, i.e., $\mu_{j,k} = \mu_k$ for all j, then the problem can be solved in polynomial time. If the $\mu_{j,k}$'s are independent of k, i.e., $\mu_{j,k} = \mu_j$ for all k, then the problem is NP-complete, but can be approximated within a factor 1/10 by a reduction to the k-splittable flow problem [6]. For the general case, we show in Section 5 that in polynomial time one can find a solution within a factor $1/10w_{max}$, where $w_{max} := \max_j \max_{k_1,k_2} \frac{\mu_{j,k_1}}{\mu_{j,k_2}}$. The bulk of the remainder of the paper is concerned with how one can solve (AP). Before doing this, we complete the connection between solving (AP) and the problem of finding the maximal capacity in the original queueing network.

For the original queueing network, we consider the set of generalized round robin policies. A generalized round robin policy π is given by a set of integers $\{\ell^\pi_{j,k}\}$ and an ordered list of classes V^π_j. Server j serves classes in V^π_j in cyclic order, with server j performing $\ell^\pi_{j,k}$ services at each class in V^π_j (unless server j idles, in which case the server moves to the next class in V^π_j). If the classes in V^π_j are all empty, the server idles at an arbitrary class in V^π_j. The details of how to construct a generalized round robin policy π given a set of required proportional allocations $\{\delta^*_{j,k}\}$ is given in Section 3.3 of [4]. As this can be used directly, we give no further discussion of the construction here.

Define $Q_k(t)$ to be the number of class k customers present at time t and $Q(t)$ be a vector with kth entry $Q_k(t)$. The following theorem gives the strong connection between maximizing capacity in the queueing network and the solution to (AP).

Theorem 1 **(i)** *Any capacity less than λ^* may be achieved. More specifically, for an arrival process with rate $\lambda < \lambda^*$, there exists a generalized round robin policy such that the distribution of the queue length process $\{Q(t)\}$ converges to a steady-state distribution φ as $t \to \infty$.*

(ii) *A capacity larger than λ^* cannot be achieved. More specifically, for an arrival process with rate $\lambda > \lambda^*$, as $t \to \infty$,*

$$P(|Q(t)| \to \infty) = 1.$$

Proof This is a trivial extension of Theorem 1 in [4]. We simply need

1. a generalized round robin policy can be constructed that achieves allocations arbitrarily close to $\{\delta^*_{j,k}\}$
2. the constraint (4) in the allocation LP is equivalent to the required constraint on the stochastic network

The first of these follows from Proposition 3 of [4]. The second follows from the observation that $\delta^*_{j,k} = 0$ if and only if $\ell^\pi_{j,k} = 0$. □

Theorem 1 says that the difficult stochastic optimization problem can be converted into a deterministic optimization problem. The mapping of the solution to the deterministic problem back to a solution to the original stochastic problem does not depend on the complexity of the deterministic problem (it simply uses the resulting solution). For the remainder of the paper, we thus focus on solving (AP). In [4], the deterministic problem is simply (AP), with the constraint (4) removed. This is easily seen to be a linear programming problem, and so there is the appealing result that a difficult stochastic problem becomes a simple deterministic problem. However, in our case, the resulting deterministic problem can also be difficult, as will be seen below. From this point, we refer to the required deterministic optimization problem as the TOTAL DISCRETE CAPACITY CONSTRAINED PROBLEM (TDCCP).

4 Solving TDCCP - special cases

It is instructive to first look at several special cases of TDCCP that give an idea of the inherent complexity.

4.1 The case $\mu_{j,k} = \mu_k$ for all j

Suppose that the service rates are independent of the server and that each server is capable of working at every class, so $\mu_{j,k} = \mu_k$ for $j = 1, \ldots, M$. Here, (AP) can be rewritten as

$$\max \lambda$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{j=1}^{M} \delta_{j,k} \geq \lambda a_k/\mu_k,\ k = 1, \ldots, K, && (5) \\ & \sum_{k=1}^{K} \delta_{j,k} \leq 1,\ j = 1, \ldots, M, && (6) \\ & \delta_{j,k} \geq 0,\ k = 1, \ldots, K, j = 1, \ldots, M, \\ & \sum_{j=1}^{M} \mathbf{1}\{\delta_{j,k} > 0\} \leq c_k,\ k = 1, \ldots, K, \end{aligned}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function.

Proposition 1 *If $\mu_{j,k} \equiv \mu_k$, the maximal capacity is*

$$\lambda^* = \min\left(\frac{M}{\sum_{k=1}^{K} a_k/\mu_k}, \min_{1\leq k\leq K} \frac{c_k \mu_k}{a_k}\right).$$

Proof Suppose the solution to the allocation program is such that $\sum_{j=1}^{M} \mathbf{1}\{\delta^*_{j,k} > 0\} < c_k$ for all classes k. Then (5) and (6) are necessarily tight for all $k = 1, \ldots, K$ and $j = 1, \ldots, M$, respectively (or else λ^* could be trivially increased). As a result,

$$\sum_{k=1}^{K} \sum_{j=1}^{M} \delta^*_{j,k} = \lambda^* \sum_{k=1}^{K} \frac{a_k}{\mu_k}.$$

But $\sum_{j=1}^{M} \sum_{k=1}^{K} \delta^*_{j,k} = M$, so

$$\lambda^* = \frac{M}{\sum_{k=1}^{K} a_k/\mu_k}.$$

Now, suppose that for some ℓ, $\sum_{j=1}^{M} \mathbf{1}\{\delta^*_{j,\ell} > 0\} = c_\ell$. Then, if we let $k_0 = \arg\min_{1\leq k\leq K} c_k\mu_k/a_k$, either $\sum_{j=1}^{M} \mathbf{1}\{\delta^*_{j,k_0} > 0\} = c_{k_0}$, or else we can transform the solution so that $\sum_{j=1}^{M} \mathbf{1}\{\delta^*_{j,k_0} > 0\} = c_{k_0}$ without decreasing λ^*, by increasing δ^*_{j,k_0} and decreasing $\delta^*_{j,k}$ for at least one j. The resulting value of λ^* is here $\min_{1\leq k\leq K} c_k\mu_k/a_k$.

Combining the two cases yields the result. □

Note that in this case (the servers are identical), the solution is very simple. To see how this relates to the literature on limited flexibility, consider the case when $a_k/\mu_k = 1/\mu$ for $k = 1, \ldots, K$, and $K = M$. Then,

$$\lambda^* = \min\left(\mu, \min_{1\leq k\leq K} c_k\mu\right),$$

so we see that $c_k = 1$ gives the maximal capacity that is equal to that of a system with $c_k = M$ (full flexibility). In general, if the system is roughly balanced (i.e. a_k/μ_k are all close), then we would expect $c_k = 2$ would be sufficient to achieve the same performance as full flexibility. The chaining structure in [14, 16, 24, 18] satisfies this. There are other structures that also achieve this, but the chaining structure is the one that in addition to yielding $c_k = 2$, minimizes the difference between the number of classes to which servers are assigned.

This result suggests something more general in the sense that as long as we build in flexibility such that

$$\frac{c_k \mu_k}{a_k} > \frac{M}{\sum_{k=1}^K a_k/\mu_k}$$

for all classes k, then we should have near optimal performance. The exact structure of the assignment of servers to classes would be revealed by the resulting set $\{\delta^*_{j,k}\}$.

In the flexibility literature, the case when servers are heterogeneous has not been studied (as far as we know of). This may be partially explained by the fact that solving for the maximal capacity under limited flexibility is a much more difficult problem, as we shall see in the next section.

4.2 The case $\mu_{j,k} = \mu_j$ for all k

Suppose now that the service rates depend only on the server and that each server is capable of working at every class, so $\mu_{j,k} = \mu_j$ for $k = 1, \dots, K$. In this case (AP) can be written as

$$\begin{array}{rcll} \max & \lambda & \text{s.t.} & \\ \sum_{j=1}^M x_{j,k} & \geq & \lambda a_k, & k = 1, \dots, K \\ \sum_{k=1}^K x_{j,k} & \leq & \mu_j, & j = 1, \dots, M \\ x_{j,k} & \geq & 0, & k = 1, \dots, K, j = 1, \dots, M \\ \sum_{j=1}^M \mathbf{1}\{x_{j,k} > 0\} & \leq & c_k, & k = 1, \dots, K \end{array} \tag{AP$'$}$$

where we performed the substitution $x_{j,k} := \mu_j \delta_{j,k}, \ \forall j, k$. This case is already NP-complete, as is shown in Theorem 2 in Section 6.

(AP$'$) actually is an instance of the *maximum concurrent multicommodity k-splittable flow problem* which can be stated as follows: Let $G = (V, E)$ be a directed or undirected graph with integral edge capacities $u_e > 0$, for all $e \in E$. There are l source-sink pairs $(s_i, t_i), \ i = 1, \dots, l$, one for each of l different commodities. For each commodity i there is also a demand d_i, and a bound k_i on the number of different paths allowed for this commodity. Then the maximum concurrent multicommodity k-splittable flow problem is asking for a flow assignment to paths in G that respects the edge capacities and the splittability bounds for the commodities, and routes the maximum possible fraction of all commodity demands *simultaneously*. This, together with several other versions of k-splittable problems, are studied in [6]. Also, when $k_i = 1, \ \forall i$, then these problems are simply called *unsplittable* (instead of 1-splittable).

Problem (AP$'$) is an instance of the multicommodity k-splittable flow problem: the K classes can be seen as K commodities of demand $a_k, \ k = 1, \dots, K$, each with a splittability upper bound of $0 < c_k \leq M$. These commodities are routed on the network of Figure 1. All commodities have the same source s, but commodity i has its own sink t_i. Each of the vertices $t_i, \ i = 1, \dots, K$ is connected to all vertices $u_j, \ j = 1, \dots, M$, and s is connected to all vertices $v_j, \ j = 1, \dots, M$. The edge (v_j, u_j) has capacity μ_j for all $j = 1, \dots, M$, while the rest of the edges have infinite capacity. Note that a solution to the maximum concurrent multicommodity k-splittable flow problem on this instance will also give a solution to our original problem (AP$'$), since every flow path that carries flow f of commodity k through edge (v_j, u_j) corresponds to setting $\delta_{j,k} := f$. And vice versa, a solution to (AP$'$) gives us also a path flow assignment that

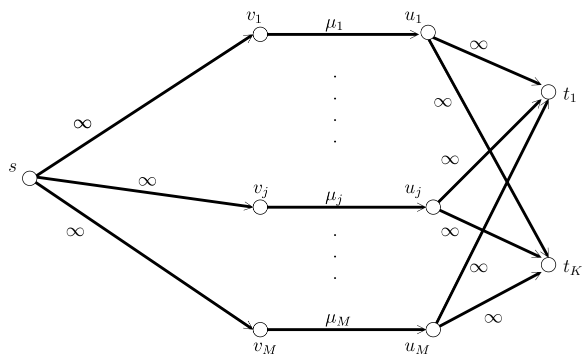

Figure 1: The graph for our special k-splittable flow instance.

achieves the same value for the minimum fraction of commodity demand that is satisfied in the maximum concurrent multicommodity k-splittable flow problem above.

Baier et al. [6] show that any ρ-approximation algorithm[2] for the maximum concurrent unsplittable flow problem yields a $\rho/2$-approximation algorithm for the maximum concurrent k-splittable flow problem. Dinitz et al. [10] present an algorithm that achieves an approximation factor of $\rho = 1/5$ in running time $O(KM(K+M))$, using ideas by Kolliopoulos and Stein [19]. Therefore the solution we get for our problem has a guaranteed *worst-case* performance of at least 1/10 of the optimum. Note that in our case, the usual *balancing* assumption

$$\text{max demand} \leq \text{min capacity}$$

does not necessarily hold (cf. Stage 2 of the following algorithm description), hence the somewhat worse approximation ratios achieved, as compared to the ratios achieved if this assumption holds. We will denote this first algorithm as Algorithm 1. It is the combination of the algorithm of [6] with the algorithm of [10], and goes through the following stages:

Stage 1: Transformation of initial problem to a maximum concurrent uniform exactly-k-splittable flow problem. Replicate each terminal t_k into c_k identical terminals $t_{k,i}, i = 1, \ldots, c_k$. Every commodity k with demand d_k is split into c_k commodities $(k,i), i = 1, \ldots, c_k$, each with the same demand $d_{k,i} := d_k/c_k$ and with the same source s and a different sink $t_{k,i}$ (the values of the $d_{k,i}$'s are determined in the next stage). Each one of the new commodities (k,i) will have to be routed unsplittably.

Stage 2: Calculation of demands $d_{k,i}$. We perform a binary search on the value λ of the fraction of the initial demands a_k that can be routed. Initially $\lambda_{min} := 0, \lambda_{max} := M \cdot \max_{j,k}\{\mu_j c_k/a_k\}$. Note that the initial value of λ_{max} is an upper bound on the maximum possible fraction of demands that can be routed *simultaneously* on the given network. Do the following until $\lambda_{max} < (1+\epsilon)\lambda_{min}$ for some (given) accuracy parameter $\epsilon > 0$:

1. $\lambda := (\lambda_{min} + \lambda_{max})/2$.
2. $d_{k,i} := \lambda a_k/c_k, \ \forall k, i$.

[2]An *α-approximation algorithm* is an algorithm producing solutions within a factor α of the optimum.

3. We formulate a standard LP for the *fractional* routing of commodities (k, i) in the network built in Stage 1. Let $x_{j,k,i}$ be a variable that denotes the amount of commodity (k, i) that flows through edge (v_j, u_j). If $\mu_j < d_{k,i}$ then we add to the LP the constraint $x_{j,k,i} = 0$, and we do this for all j, k, i.

4. If the LP is infeasible then set $\lambda_{max} := \lambda$ and go to Step 1, else set $\lambda_{min} := \lambda$ and go to Step 1.

At the end of this stage we have calculated a *fractional* flow whose λ comes within $1 - \epsilon$ of the best possible fraction of demands that can be routed on the given network satisfying the condition $x_{j,k,i} = 0$ if $\mu_j < d_{k,i}$. The latter optimum is an upper bound to the optimum fraction in the *unsplittable* case. Let λ^* be the λ value we have calculated, and $x_{j,k,i}$ the flow of commodity (k, i) that passes through edge (v_j, u_j) for all j, k, i that achieves this λ^*. Then the demand for commodity (k, i) that is satisfied is $d^*_{k,i} = \lambda^* a_k / c_k$. Since it may be the case that $\max_k\{\frac{\lambda^* a_k}{c_k}\} > \min_j\{\mu_j\}$, the balancing assumption may not hold.

Stage 3: Transformation of the fractional flow calculated in Stage 2 into an unsplittable flow. Let $d_{min} := \min_{k,i}\{d^*_{k,i}\}, d_{max} := \max_{k,i}\{d^*_{k,i}\}$. We consider the following sequence of intervals:

$$[d_{min}, 2d_{min}), [2d_{min}, 2^2 d_{min}), \ldots, [2^l d_{min}, 2^{l+1} d_{min}), \ldots, [2^{\nu-1} d_{min}, 2^{\nu} d_{min}]$$

where $\nu := \lceil \log \frac{d_{max}}{d_{min}} \rceil$. Let $l := 0$ initially. Do the following:

1. Find the commodities (k, i) whose demand $d^*_{k,i}$ falls within the interval $[2^l d_{min}, 2^{l+1} d_{min})$. We assume that there are m such commodities, and for clarity we denote them as commodities $1, 2, \ldots, m$ and their corresponding sinks as $t_1, t_2, \ldots, t_m$. Assign these commodities unsplittably to edges (v_j, u_j) as follows:

   (a) Let x_{j,t_l} be the flow variable that denotes the flow of commodity l that goes through edge (v_j, u_j). Remove all edges with 0 flow going through them.

   (b) If there is any t_l with only one incoming edge (u_j, t_l) then assign commodity l to edge (v_j, u_j) and remove t_l and (u_j, t_l) from the network. Also set $x_{j,t_l} := 0$ and remove commodity l from the network by adjusting the flow variables of the network appropriately. After this step is done, each of the remaining t_l's has at least two incoming edges.

   (c) Starting from a node u_{j_1}, start going alternatively forward and backward on edges of the network between nodes u_j and terminals t_l until we either find a cycle of the form

   $$u_{j_1} \to t_{l_1} \to u_{j_2} \to t_{l_2} \to \cdots \to u_{j_1}$$

   or a path of the form

   $$u_{j_1} \to t_{l_1} \to u_{j_2} \to t_{l_2} \to \cdots \to u_{j_p}$$

   and there is no outgoing edge from u_{j_p} that has not been traversed yet. In the second case, complete a cycle by adding $s \to v_{j_1} \to u_{j_1}$ going forward on edges, and $u_{j_p} \to v_{j_p} \to s$ going backwards on edges. We emphasize that every edge in this cycle or path should be traversed only once. Let e be the minimum amount of total flow through any of the forward traversed edges of the cycle. By adjusting the edge flow variables appropriately, we reduce the flow on all forward going paths by e and increase the flow in backward edges by e. This will eliminate the flow from at least one edge (the edge where the flow is exactly e). Go to Step (a).

2. We scale down the calculated unsplittable flow by a factor of $\max_j \frac{\sum_{(k,i)} x_{j,k,i}}{\mu_j}$ if this quantity is greater than 1. This is our solution, and the λ that corresponds to this solution is the throughput we achieved.

4.2.1 A different approximation algorithm (Algorithm 2)

The previous algorithm cannot take advantage of the better approximation factor of 2 for the unsplittable flow problem, because the balancing assumption does not hold in our case. Here we follow a different path, in order to provide an approximation algorithm that under certain assumptions achieves a factor better than 1/10 for the case $\mu_{j,k} = \mu_j$ for all k. We will reduce our problem to the generalized assignment problem, and then we will use the approximation algorithm by Shmoys and Tardos [25].

The first step of the new algorithm is the same as before: we transform the given problem into an exactly-k-splittable flow problem, with a loss of a factor of 1/2. Hence commodity k is split into c_k commodities (k,i), $i = 1, \ldots, c_k$, each with demand a_k/c_k.

During the second step, we solve the following concurrent flow problem in the network defined above, which in turn is a relaxation of the concurrent unsplittable flow:

$$\begin{array}{rcll} \max & \lambda & \text{s.t.} & \\ \sum_{j=1}^{M} x_{j,(k,i)} & \geq & \lambda \frac{a_k}{c_k}, & \forall k,i \\ \sum_{(k,i)} x_{j,(k,i)} & \leq & \mu_j, & \forall j \\ x_{j,(k,i)} & \geq & 0, & \forall i,j,k \end{array} \tag{LP-NEW}$$

If x^*, λ^* is the optimal solution for (LP-NEW), then define $\lambda_{(k,i)} := (\sum_{j=1}^{M} x^*_{j,(k,i)})/(a_k/c_k)$. Obviously $\lambda_{(k,i)} \geq \lambda^* > 0$, $\forall (k,i)$. Also, we define $y_{j,(k,i)} := \frac{c_k}{\lambda_{(k,i)} a_k} x^*_{j,(k,i)}$ and $p_{j,(k,i)} := \frac{a_k}{c_k \mu_j}, \forall i,j,k$. Then y satisfies the following system of inequalities:

$$\begin{array}{rcll} \sum_{j=1}^{M} y_{j,(k,i)} & = & 1, & \forall k,i \\ \sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} & \leq & 1/\lambda^*, & \forall j \\ y_{j,(k,i)} & \geq & 0, & \forall i,j,k \end{array}$$

This is exactly the relaxation of the problem (without costs) of scheduling unrelated parallel machines that [25] studies. We can think of the commodities (k,i) as jobs, the edges of capacities μ_j as machines, $p_{j,(k,i)}$ as the processing time of job (k,i) on machine j, $1/\lambda^*$ as the makespan, and y as a feasible (fractional) assignment of jobs to machines that achieves this makespan. Suppose that there is some $\rho > 0$ such that $p_{j,(k,i)} \leq \rho/\lambda^*, \forall i,j,k$. Then Theorem 2.1 of [25] implies that their algorithm produces an (integral) assignment of jobs to machines $\hat{y}$ with makespan at most $(1+\rho)/\lambda^*$. This is the third step of our algorithm.

Our solution assigns $\hat{x}_{j,(k,i)} := \frac{\lambda_{(k,i)} a_k}{c_k} \hat{y}_{j,(k,i)}$ (note that for every (k,i), these values are going to be 0 for all j except one.) It is easy to prove the following:

Lemma 1 *The solution produced by Algorithm 2 is within* $1/2(1+\rho)$ *of the optimum.*

Proof The solution $\hat{x}$ satisfies the constraints of (LP-NEW) for $\lambda := \lambda^*/(1+\rho)$. Hence it approximates the maximum concurrent unsplittable flow within a factor of $1/(1+\rho)$. Together with the approximation factor of $1/2$ from the first step, this implies the lemma. □

If the assumption $\rho < 4$ holds, the approximation guarantee of this algorithm is better than the guarantee of $1/10$ achieved by the algorithm in the previous section.

4.3 The case $\mu = \alpha \cdot \beta^T$

The results of the previous section can be generalized to any $M \times K$ matrix μ which is the product of an $M \times 1$ vector α and the transpose of a $K \times 1$ vector β, i.e., $\mu = \alpha \cdot \beta^T$ (in other words, the service rates satisfy $\mu_{j,k} = \alpha_j \beta_k$). Then it is easy to see that the initial problem (AP) is equivalent to

$$\begin{array}{rcll} \max & \lambda & \text{s.t.} & \\ \sum_{j=1}^{M} x_{j,k} & \geq & \lambda b_k, & k = 1, \ldots, K \\ \sum_{k=1}^{K} x_{j,k} & \leq & \alpha_j, & j = 1, \ldots, M \\ x_{j,k} & \geq & 0, & k = 1, \ldots, K, j = 1, \ldots, M \\ \sum_{j=1}^{M} \mathbf{1}\{x_{j,k} > 0\} & \leq & c_k, & k = 1, \ldots, K \end{array} \tag{AP$''$}$$

where $x_{j,k} := \alpha_j \delta_{j,k}$, for all j, k, and $b_k := a_k/\beta_k$. (AP$''$) then falls into the case of Section 4.2.

4.4 Extension to TDCCP with costs

We can extend TDCCP by introducing *costs* to the assignment of servers to classes. Let $c_{j,k}$ be the per unit cost of assigning server j to class k. Hence, if $\delta_{j,k}$ is the fraction of its effort dedicated by server j to class k, then the cost incurred is $c_{j,k}\delta_{j,k}$. For example, the assignment of a worker to a machine where he has no expertise may incur a bigger cost (because of training needs, damages because of deficient products he produces etc.) than the cost of an experienced worker to the same machine. Together with these costs $c_{j,k}$, we are also given a *budget* that cannot be exceeded by our final assignment. Hence we are asked for an assignment of servers to classes that respects the given budget and maximizes the capacity.

The algorithms of [6] and [25] used in Section 4.2.1 are *cost preserving*, in the sense that the cost of the integral solution produced by rounding a fractional solution is not bigger than the cost of that fractional solution. When in the first step we transform the budget-constrained k-splittable flow problem into a budget-constrained exactly-k-splittable flow problem, [6] proves that the optimal solution of the latter is not only an $1/2$ approximation of the former, but it also respects the initial budget constraint. Also the algorithm of [25] we use in Section 4.2.1 produces an assignment that always respects the budgetary constraint (although it may not produce the optimal makespan).[3]

[3]Obviously the costs in the budgetary constraint in each of the LP formulations above are scaled following the scaling of the assignment variables.

5 Solving TDCCP - general case

For the general case, let

$$w_j := \frac{\mu_j^{max}}{\mu_j^{min}},\ j = 1, 2, \ldots, M$$

where $\mu_j^{max} := \max_k\{\mu_{j,k}\}$ and $\mu_j^{min} := \min_k\{\mu_{j,k}\}$. Note that $\mu_{j,k} = 0$ implies that $\delta_{j,k} = 0$, so without loss of generality, we will assume that $\mu_{j,k} > 0$ for all j, k. Also, let

$$w_{max} := \max_j\{w_j\}$$

and let δ^*, λ^* be the optimal solution to (AP). Instead of the original problem (AP), we will try to solve (approximately) the following problem:

$$\begin{array}{rcll} \max & \lambda & \text{s.t.} & \\ \sum_{j=1}^M \mu_{j,k}\delta_{j,k} & \geq & \lambda a_k, & k = 1, \ldots, K \\ \sum_{k=1}^K \mu_{j,k}\delta_{j,k} & \leq & \mu_j^{max}, & j = 1, \ldots, M \\ \delta_{j,k} & \geq & 0, & k = 1, \ldots, K, j = 1, \ldots, M \\ \sum_{j=1}^M \mathbf{1}\{\delta_{j,k} > 0\} & \leq & c_k, & k = 1, \ldots, K. \end{array} \tag{NEW AP}$$

It is clear that, as in Section 4.2, we can set $x_{j,k} := \mu_{j,k}\delta_{j,k}$ in (NEW AP) to get exactly the same formulation as (AP′). Hence we can apply the same techniques we applied in Section 4.2, to obtain an approximate solution $\hat{x}, \hat{\lambda}$, which is within 1/10 of the optimum solution (of (AP′)). Then we output the following solution to the original problem:

$$\delta_{j,k} := \frac{\hat{x}_{j,k}}{w_j\mu_{j,k}},\ \forall j, k. \tag{7}$$

Proposition 2 *Solution (7) is a feasible solution for (AP), and achieves a λ of value at least $\lambda^*/10w_{max}$.*

Proof By construction, we know that (7) satisfies the characteristic function constraints, since $\delta_{j,k}$ is non-zero iff $\hat{x}_{j,k}$ is non-zero. Also, note that

$$\sum_{k=1}^K \delta_{j,k} = \sum_{k=1}^K \frac{1}{w_j\mu_{j,k}}\hat{x}_{j,k} \leq \frac{1}{\mu_j^{max}}\sum_{k=1}^K \hat{x}_{j,k} \leq 1$$

where the first of the inequalities is due to the following fact:

$$\mu_j^{max} = w_j\mu_j^{min} \leq w_j\mu_{j,k},\ \forall k$$

and the second inequality is due to the fact that $\hat{x}$ is feasible for (NEW AP).

Also, note that

$$\sum_{j=1}^M \mu_{j,k}\delta_{j,k} = \sum_{j=1}^M \frac{\hat{x}_{j,k}}{w_j} \geq \frac{\sum_{j=1}^M \hat{x}_{j,k}}{w_{max}} \geq \frac{\hat{\lambda}}{w_{max}}.$$

Hence we have found a solution with a λ value at least $1/4w_{max}$ times the optimum.

□

Obviously, this result extends to the case of a budgetary constraint problem, i.e. the approximation factor can be achieved without violating the (given) budget (cf. Section 4.4).

One other possibility is that if

$$\frac{\mu_{j,k}}{\mu_{j',k}} \leq (1+\varepsilon)$$

for some $\varepsilon > 0$, for all j, j', k, then one can use the result of Section 4.1 to get an approximation. (Note that if $\mu_{j,k} = 0$ for some j, k, then there is no finite ε such that the above bound holds.) Suppose the system has maximal capacity λ^* (as seen above, this requires solving an NP-complete problem). If instead we were to find the maximal capacity of a system with $\mu_{j,k}$ replaced by $\min_{j'} \mu_{j',k}$ and compute the resulting maximal capacity $\tilde{\lambda}$ using Proposition 1, it is easy to see that

$$\tilde{\lambda} \leq \lambda^* \leq \tilde{\lambda}(1+\varepsilon).$$

In other words, this may be a useful approach if there is "near homogoneity" of servers.

6 NP-completeness

We reduce a slight variation of the classical PARTITION problem (see [SP12] in [12]) to the version of our problem that is studied in Section 4.2, which, by abusing the terminology a little bit, we will call problem (AP′):

AP′
Instance: We are given (AP′) and $\lambda^* \in \mathbb{R}$.
Question: Is the solution of (AP′) greater than or equal to λ^*?

Obviously this problem is in NP. The PARTITION problem variation we reduce it to is the following:

PARTITION
Instance: Finite set A of even cardinality and a size $s(a) \in \mathbb{Z}^+$ for each item $a \in A$.
Question: Is there a subset $A' \subseteq A$ of cardinality $|A|/2$ and such that $\sum_{a\in A'} s(a) = \sum_{a\in A\setminus A'} s(a)$?

Given the PARTITION instance, we identify the elements of A with the numbers $1, 2, \ldots, |A|$. Let $S := \sum_{j=1}^{|A|} s(j)$ be the total size. We set $K := 2, M := |A|$ and $\mu_j := s(j),\ j = 1, \ldots, |A|$. We also set $c_1 = c_2 := |A|/2$, and $a_1 = a_2 := 1$. Finally we set $\lambda^* := S/2$. Therefore we get an instance of (AP′) in polynomial time. From now on, when we refer to (AP′), we actually refer to this specific instance we constructed. We prove the following

Theorem 2 PARTITION *has a solution iff (AP′) achieves* $\lambda \geq \lambda^*$.

Proof Before we proceed to the proof of the theorem, we will make some observations about the nature of the solution to (AP′).

Claim 1 *Every solution of (AP′) can be transformed into a solution that achieves the same* λ *and, in addition, has the following properties:*

1. *For all servers j, except possibly one, $x_{j,1}x_{j,2} = 0$, i.e., $x_{j,1}, x_{j,2}$ cannot be both non-zero, and if one is 0, the other is either 0 or μ_j.*

2. *If there is a server j with $x_{j,1}x_{j,2} > 0$, then there is exactly one server $l \neq j$ with $x_{l,1} = x_{l,2} = 0$.*

Proof (of claim) First note that for any server j such that $x_{j,1}, x_{j,2}$ are not both 0, say, $x_{j,1} > 0$, we can increase $x_{j,1}$ without losing feasibility, until the constraint

$$\sum_{k=1}^{2} x_{j,k} \leq \mu_j$$

becomes tight. Therefore we can assume from now on that these constraints are tight for any server j that is actually used, i.e., $x_{j,1} > 0$ or $x_{j,2} > 0$.

Suppose that there are two servers j, l such that $x_{j,1}x_{j,2} > 0$ and $x_{l,1}x_{l,2} > 0$. Let $\epsilon := \min\{x_{j,1}, x_{l,2}\}$. Set

$$\begin{aligned} x_{j,1} &:= x_{j,1} - \epsilon \\ x_{j,2} &:= x_{j,2} + \epsilon \\ x_{l,1} &:= x_{l,1} + \epsilon \\ x_{l,2} &:= x_{l,2} - \epsilon. \end{aligned}$$

Note that this transformation does not violate feasibility or affect λ. But as a result, the minimum of $x_{j,1}, x_{l,2}$ becomes 0. We continue performing this transformation until there is at most one server j with $x_{j,1}x_{j,2} > 0$, and part 1 of the claim is proven. If there is one such server, then because of part 1 and the constraints

$$\sum_{j=1}^{|A|} \mathbf{1}\{x_{j,k} > 0\} \leq |A|/2, \;\; k = 1, 2$$

there are at most $|A| - 1$ servers that are used, therefore there is at least one server l with $x_{l,1} = x_{l,2} = 0$. In fact, there will be exactly one such server, since if there were more, the constraints above would not be tight, and we could have used one more (so far unused) server, to increase λ, contradicting the optimality of the solution we got in part 1. □

It is easy to see that without the constraints

$$\sum_{j=1}^{|A|} \mathbf{1}\{x_{j,k} > 0\} \leq |A|/2, \;\; k = 1, 2,$$

(AP′) can achieve at most $\lambda = S/2$. Therefore, in order to prove Theorem 2 all we need to show is that PARTITION has a solution iff (AP′) achieves $\lambda = S/2$. One direction is trivial: if PARTITION has a solution, then set $x_{j,1} := \mu_j, x_{j,2} := 0, \; \forall j \in A'$ and $x_{j,1} := 0, x_{j,2} := \mu_j, \; \forall j \in A \setminus A'$; this achieves the maximum possible $\lambda = S/2$.

For the opposite direction, we first transform the solution to (AP′) that achieves $\lambda = S/2$, so that it complies with Claim 1. If there is no server j such that $x_{j,1}x_{j,2} > 0$, then all servers are used (due to optimality of λ), and for each server exactly one of $x_{j,1}, x_{j,2}$ is equal to μ_j and the other is equal to 0. Hence in this case the solution of (AP$'$) corresponds exactly to a solution of PARTITION. Now suppose that there is a server j such that $x_{j,1}x_{j,2} > 0$. Part 2 of Claim 1 then implies that there is one server l that is not used, i.e., $x_{l,1} = x_{l,2} = 0$. But then $\sum_{k=1}^{2}\sum_{j=1}^{|A|} x_{j,k} \leq S - s(l)$, which contradicts the fact that $\sum_{j=1}^{|A|} x_{j,1} \geq S/2$ and $\sum_{j=1}^{|A|} x_{j,2} \geq S/2$.

□

7 Examples

Here we present three examples to give an idea of the algorithms' performance and also to provide some discussion on the difficulty of finding flexibility structures for heterogeneous servers. It should be stressed that the approximation bounds derived for our algorithms above are *worst-case* guarantees. Therefore, it is conceivable that these algorithms may perform much better in practice. The following examples suggest that this may be the case but are of course not conclusive. This is the case in the examples studied below.

Example 1. Let $M = 4$, $K = 10$ classes in tandem, and consider $\mu_{j,k} = \mu_j$, with $\mu_1 = 0.5$, $\mu_2 = \mu_3 = 1$, $\mu_4 = 2$. Here, with full flexibility ($c_k = 4$), the maximal capacity is 9/20. For more limited flexibility, we get the results in Table 1 for the appropriate maximal capacities. Note that Algorithm 1 performs reasonably well (this example is small enough that the optimal

Table 1: Maximal capacities for Example 1

	Algorithm 1	Algorithm 2	Optimal
$c_k = 1$	1/3	1/4	2/5
$c_k = 2$	1/3	1/4	9/20
$c_k = 3$	3/7	3/7	9/20

solution can be produced by hand). Also, we see that one only needs $c_k = 2$ to capture the benefits of full flexibility.

Example 2. Here we use the same configuration as the previous example, but set $\mu_1 = 0.1$, $\mu_2 = \mu_3 = 1$, $\mu_4 = 10$. With full flexibility, $\lambda^* = 1.21$. The results for more limited flexibility are in Table 2. Once again, Algorithm 1 performs reasonably. Algorithm 2 has the undesirable property that the estimate of the value of the maximal capacity has decreased in this instance when moving from $c_k = 1$ to $c_k = 2$ (there is a trivial remedy by simply taking the value for $c_k = 1$ for both cases).

An interesting observation (and perhaps one that gives some intuition as to why these problems are so difficult in general) is that for $c_k = 1$, server 1 is not utilized at all in the optimal solution, while for $c_k = 2$, there is some benefit to using this very slow server. This tradeoff between effectively using faster servers versus idling slower ones is in essence what

Table 2: Maximal capacities for Example 2

	Algorithm 1	Algorithm 2	Optimal
$c_k = 1$	1	1	1
$c_k = 2$	1	2/3	1.21
$c_k = 3$	1	3/4	1.21

makes the problem so difficult (this issue also arises in Rubinovitch [22]). With full flexibility or homogeneous servers, this tradeoff disappears.

Example 3. Here we consider an instance of the general case. The topology is taken from System 2 of [4]. Here, $\mu_{j,k}$ is given by the (j, k) entry in the following matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

In addition, the routing in the network is such that $a_k = 1$ for $1 \leq k \leq 5$ and $a_k = 0.5$ for $6 \leq k \leq 10$. Here, if we first choose $c_k = 2$, $1 \leq k \leq 10$, we get maximal capacity estimates of 0.5 for Algorithms 1 and 2, while the optimal value is 0.9143 (the optimal value for full flexibility is 0.9474). Once again, the performance of Algorithm 1 is well above its guarantee, but is relatively worse than for the previous examples. If we further reduce the flexibility by setting $c_1 = c_{10} = 1$ and $c_k = 2$ for $2 \leq k \leq 9$, the results are unchanged. This demonstrates that while we are employing a state of the art algorithm, it still may be problematic to employ for design decisions. (That is, if one needs absolute values. The issue of using it to make relative choices between design options is something that has not been explored and could be a useful topic for future work.)

8 Concluding remarks

Except for the special cases considered in this work, we do not know the approximability of the general allocation problem (AP). In the special case where the servers are homogeneous, we have seen that the problem has a simple solution, and as a result it is relatively straightforward to study the impact of limited flexibility. When the servers are heterogeneous, we see that finding the maximal capacity is difficult under limited flexibility (as opposed to under full flexibility, in which case it is straightforward), which leads one to believe that the problem of how to exploit limited flexibility in this case is one that may be difficult to look at in general. The fact that computing the maximal capacity itself is difficult also suggests that finding general, structural results for heterogeneous servers may be very difficult. However, the algorithm presented here gives hope that it may be used to study specific examples and as such build insight into the issue.

Acknowledgments. The authors acknowledge the assistance of Liuxing Kan for his work on the numerical examples.

References

[1] O.Z. Akşin and F. Karaesmen. Designing flexibility: characterizing the value of cross-training practices. Working paper.

[2] S. Andradóttir and H. Ayhan. Throughput maximization for tandem lines with two stations and flexible servers. *Operations Research*, 53:516-531, 2005.

[3] S. Andradóttir, H. Ayhan, and D.G. Down. Server assignment policies for maximizing the steady-state throughput of finite queueing systems. *Management Science*, 47:1421-1439, 2001.

[4] S. Andradóttir, H. Ayhan and D.G. Down. Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, 51:952-968, 2003.

[5] M. Armony and N. Bambos. Queueing networks with interacting service resources. *Proceedings of the 37th Annual Allerton Conference on Communications, Control, and Computing*, 42-51, 1999.

[6] G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. *Proceedings of ESA'02*.

[7] D.P. Bischak. Performance of a manufacturing module with moving workers. *IIE Transactions*, 28:723-733, 1996.

[8] J.G. Dai. On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *Annals of Applied Probability*, 5:49-77, 1995.

[9] J.G. Dai and W. Lin. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53:197-218, 2005.

[10] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19:17-41, 1999.

[11] D.G. Down and G. Karakostas. Maximizing throughput in queueing networks with limited flexibility. *Proceedings of 7th Latin American Symposium on Theoretical INformatics (LATIN),* Springer LNCS 3887, pp. 398–409, 2006.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness.* W. H. Freeman and Co., 1979.

[13] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, Chapter 6. Springer-Verlag, 1993.

[14] S. Gurumurthi and S. Benjaafar. Modeling and analysis of flexible queueing systems. *Naval Research Logistics*, 51:755-782, 2004.

[15] F.S. Hillier and K.C. So. On the simultaneous optimization of server and work allocations in production line systems with variable processing times. *Operations Research*, 44:435-443, 1996.

[16] W.J. Hopp, E. Tekin, and M.P. van Oyen. Benefits of skill chaining in serial production lines with cross-trained workers. *Management Science*, 50:83-98, 2004.

[17] W.J. Hopp and M.P. van Oyen. Agile workforce evaluation: A framework for cross-training and coordination. *IIE Transactions*, 36:919-940, 2004.

[18] W.J. Jordan and S.C. Graves. Principles on the benefits of manufacturing process flexibility. *Management Science*, 41:577-594, 1995.

[19] S. G. Kolliopoulos and C. Stein. Approximation algorithms for single-source unsplittable flow. *SIAM J. Computing* 31:919-946, 2002.

[20] S.P. Meyn and D. Down. Stability of generalized Jackson networks. *Annals of Applied Probability*, 9:124-148, 1994.

[21] J. Ostalaza, J. McClain and J. Thomas. The use of dynamic (state-dependent) assembly-line balancing to improve throughput. *Journal of Manufacturing and Operations Management*, 3:105-133, 1990.

[22] M. Rubinovitch. The slow server problem: a queue with stalling. *Journal of Applied Probability*, 22:877-892, 1985.

[23] A.N. Rybko and A.L. Stolyar. Ergodicity of stochastic processes describing the operation of open queueing networks. *Problems of Information Transmission*, 28:199-220, 1992.

[24] M. Sheikhzadeh, S. Benjaafar, and D. Gupta. Machine sharing in manufacturing systems: Flexibility versus chaining. *International Journal of Flexible Manufacturing Systems*, 10:351-378, 1998.

[25] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461-474, 1993.

[26] K. Sigman. The stability of open queueing networks. *Stochastic Processes and their Applications*, 35:11-25, 1990.

[27] M.S. Squillante, C.H. Xia, D.D. Yao, and L. Zhang. Threshold-based priority policies for parallel-server systems with affinity scheduling. *Proceedings of the 2001 American Control Conference*, 2992-2999, 2001.

[28] L. Tassiulas and P.B. Bhattacharya. Allocation of independent resources for maximal throughput. *Stochastic Models*, 16:27-48, 2000.

[29] L. Tassiulas and A. Ephrimedes. 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.

[30] R.B. Wallace and W. Whitt. A staffing algorithm for call centers with skill-based routing. *Manufacturing and Service Operations Management*, 7:276-294, 2005.

[31] E. Zavadlav, J.O. McClain, and L.J. Thomas. Self-buffering, self-balancing, self-flushing production lines. *Management Science*, 42:1151-1164, 1996.