Scheduling and Traffic Allocation for Tasks with Bounded Splittability

Piotr Krysta   Peter Sanders
Berthold Vöcking

MPI–I–2003–1–002   February 2003

Authors' Addresses

Piotr Krysta
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
krysta@mpi-sb.mpg.de
Partially supported by DFG grant Vo889/1-1, and IST program of the EU under contract IST-1999-14186 (ALCOM-FT).

Peter Sanders
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
sanders@mpi-sb.mpg.de
Partially supported by DFG grant Sa933/1-1, and IST program of the EU under contract IST-1999-14186 (ALCOM-FT).

Berthold Vöcking
Dept. of Computer Science
Universität Dortmund
Baroper Str. 301
44221 Dortmund
voecking@ls2.cs.uni-dortmund.de
Partially supported by DFG grant Vo889/1-1.

Abstract

We investigate variants of the well studied problem of scheduling tasks on uniformly related machines to minimize the makespan. In the k-splittable scheduling problem each task can be broken into at most $k \geq 2$ pieces each of which has to be assigned to a different machine. In the slightly more general SAC problem each task j comes with its own splittability parameter k_j, where we assume $k_j \geq 2$. These problems are known to be NP-hard and, hence, previous research mainly focuses on approximation algorithms.

Our motivation to study these scheduling problems is traffic allocation for server farms based on a variant of the Internet Domain Name Service (DNS) that uses a stochastic splitting of request streams. Optimal solutions for the k-splittable scheduling problem yield optimal solutions for this traffic allocation problem. Approximation ratios, however, do not translate from one problem to the other because of non-linear latency functions. In fact, we can prove that the traffic allocation problem with standard latency functions from Queueing Theory cannot be approximated in polynomial time within any finite factor because of the extreme behavior of these functions.

Because of the inapproximability, we turn our attention to fixed-parameter tractable algorithms. Our main result is a polynomial time algorithm computing an exact solution for the k-splittable scheduling problem as well as the SAC problem for any fixed number of machines. The running time of our algorithm increases exponentially with the number of machines but is only linear in the number of tasks. This result is the first proof that bounded splittability reduces the complexity of scheduling as the unsplittable scheduling is known to be NP-hard already for two machines. Furthermore, since our algorithm solves the scheduling problem exactly, it also solves the traffic allocation problem that motivated our study.

Keywords

Traffic Allocation, Splittability, Scheduling

1 Introduction

A server farm is a collection of servers delivering data to a set of clients. Some large scale server farms are distributed all over the Internet and deliver various types of site content including graphics, streaming media, downloadable files, and HTML on behalf of other content providers who pay for an efficient and reliable delivery of their site data. To satisfy these requirements, one needs an advanced traffic management that takes care for the assignment of traffic streams to individual servers. Such streams can be formed, e.g., by traffic directed to the same page, traffic directed to pages of the same content provider, or by the traffic requested from clients in the same geographical region or domain, or also by combinations of these criteria. The objective is to distribute these streams as evenly as possible over all servers in order to ensure site availability and optimal performance.

For each traffic stream there is a corresponding stream of requests sent from the clients to the server farm. Current implementations of commercial Web server farms use the Internet Domain Name Service (DNS) to direct the requests immediately to the server that is responsible for delivering the data of the corresponding traffic stream. The DNS can answer a query such as "What is `www.uni-dortmund.de`?" with a short list of IP addresses rather than only a single IP address. The original idea behind returning this list is that, in case of failures, clients can redirect their requests to alternative servers. Nowadays, slightly deviating from this original idea, these lists are also used for the purpose of load balancing among replicated servers (cf., e.g., [12]). When clients make a DNS query for a name mapped to a list of addresses, the server responds with the entire list of IP addresses, but rotates the ordering of addresses within each reply. In this way, as clients typically send their HTTP requests to the IP address listed first, DNS rotation distributes the requests more or less evenly among all the replicated servers in the list.

Suppose the request streams are formed by a sufficiently large number of clients such that it is reasonably well described by a Poisson process. Let λ_j denote the rate of stream j, i.e., the expected number of requests in some specified time interval. Under this assumption, rotating a list of ℓ servers corresponds to splitting stream j into ℓ substreams each of which having rate λ_j/ℓ. We propose a slightly more sophisticated stochastic splitting policy that allows for a better load balancing and additionally preserves the Poisson property of the request streams. Suppose, the DNS attaches a vector $p_1^j, \dots, p_\ell^j$ with $\sum_i p_i^j = 1$ to the list of each stream j. In this way, every individual request in stream j can be directed to the ith server on this list with probability p_i^j. This policy breaks Poisson stream j into ℓ Poisson streams of rate $p_1^j\lambda_j, \dots, p_\ell^j\lambda_j$, respectively.

The possibility to split streams into smaller substreams can obviously reduce the maximum latency. It is not obvious, however, whether it is easier or more difficult to find an optimal assignment if one allows that every stream is allowed to be broken into a bounded number of substreams. Observe that the allocation problem above is a variant of machine scheduling in which streams correspond to jobs and servers to machines. In the context of machine scheduling, bounded splittability has been investigated before with the motivation to speed up the execution of parallel programs. Let us begin our study by introducing the relevant background in scheduling.

Scheduling on uniformly related machines. Suppose a set of jobs $[n] = \{1,\dots,n\}$ need to be scheduled on a set of machines $[m] = \{1,\dots,m\}$. Jobs are described by sizes $\lambda_1,\dots,\lambda_n \in \mathbb{Q}_{>0}$, and machines are described by their speeds $s_1,\dots,s_n \in \mathbb{Q}_{>0}$. In the classical, *unsplittable scheduling problem* for uniformly related machines, every job must be assigned to exactly one machine. This mapping can be described by an assignment matrix $(x_{ij})_{i\in[m],j\in[n]}$, where x_{ij} is an indicator variable with $x_{ij} = 1$ if job j is assigned to machine i and 0 otherwise. The objective is to minimize the *makespan* $z = \max_{i\in[m]} \sum_{j\in[n]} \lambda_j x_{ij}/s_i$.

It is well known that this problem is strongly NP-hard. Hochbaum and Shmoys [8, 9] gave the first polynomial time approximation schemes (PTAS) for this problem. If the number of machines is fixed, then the problem is only weakly NP-hard and it admits a fully polynomial time approximation scheme (FPTAS) [10].

A fractional relaxation of the problem leads to splittable scheduling. In the *fully splittable scheduling* problem the variables x_{ij} can take arbitrary real values from $[0,1]$ subject to the constraints $\sum_{i\in[m]} x_{ij} \geq 1$, for every $j \in [n]$. This problem is trivially solvable, e.g., by assigning a piece of each job to each machine whose size is proportional to the speed of the machine.

***k*-splittable machine scheduling and the SAC problem.** In the *k-splittable machine scheduling problem* each job can be broken into at most $k \geq 2$ pieces that might be placed on different machines, that is, at most k of the variables $x_{ij} \in [0,1]$, for every job j, are allowed to have positive values. Recently, Shachnai and Tamir [18] introduced a generalization of this problem, called *scheduling with machine allotment constraints (SAC)*. In this problem, each job j comes with its own splittability parameter $k_j \geq 1$. In our study, we will mostly assume $k_j \geq 2$, for every $j \in [n]$.

Shachnai and Tamir [18] prove that, in contrast to the fully splittable scheduling problem, the k-splittable machine scheduling problem is strongly NP-hard even on identical machines. As a positive result, they can give a PTAS for the SAC problem, whose running time, however, does not render practical as the splittability appears double exponentially in the running time. As a more practically relevant result, they present a very fast $\max_j(1 + 1/k_j)$-approximation algorithm. This result suggests that, in fact, approximation should get easier when the splittability is increased.

We should mention here, that there is a related scheduling problem in which *preemption* is allowed, that is, jobs can be split arbitrarily but pieces of the same job cannot be processed at the same time on different machines. Shachnai and Tamir study also combinations of SAC and scheduling with preemption in which jobs can be broken into a bounded number of pieces and additionally there are bounds on the number of pieces that can be executed at the same time. Further variants of scheduling with different notions of splittability with various motivations from parallel computing and production planning can be found in [18] and [19].

Scheduling for non-linear latency functions. The only difference between the k-splittable scheduling problem and the traffic allocation problem described above is that the latency occurring at servers cannot be assumed to be linear. A typical example for a set of functions that describes the latency at a server of speed s with an incoming Poisson stream at rate λ is

$$f_s(\lambda) = \frac{\lambda}{s(s - \min\{s,\lambda\})} .$$

This family of functions can be derived from the formula for the waiting time on an M/M/1 queueing system. Of course, M/M/1 waiting time is only one out of many examples for latency functions that can be obtained from Queueing Theory. In fact, a typical property of such functions is that the latency goes to infinity when the injection rate approaches the service rate.

Instead of focusing on particular latency functions, let us set up a more general framework in which we can analyze the effects of non-linearity. The *k-splittable traffic allocation problem* is a variant of the k-splittable scheduling problem. Streams are described by rates $\lambda_1, \ldots, \lambda_n$, and servers are described by bandwidths or service rates $s_1, \ldots, s_m$. In this way, traffic streams can be identified with jobs and servers with machines. The latencies occurring at the servers are described by a *family of latency functions* $\mathcal{F} = \{f_s : \mathbb{Q}_{\geq 0} \to \mathbb{Q}_{\geq 0} \cup \{\infty\} \mid s \in \mathbb{Q}_{>0}\}$ where f_s denotes the latency function for a server with service rate s. These functions are assumed to be non-decreasing.

Scheduling under non-linear latency functions has been considered before as well. Alon et al. [1] give a PTAS for the problem of makespan minimization on identical machines with certain well-behaved latency functions. Their results were further generalized and extended into a PTAS for the makespan minimization on uniformly related machines by Epstein and Sgall [5]. In both studies, the latency functions are assumed to fulfill certain analytical properties like convexity and uniform continuity under a logarithmic scale. Unfortunately, the uniform continuity condition excludes the typical functions from Queueing Theory.

1.1 Our results

The main result of this paper is a fixed-parameter tractable algorithm for the k-splittable scheduling and the more general SAC problem with splittability at least two for every job. Our algorithm has polynomial running time for every fixed number of machines. This result is remarkable as unsplittable scheduling is known to be NP-hard already on two machines. In fact, our result is the first proof that bounded splittability reduces the complexity of scheduling.

In more detail, given any upper bound T on the makespan of an optimal assignment, our algorithm computes a feasible assignment with makespan at most T in time $O(n + m^{m+m/(k_0-1)})$ with $k_0 = \min\{k_1, k_2, \ldots, k_n\}$. Furthermore, despite the possibility to split the jobs into pieces of non-rational size, we can prove that the optimal makespan can be represented by a rational number with only a polynomial number of bits. Thus the optimal makespan can be found efficiently by using binary search techniques over the rational numbers. This yields an exact algorithm for SAC on a fixed number of machines with polynomial running time.

In addition, we study the effects due to the non-linearity of latency functions. The algorithm above can be adopted to work efficiently for a wide class of latency functions containing even such extreme functions as M/M/1 waiting time. In particular, we are also able to use this algorithm for latency functions describing an M/G/1 queue waiting time under heterogeneous traffic. On the negative side, we prove that latency functions like M/M/1 do not admit polynomial time approximation algorithms with finite approximation ratio if the number of machines is unbounded. The latter result is an ultimate rationale for our approach to devise efficient algorithms for a fixed number of machines.

2 An exact algorithm for SAC with given makespan

In this section, we present an exact algorithm for SAC with splittability at least two for every job. Our algorithm has polynomial running time for any fixed number of machines. We assume that an upper bound on the optimal makespan is given. This upper bound defines a capacity for each machine. The capacity of machine i is denoted by c_i. The computed schedule has to satisfy $\sum_{j\in[n]} \lambda_j x_{ij} \leq c_i$, for every $i \in [m]$.

A difficult subproblem that has to be solved is to decide into which pieces of which size the jobs should be cut. In principle, the number of possible cuts is unbounded. Our analysis, however, will show that it suffices to consider only those cuts that "saturate" a machine. Let $\pi_{ij} = \lambda_j x_{ij}$ denote the size of the piece of job j that is allocated to machine i. Then machine i is *saturated* by job j if $\pi_{ij} = c_i$.

Our algorithm, called Algorithm 1, schedules the *bulkiest* job j first where the bulkiness of j is defined as $\lambda_j/(k_j-1)$. Using a backtracking approach it tries all ways to cut one piece from job j such that a machine is saturated. The saturated machine is removed from the problem; the splittablity and size of j is reduced accordingly. The remaining problem is solved recursively. Two special cases have to be considered. Let I being the set of remaining machines. If j satisfies $\lambda_j/(k_j-1) \leq \min\{c_i : i \in I\}$ then j and all other remaining jobs can be scheduled using a simple greedy approach known as McNaughton's rule [15]. Since the splittability k_j of a job is decreased whenever a piece is cut off, a remaining piece can eventually become unsplittable. Since this remaining piece will be infinitely bulky, it will be scheduled in the next iteration. In this case, all machines that can accommodate the piece are tried. A formal description of the algorithm can be found in Figure 1.

Theorem 1. *Algorithm 1 finds a feasible solution for SAC with a given possible makespan, provided that the splittability of each job is at least two. It can be implemented to run in time* $O\left(n + m^{m+m/(k_0-1)}\right)$, *where* $k_0 = \min\{k_j : j = 1, 2, \ldots, n\}$.

Proof. All the necessary data structures can be initialized in time $O(m+n)$ if we use a representation of the piece size matrix (π_{ij}) that only stores nonzero entries. There can be at most m recursive calls that saturate a machine and at most $m/(k_0-1)$ recursive calls made for unsplittable pieces that remain after a job j was split $k_j - 1$ times. All in all, the backtrack tree considers no more than $m!m^{m/(k_0-1)}$ possibilities. The selection of the bulkiest job can be implemented to run in time $O(\log m)$ independent of n: Only the m largest jobs can ever be a candidate. Hence it suffices to select these jobs initially using an $O(n)$ time algorithm [2] and keep them in a priority queue data structure. Greedy scheduling using McNaughton's rule takes time $O(n+m)$. Overall, we get an execution time of $O\left(n + m + m!m^{m/(k_0-1)} \log m\right) = O\left(n + m^{m+m/(k_0-1)}\right)$.

```
$I := [m]$                                                    -- machines that are not saturated
$J := [n]$                                                    -- jobs still to be scheduled
if $\sum_{j \in J} \lambda_j > \sum_{i \in I} c_i \vee \neg$solve() then output "no solution possible"
else output nonzero $\pi_{ij}$ values

Function solve() : Boolean
    if $J = \emptyset$ then return true
    find a $j \in J$ that maximizes $\lambda_j/(k_j - 1)$
    if $k_j = 1$ then                                         -- Unsplittable remaining piece
        forall $i \in I$ with $c_i \geq \lambda_j$ do
            $\pi_{ij} := \lambda_j$; $c_i := c_i - \lambda_j$; $J := J \setminus \{j\}$        -- (*)
            if solve() then return true
            undo changes made in line (*)
    else                                                      -- Job $j$ is splittable
        if $\lambda_j/(k_j - 1) \leq \min\{c_i : i \in I\}$ then McNaughton(); return true
        forall $i \in I$ with $c_i < \lambda_j$ do
            $\pi_{ij} := c_i$; $\lambda_j := \lambda_j - c_i$; $k_j := k_j - 1$; $I := I \setminus \{i\}$      -- (**)
            if solve() then return true
            undo changes made in line (**)
    return false

Procedure McNaughton()                                        -- Schedule greedily
    pick any $i \in I$
    foreach $j \in J$ do
        while $c_i \leq \lambda_j$ do $\pi_{ij} := c_i$; $\lambda_j := \lambda_j - c_i$; $I := I \setminus \{i\}$; pick any new $i \in I$
        $\pi_{ij} := \lambda_j$; $c_i := c_i - \lambda_j$
```

Figure 1: Algorithm 1: Find a schedule of n jobs with splittabilities k_j to m machines.

The algorithm also produces only correct schedules. In particular, when $\lambda_j/(k_j - 1) \leq \min\{c_i : i \in I\}$ McNaughton's rule can complete the schedule because no remaining job is large enough to saturate more than $k_j - 1$ of the remaining machines. In particular, solve() maintains the invariant $\sum_{j \in J} \lambda_j \leq \sum_{i \in I} c_i$ and when McNaughton's rule is called, it can complete the schedule:

Lemma 2. *McNaughton's rule computes a correct schedule if $\sum_{j \in J} \lambda_j \leq \sum_{i \in I} c_i$ and $\forall i \in I, j \in J : \lambda_j/(k_j - 1) \leq c_i$.*

Proof. The only thing that can go wrong is that a job j is split more than $k_j - 1$ times, i.e., into at least $k_j + 1$ pieces. But then, it completely fills at least $k_j - 1$ machines with capacity at least $\min_{i \in I} c_i$. But this contradicts the assumption that $\lambda_j/(k_j - 1) \leq c_i$. $\square$

Now we come to the interesting part of the proof. We have to show that the search succeeds

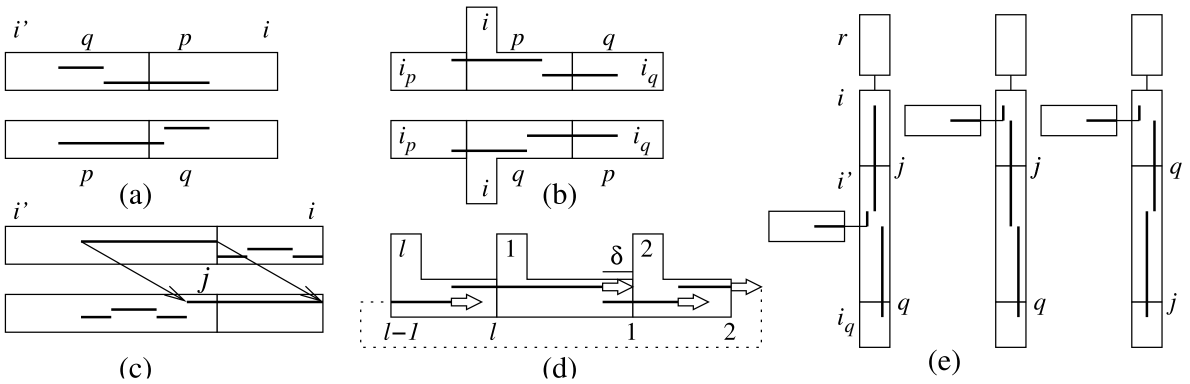

Figure 2: Manipulating schedules. (a): The move from Lemma 4; (b): The swap from Lemma 5; (c): Saturation using Lemma 6; (d): The rotation from Lemma 7; (e): Moving j away from r.

if a feasible schedule exists. We show the stronger claim that the algorithm is correct even if unsplittable jobs are present. (In this case only the above running time analysis would fail.) The proof is by induction on m. For $m = 1$ this is trivial since no splits are necessary.

Now consider the case $m > 1$. If there are unsplittable jobs, they are infinitely bulky and hence are scheduled first. Since all possible placements for them are tried, nothing can be missed for them. When a splittable job is bulkiest, only those splits are considered that saturate one machine. But Lemma 3 below shows that if there is a feasible schedule at all, there must also be one with this property. The recursive call leaves a problem with one machine less and hence the induction hypothesis can be applied. □

Lemma 3. *If a feasible schedule exists and the bulkiest job is large enough to saturate a machine then there is a feasible schedule where the bulkiest job saturates a machine.*

Our approach to proving Lemma 3 is to show that any feasible schedule can be transformed into a feasible schedule where the bulkiest job saturates a machine. To simplify this task, we first establish a toolbox of simpler transformations. We begin with two very simple transformations that affect only two jobs and obviously maintain feasibility. Fig. 2-(a) and 2-(b) illustrate these transformations.

Lemma 4. *For any feasible schedule, consider two jobs p and q sharing machine i', i.e., $\pi_{i'p} > 0$ and $\pi_{i'q} > 0$. For any machine i such that $\pi_{i'q} < \pi_{ip}$ there is a feasible schedule where the overlapping piece of q is moved to machine i, i.e., $(\pi_{i'p}, \pi_{ip}, \pi_{i'q}, \pi_{iq}) := (\pi_{i'p} + \pi_{i'q}, \pi_{ip} - \pi_{i'q}, 0, \pi_{iq} + \pi_{i'q})$.*

Lemma 5. *For any feasible schedule, consider two jobs p and q sharing machine i, i.e., $\pi_{ip} > 0$ and $\pi_{iq} > 0$. Furthermore, consider two other pieces $\pi_{i_p p}$ and $\pi_{i_q q}$ of p and q. If $\pi_{i_p p} \le \pi_{iq} + \pi_{i_q q}$ and $\pi_{i_q q} \le \pi_{ip} + \pi_{i_p p}$ then there is a feasible schedule where the pieces $\pi_{i_p p}$ and $\pi_{i_q q}$ are swapped as follows:*

$$\begin{array}{ccccccc} (& \pi_{i_p p}, & \pi_{ip}, & \pi_{i_q p}, & \pi_{i_p q}, & \pi_{iq}, & \pi_{i_q q} \quad) := \\ (& 0, & \pi_{ip} + \pi_{i_p p} - \pi_{i_q q}, & \pi_{i_q p} + \pi_{i_q q}, & \pi_{i_p q} + \pi_{i_p p}, & \pi_{iq} + \pi_{i_q q} - \pi_{i_p p}, & 0 \quad) \end{array}$$

As a first application of Lemma 4 we now explain how a large job j allocated to at most $k_j - 1$ machines can "take over" a small machine.

Lemma 6. *Consider a job j and machine i such that $\lambda_j/(k_j-1) \geq c_i$. If there is a feasible schedule where j is scheduled to at most (k_j-1) machines, then there is a feasible schedule where j saturates machine i.*

Proof. Let i' denote a machine index that maximizes $\pi_{i'j}$ and note that $\pi_{ij} \geq \lambda_j/(k_j-1) \geq c_i$. We can now apply Lemma 4 to subsequently move all the pieces on machine i to machine i'. Lemma 4 remains applicable because π_{ij} is large enough to saturate machine i. Figure 2-(c) illustrates this transformation. $\square$

After the above local transformations, we come to a global transformation that greatly simplifies the kind of schedules we have to consider.

Definition 1. Job j is called *split* if $\left|\left\{i : \pi_{ij} > 0\right\}\right| > 1$. The *split graph* corresponding to a schedule is an undirected hypergraph $G = ([m], E)$ where each split job j corresponds to a hyperedge $\left\{i : \pi_{ij} > 0\right\} \in E$.

Lemma 7. *If a feasible schedule exists, then there is a feasible schedule whose split graph is a forest.*

Proof. It suffices to show that for a feasible schedule whose split graph G contains a cycle there is also a feasible schedule whose corresponding split graph has a smaller value of $\sum_{e\in E}|e|$. Then it follows that a feasible schedule that minimizes $\sum_{e\in E}|e|$ is a forest.

So suppose G contains a cycle involving ℓ edges. Let $\operatorname{succ}(j)$ stand for $(j+1) \bmod \ell+1$. By appropriately renumbering machines and jobs we can assume without loss of generality that this cycle is made up of jobs 1 to ℓ and machines 1 to ℓ such that for $j \in [\ell]$, $\pi_{jj} > 0$, $\pi_{\operatorname{succ}(j)j} > 0$, and $\delta = \pi_{11} = \min_{j\in[\ell]} \min\left\{\pi_{jj}, \pi_{\operatorname{succ}(j)j}\right\}$. Figure 2-(d) depicts this normalized situation.

Now we *rotate* the pieces in the cycle by increasing π_{jj} by δ and decreasing $\pi_{\operatorname{succ}(j)j}$ by the same amount. The schedule remains feasible since the load of the machines in the cycle remains unchanged. Since the first job is now split in one piece less, $\sum_{e\in E}|e|$ decreases. $\square$

Now we have all the necessary tools to establish Lemma 3.

Proof. Consider any feasible schedule, let j denote the bulkiest job and assume that there is a machine i_0 with $\lambda_j/(k_j-1) \geq c_{i_0}$. We transform this schedule in several steps.

We first apply Lemma 7 to obtain a schedule whose split graph is a forest. We now concentrate on the tree T where j is allocated.

If job j is allocated to at most $k_j - 1$ machines, we can saturate i_0 using Lemma 6 and we are done.

If one piece of j is allocated to a leaf i of T then all other jobs mapped to machine i are allocated there entirely. Let i' denote another machine j is mapped to. We apply Lemma 4 to move small jobs from i to i'. When this is no longer possible, either job j saturates machine i and we are done or there is a job j' with $\lambda_{j'} = \pi_{ij'} > \pi_{i'j}$. Now we can apply Lemma 5 to pieces π_{ij},

$\pi_{i'j}$, $\pi_{ij'}$, and a zero size piece of job j'. This transformation reduces the number of pieces of job j so that we can saturate machine i_0 using Lemma 6.

Finally, j could be allocated to machines that are all interior nodes of T. We focus on the two largest pieces π_{ij} and $\pi_{i'j}$ so that $\pi_{ij} + \pi_{i'j} \geq 2\lambda_j/k_j$. Now fix a leaf r that is connected to i via a path that does not involve j as an edge. This is possible since j is connected to interior nodes only. Now we intend to move job j *away* from r, i.e., we transform the schedule such that the path between node r and job j becomes longer. (The path between a node v and a job e in a tree starts at v and uses edges $e' \neq e$ until a node is reached that has e as an incident edge.) We do this iteratively until j is incident to a leaf in T. Then we can apply the transformations described above and we are done.

We first apply Lemma 4 to move small pieces of jobs allocated to machine i' to machine i. Although this changes the shape of T, it leaves the distance between jobs j and r invariant unless j ends up in machine i' completely so that we can apply Lemma 6 and we are done. When Lemma 4 is no longer applicable, either j saturates machine i' and we are done or there is a job q with $\pi_{i'q} > \pi_{ij}$. In that case we consider the smallest other piece $\pi_{i_q q}$ of job q. More precisely, if q is split into at most $k_q - 1$ nonzero pieces we pick some i_q with $\pi_{i_q q} = 0$. Otherwise we pick $i_q = \min\left\{\ell \neq i' : \pi_{\ell q} > 0\right\}$. In either case $\pi_{i_q q} \leq \lambda_q/(k_q - 1)$. Recall that $\pi_{ij} + \pi_{i'j} \geq 2\lambda_j/k_j$ since this sum is invariant under the move operations we have performed. Furthermore, j is the bulkiest job so that

$$\pi_{i_q q} \leq \frac{\lambda_q}{k_q - 1} \leq \frac{\lambda_j}{k_j - 1} = \frac{2\lambda_j}{(k_j - 1) + (k_j - 1)} \leq \frac{2\lambda_j}{(k_j - 1) + 1} = \frac{2\lambda_j}{k_j} \leq \pi_{ij} + \pi_{i'j} \ .$$

Hence, we can apply Lemma 5 to pieces i and i' of job j and to pieces i' and i_q of job q. This transformation increases the distance from job j to machine r as desired. Figure 2-(e) gives an example where we apply Lemma 4 once and then Lemma 5. □

3 Finding the optimal makespan

In the previous section, we assumed that an upper bound on the optimal makespan is known. The obvious idea now is to find the optimal makespan using binary search. In order to show that this search terminates one needs to prove that the optimal makespan is a rational number. This is not completely obvious as in principle jobs might be broken into pieces of non-rational size. The following lemma, however, shows that the optimal makespan can be represented by rational numbers of polynomial length. Let $\mathbb{Q}_\ell$ denote the set of non-negative rational numbers that can be represented by an ℓ-bit nominator and an ℓ-bit denominator and the symbol ∞.

Lemma 8. *There is a constant $\kappa > 0$ such that the value of an optimum solution to the SAC problem with splittability at least two is in $\mathbb{Q}_{N^\kappa}$, with N denoting the problem size.*

Proof. Let the task weights $\lambda_1, \ldots, \lambda_n$ and the machine speeds $s_1, \ldots, s_m$ be rational from $\mathbb{Q}_N$.

The problem of finding k-splittable assignments can be characterized as follows. Let $M_j \subseteq [m]$ be a set of machines with $|M_j| \leq k_j$ and such that task j can be assigned only to machines of set M_j (i.e. $x_{ij} = 0$ for all $i \notin M_j$). Having fixed the sets $M_1, \dots, M_n$, we can write our problem using the standard LP formulation with the additional constraint $x_{ij} = 0$, for $i \in [m] \setminus M_j$.

$$\begin{array}{lll} \min & z & \\ \text{s.t.} & \sum_{i=1}^{m} x_{ij} = 1 & \forall j \in [n] \\ & \sum_{j=1}^{n} \lambda_j x_{ij} \le z s_i & \forall i \in [m] \\ & x_{ij} = 0 & \forall j \in [n]\ \forall i \in [m] \setminus M_j \\ & x_{ij} \in [0,1] & \forall i, j. \end{array}$$

Thus for a fixed vector of sets $M_1, \dots, M_n$, our problem is a polynomial-size linear program (LP). Consequently, an optimum solution to our problem is attained as the value of this LP for one of the choices of the sets $M_1, \dots, M_n$. The value of an optimum solution to this LP (and thus also for our problem) belongs to $\mathbb{Q}_{N^\kappa}$, for some suitable constant κ. □

The lemma implies that the optimal makespan can be calculated by using binary search methods over the rationals (see, e.g., [13, 16]) with Algorithm 1 as a decision oracle. Thus, we obtain:

Corollary 9. *For every fixed number of machines, there exists an exact polynomial time optimization algorithm for the SAC problem with splittability at least two.*

4 Solving the traffic allocation problem

In this section, we show how to apply the binary search approach to the traffic allocation problem, that is, we solve the SAC problem with non-linear latency functions. We need to make some very modest assumptions about these functions. A latency function is called *monotone* if it is positive and non-decreasing. The functions need not to be continuous or strictly increasing, e.g., step functions are covered. We define the "inverse" of such functions as follows. Given a monotone function $f : \mathbb{Q}_{\ge 0} \to \mathbb{Q}_{\ge 0} \cup \{\infty\}$, let the *inverse of* f be defined by $f^{-1}(y) = \sup\{\lambda | f(\lambda) \le y\}$, for $y \ge f(0)$, and $f^{-1}(y) = 0$, for $y < f(0)$.

We say that a function f is *polynomially length-bounded* if for every $\lambda \in \mathbb{Q}_\ell$, $f(\lambda) \in \mathbb{Q}_{poly(\ell)}$. (Recall that $\mathbb{Q}_\ell$ denotes the set of non-negative rational numbers that can be represented by an ℓ bit nominator and an ℓ-bit denominator and the symbol ∞.) For example, the functions in the M/M/1 waiting time family are polynomially length-bounded although $\lim_{\lambda \to b^-} f_s(\lambda) = \infty$. This is because, for $\lambda, s \in \mathbb{Q}_\ell$ with $\lambda < s$, one can show $(s - \lambda) \in \mathbb{Q}_{2\ell}$, $s(s-\lambda) \in \mathbb{Q}_{4\ell}$ and $\lambda/(s(s-\lambda)) \in \mathbb{Q}_{8\ell}$ so that $f(\lambda) \in \mathbb{Q}_{8\ell}$. Furthermore, we say that a family of latency functions $\mathcal{F}$ is *efficiently computable* if, for every s and λ from $\mathbb{Q}_\ell$, $f_s(\lambda)$ as well as $f_s^{-1}(\lambda)$ can be calculated in time polynomial in ℓ. Observe that the functions from an efficiently computable family must also be polynomially length-bounded. It is easy to check that the M/M/1 waiting time family as well as other typical latency functions from Queueing Theory are efficiently computable in this sense.

Theorem 10. *Let $\mathcal{F}$ be any efficiently computable family of monotone functions. Consider the SAC problem with latency functions from $\mathcal{F}$ and splittability at least two. Suppose the value of the best possible maximum latency can be represented by a number from $\mathbb{Q}_\ell$. Then an optimal solution can be found in time $O\Big(poly(\ell)\cdot(n+m^{m+m/(k_0-1)})\Big)$ with $k_0=\min\{k_j : j=1,2,\dots,n\}$.*

Proof. We will first argue how to solve a decision version of SAC with latency functions from $\mathcal{F}$. Let z be the specified bound on the cost. The decision problem is described by the feasible region of the following program with an additional side constraint that for any job j there are at most k_j variables x_{ij} with positive value.

$$\begin{array}{rl}
\sum_{i=1}^m x_{ij} = 1 & \forall j\in[n]\\
f_{s_i}\left(\sum_{j=1}^n \lambda_j x_{ij}\right) \le z & \forall i\in[m]\\
x_{ij}\in[0,1] & \forall i,j.
\end{array}$$

Here x_{ij} is the fraction of task j assigned to server i. The feasible region of this program is equivalent to the region described by the following linear program,

$$\begin{array}{rl}
\sum_{i=1}^m x_{ij} = 1 & \forall j\in[n]\\
\sum_{j=1}^n \lambda_j x_{ij} \le f_{s_i}^{-1}(z) & \forall i\in[m]\\
x_{ij}\in[0,1] & \forall i,j.
\end{array}$$

Now observe this program describes the decision version of the SAC problem with machine speeds $f_{s_i}^{-1}(z)$ and makespan 1. As the inverse of the cost functions in $\mathcal{F}$ can be calculated in polynomial time, the sizes of the two programs are polynomially equivalent. Thus, solving the scheduling problem takes time polynomial in the size of the input to the SAC problem under $\mathcal{F}$. We run the exact algorithm Algorithm 1 from Theorem 1 on this decision version of SAC. It is easy to see that this algorithm outputs a feasible solution to SAC if and only if there is a feasible solution to the SAC problem under $\mathcal{F}$. Let $O\Big(poly^*(\ell)\cdot(n+m^{m+m/(k_0-1)})\Big)$ be a bound on the running time of this decision algorithm for SAC under $\mathcal{F}$.

The bound ℓ on the number of bits to represent possible values of the makespans can now be used to perform the binary search by using the previous algorithm as an oracle. For this purpose we use methods of searching for rationals [13, 16]. The overall running time of the algorithm is $O\Big(\ell\cdot poly^*(\ell)\cdot(n+m^{m+m/(k_0-1)})\Big) = O\Big(poly(\ell)\cdot(n+m^{m+m/(k_0-1)})\Big)$. $\square$

Observe that ℓ is an obvious lower bound on the running time of any algorithm that computes the exact, optimal makespan. It remains unclear whether there exists latency functions for which ℓ cannot be bounded polynomially in the input length. If an appropriate upper bound on ℓ is not known in advance, then we can use the geometric search algorithm. This search algorithm can be stopped any time after computing the optimal maximum latency with sufficient precision.

5 Non-approximability for non-linear scheduling

The M/M/1 waiting time cost function family as specified in the Introduction has an infinity pole for $\lambda \to b^-$. Intuitively, this infinity pole reflects the capacity restriction on the servers and it is typical also for other families of latency functions that can be derived from Queueing Theory. The following theorem shows that the non-linear k-splittable scheduling can be viewed as completely inapproximable. This negative result holds even when all servers are identical.

Theorem 11. *Let $\mathcal{F}$ be an efficiently computable family of monotone latency functions. Suppose there exists $s \in \mathbb{Q}_{>0}$ such that $\lim_{\lambda \to s} f_s(\lambda) = \infty$. Then there does not exist a polynomial time approximation algorithm with finite approximation ratio for the non-linear k-splittable scheduling problem under $\mathcal{F}$, provided* $\mathsf{P} \neq \mathsf{NP}$.

Proof. For the purpose of contraction, let us assume there is an α-approximation for the non-linear k-splittable scheduling problem for identical servers with latency function f_s. (α might be any finite approximation ratio, e.g., defined by some function on the input length.) We show that we can decide the NP-hard k-splittable scheduling problem in polynomial time under this assumption as follows. Suppose we are given jobs with sizes $\lambda_1, \ldots, \lambda_n$ and a makespan c, and have to decide whether there is a k-splittable schedule with makespan at most c. Define $c' = c + 2^{-N^\kappa}$ with N denoting the problem size and κ being the constant specified in Lemma 8. Observe that this lemma gives a separation bound in the sense that, if the optimum makespan for $\lambda_1, \ldots, \lambda_n$ is in the interval $[c, c']$, then it is c because $[c, c'] \cap \mathbb{Q}_{N^\kappa} = \{c\}$. Now we define $\lambda'_j := s\lambda_j / c'$ and run the approximation algorithm with the new weights. If this algorithm returns a solution with finite maximum latency then the scheduling problem is feasible, otherwise it is infeasible. Thus we can decide the given k-splittable scheduling problem in polynomial time, which contradicts $\mathsf{P} \neq \mathsf{NP}$. $\square$

6 Splittable scheduling under heterogeneous traffic

Until now we implicitly assume homogeneous traffic, i.e., all streams have the same (general) session length distribution. However, several practical studies show that Internet traffic is far away from being homogenous [3, 6, 17]. Assuming heterogeneous traffic, one has to take into account different session lengths distributions.

Let us investigate the consequences of heterogeneous traffic using a particular instance of a cost function. The *Pollaczek-Khinchin* (P-K) formula (see, e.g., [11]) describes expected waiting time in M/G/1 queues. Observe that this is a very general class of queueing systems that allows arbitrary service time distributions.

Suppose every stream i is characterized by two weights λ_j and V_j (corresponding to the expected load of data stream (job) j and the variance of this load, respectively). Then the P-K family of cost functions describing the expected time on a server (machine) with speed s_i can be defined as (see [4] for more details):

$$f_{s_i}(x_{i1}, \ldots, x_{in}) = \frac{\sum_{j=1}^{n} V_j x_{ij}}{s_i \left(s_i - \sum_{j=1}^{n} \lambda_j x_{ij}\right)} .$$

The remarkable fact here is that both parameters, the expected load and the variance, can be aggregated independently in a simple linear fashion. That is, the expected load injected into server i is $\lambda = \sum_{j=1}^{n} \lambda_j x_{ij}$ and the variance of this load is $V = \sum_{j=1}^{n} V_i x_{ij}$.

We now show how to use our algorithm for the splittable scheduling here. Consider the heterogeneous splittable scheduling problem where the delay on identical machines is given by the P-K cost function, that is $s_1 = s_2 = \ldots = s_m = 1$. Then, the optimum solution in this model can be characterized by the following program with additional splittability constraints.

$$\begin{array}{lcl} min & z & \\ s.t. & \frac{\sum_{j=1}^{n} V_j x_{ij}}{s_i(s_i - \sum_{j=1}^{n} \lambda_j x_{ij})} \leq z & \forall i \in [m] \\ & \sum_{i=1}^{m} x_{ij} = 1 & \forall j \in [n] \\ & x_{ij} \geq 0 & \forall i, j. \end{array}$$

This program can be equivalently rewritten into the following program, called P_1.

$$\begin{array}{lcl} min & z & \\ s.t. & \sum_{j=1}^{n} \left(V_j/z + \lambda_j\right) x_{ij} \leq 1 & \forall i \in [m] \\ & \sum_{i=1}^{m} x_{ij} = 1 & \forall j \in [n] \\ & x_{ij} \geq 0 & \forall i, j. \end{array}$$

Observe, that if z is fixed, then P_1 with additional spittability constraints is equivalent to the decision version of the splittable scheduling (SAC). Thus, it can be solved by our algorithm from Section 2. Observe, that the P-K cost function family is an efficiently computable family of monotone functions – see the discussion in Section 4. Applying arguments analogous to these in the proof of Theorem 10 we can obtain the following result.

Theorem 12. *Let $\mathcal{F}$ be the family of P-K cost functions. Consider the SAC problem with identical unit speed machines, with latency functions from $\mathcal{F}$, and splittability at least two. Suppose the value of the best possible maximum latency can be represented by a number from $\mathbb{Q}_\ell$. Then an optimal solution can be found in time $O\Big(poly(\ell) \cdot (n + m^{m+m/(k_0-1)})\Big)$ with $k_0 = \min\{k_j : j = 1, 2, \ldots, n\}$.*

References

[1] N. Alon, Y. Azar, G. J. Woeginger and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

[2] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *J. Computer and System Science*, 7(4):448–461, August 1973.

[3] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic - evidence and possible causes. In *Proceedings of ACM Sigmetrics'96*, pages 160–169, 1996.

[4] A. Czumaj, P. Krysta and B. Vöcking. Selfish Traffic Allocation for Server Farms. In *Proc. ACM STOC*, 2002.

[5] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Proc. of the 7th ESA*, 151–162, 1999.

[6] A. Feldmann, A.C. Gilbert, P. Huang, and W. Willinger. A study of the role of variablity and the impact of control. In *Proc. of the ACM SIGCOMM'99*, 1999.

[7] M. R. Garey and D. S Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* Freeman, 1979.

[8] D. S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34: 144–162, 1987.

[9] D. S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17: 539–551, 1988.

[10] E. Horowitz and S. K. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23:317–327, 1976.

[11] L. Kleinrock. *Queueing Systems. Volume I: Theory.* John Wiley & Sons, New York, NY, 1975.

[12] J. F. Kurose and K. W. Ross. *Computer networking: a top-down approach featuring the Internet.* Addison-Wesley, 2001.

[13] St. Kwek and K. Mehlhorn. Optimal search for rationals. *Information Processing Letters*, to appear, 2002.

[14] J.K. Lenstra, D.B. Shmoys and E. Tardos. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Mathematical Programming*, 46: 259–271, 1990.

[15] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[16] C. H. Papadimitriou. Efficient search for rationals. *Information Processing Letters*, 8:1–4, 1979.

[17] K. Park, G. Kim, and M.E. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. IEEE International Conference on Network Protocols*, pages 171–180, 1996.

[18] H. Shachnai and T. Tamir. Multiprocessor Scheduling with Machine Allotment and Parallelism Constraints. *Algorithmica*, 32(4): 651–678, 2002.

[19] W. Xing and J. Zhang. Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, 103: 259–269, 2000.

MPI-I-2003-NWG2-002	F. Eisenbrand	Fast integer programming in fixed dimension
MPI-I-2003-NWG2-001	L.S. Chandran, C.R. Subramanian	Girth and Treewidth
MPI-I-2003-4-002	C. Theobalt, M. Li, M. Magnor, H. Seidel	A Flexible and Versatile Studio for Synchronized Multi-view Video Recording
MPI-I-2003-4-001	M. Tarini, H.P.A. Lensch, M. Goesele, H. Seidel	3D Acquisition of Mirroring Objects
MPI-I-2003-4–003	I. Ivrissimtzis, W. Jeong, H. Seidel	Neural Meshes: Statistical Learning Methods in Surface Reconstruction
MPI-I-2003-2-002	M. Jaeger	A Representation Theorem and Applications to Measure Selection and Noninformative Priors
MPI-I-2003-2-001	P. Maier	Compositional Circular Assume-Guarantee Rules Cannot Be Sound And Complete
MPI-I-2003-1-011	P. Krysta, A. Czumaj, B. Voecking	Selfish Traffic Allocation for Server Farms
MPI-I-2003-1-010	H. Tamaki	A linear time heuristic for the branch-decomposition of planar graphs
MPI-I-2003-1-009	B. Csaba	On the Bollobás – Eldridge conjecture for bipartite graphs
MPI-I-2003-1-008	P. Sanders	Soon to be published
MPI-I-2003-1-007	H. Tamaki	Alternating cycles contribution: a strategy of tour-merging for the traveling salesman problem
MPI-I-2003-1-006	H. Tamaki, M. Dietzfelbinger	On the probability of Rendezvous in Graph
MPI-I-2003-1-005	M. Dietzfelbinger, P. Woelfel	Almost Random Graphs with Simple Hash Functions
MPI-I-2003-1-004	E. Althaus, T. Polzin, S.V. Daneshmand	Improving Linear Programming Approaches for the Steiner Tree Problem
MPI-I-2003-1-003	R. Beier, B. Vcking	Random Knapsack in Expected Polynomial Time
MPI-I-2003-1-002	P. Krysta, P. Sanders, B. Vcking	Scheduling and Traffic Allocation for Tasks with Bounded Splittability
MPI-I-2003-1-001	P. Sanders, R. Dementiev	Asynchronous Parallel Disk Sorting
MPI-I-2002-4-002	F. Drago, W. Martens, K. Myszkowski, H. Seidel	Perceptual Evaluation of Tone Mapping Operators with Regard to Similarity and Preference
MPI-I-2002-4-001	M. Goesele, J. Kautz, J. Lang, H.P.A. Lensch, H. Seidel	Tutorial Notes ACM SM 02 A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2002-2-008	W. Charatonik, J. Talbot	Atomic Set Constraints with Projection
MPI-I-2002-2-007	W. Charatonik, H. Ganzinger	Symposium on the Effectiveness of Logic in Computer Science in Honour of Moshe Vardi

MPI-I-2002-1-008	P. Sanders, J.L. Trff	The Factor Algorithm for All-to-all Communication on Clusters of SMP Nodes
MPI-I-2002-1-005	M. Hoefer	Performance of heuristic and approximation algorithms for the uncapacitated facility location problem
MPI-I-2002-1-004	S. Hert, T. Polzin, L. Kettner, G. Schfer	Exp Lab A Tool Set for Computational Experiments
MPI-I-2002-1-003	I. Katriel, P. Sanders, J.L. Trff	A Practical Minimum Scanning Tree Algorithm Using the Cycle Property
MPI-I-2002-1-002	F. Grandoni	Incrementally maintaining the number of l-cliques
MPI-I-2002-1-001	T. Polzin, S. Vahdati	Using (sub)graphs of small width for solving the Steiner problem
MPI-I-2001-4-005	H.P.A. Lensch, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2001-4-004	S.W. Choi, H. Seidel	Linear One-sided Stability of MAT for Weakly Injective Domain
MPI-I-2001-4-003	K. Daubert, W. Heidrich, J. Kautz, J. Dischler, H. Seidel	Efficient Light Transport Using Precomputed Visibility
MPI-I-2001-4-002	H.P.A. Lensch, J. Kautz, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing, Transmission, and Interactive Display of High Quality 3D Models on the Web
MPI-I-2001-4-001	H.P.A. Lensch, J. Kautz, M. Goesele, W. Heidrich, H. Seidel	Image-Based Reconstruction of Spatially Varying Materials
MPI-I-2001-2-006	H. Nivelle, S. Schulz	Proceeding of the Second International Workshop of the Implementation of Logics
MPI-I-2001-2-005	V. Sofronie-Stokkermans	Resolution-based decision procedures for the universal theory of some classes of distributive lattices with operators
MPI-I-2001-2-004	H. de Nivelle	Translation of Resolution Proofs into Higher Order Natural Deduction using Type Theory
MPI-I-2001-2-003	S. Vorobyov	Experiments with Iterative Improvement Algorithms on Completely Unimodel Hypercubes
MPI-I-2001-2-002	P. Maier	A Set-Theoretic Framework for Assume-Guarantee Reasoning
MPI-I-2001-2-001	U. Waldmann	Superposition and Chaining for Totally Ordered Divisible Abelian Groups
MPI-I-2001-1-007	T. Polzin, S. Vahdati	Extending Reduction Techniques for the Steiner Tree Problem: A Combination of Alternative-and Bound-Based Approaches
MPI-I-2001-1-006	T. Polzin, S. Vahdati	Partitioning Techniques for the Steiner Problem
MPI-I-2001-1-005	T. Polzin, S. Vahdati	On Steiner Trees and Minimum Spanning Trees in Hypergraphs
MPI-I-2001-1-004	S. Hert, M. Hoffmann, L. Kettner, S. Pion, M. Seel	An Adaptable and Extensible Geometry Kernel
MPI-I-2001-1-003	M. Seel	Implementation of Planar Nef Polyhedra
MPI-I-2001-1-002	U. Meyer	Directed Single-Source Shortest-Paths in Linear Average-Case Time
MPI-I-2001-1-001	P. Krysta	Approximating Minimum Size 1,2-Connected Networks
MPI-I-2000-4-003	S.W. Choi, H. Seidel	Hyperbolic Hausdorff Distance for Medial Axis Transform
MPI-I-2000-4-002	L.P. Kobbelt, S. Bischoff, K. Khler, R. Schneider, M. Botsch, C. Rssl, J. Vorsatz	Geometric Modeling Based on Polygonal Meshes
MPI-I-2000-4-001	J. Kautz, W. Heidrich, K. Daubert	Bump Map Shadows for OpenGL Rendering
MPI-I-2000-2-001	F. Eisenbrand	Short Vectors of Planar Lattices Via Continued Fractions
MPI-I-2000-1-005	M. Seel, K. Mehlhorn	Infimaximal Frames: A Technique for Making Lines Look Like Segments
MPI-I-2000-1-004	K. Mehlhorn, S. Schirra	Generalized and improved constructive separation bound for real algebraic expressions
MPI-I-2000-1-003	P. Fatourou	Low-Contention Depth-First Scheduling of Parallel Computations with Synchronization Variables
MPI-I-2000-1-002	R. Beier, J. Sibeyn	A Powerful Heuristic for Telephone Gossiping
MPI-I-2000-1-001	E. Althaus, O. Kohlbacher, H. Lenhof, P. Mller	A branch and cut algorithm for the optimal solution of the side-chain placement problem