Dagstuhl Scheduling 2010

Open Problems

April 27, 2010

Contents

Dagstuhl Seminar Proceedings 10071
Scheduling
http://drops.dagstuhl.de/opus/volltexte/2010/2536

Open Problem

Proposed by Jim Anderson

Problem Statement: We wish to schedule a system of n sporadic tasks with implicit deadlines on m processors. Execution costs for each job of a task are independent and identically distributed according to some probability distribution, with known mean and variance. Tasks may be over-utilized (in the worst case), provided that they are under-utilized in the average case, and total average utilization does not exceed m. Mills and Anderson (RTAS 2010) show that GEDF can schedule such a task system in a way that the mean and quantiles of the tardiness distribution are bounded from above by a constant, if worst-case execution times are also known; the result extends to a general class of scheduling algorithms. The open problem is to relax the assumption that the execution costs for each job of a task are independent from one another.

Notes:

- The independence assumption is used in the analysis of the processor sharing schedule, where each processor share is analyzed as an independent $G/G/1$ queue. The upper bound on waiting time in a $G/G/1$ queue relies on the fact that U_j, the difference between the service time of the jth customer and the jth inter-arrival time, are iid.
- We would like to accommodate tasks that may execute in several states, each of which has its own execution-time distribution; for example, the task might change state between jobs deterministically or according to a Markov chain. In this case, it is clear that execution times of successive jobs will not be independent.
- Since execution-time distributions are non-negative, would it help to assume that they can be approximated by a phase-type distribution?

For which uniprocessor scheduling problems are non-preemptive not-inserting-idle-time EDF optimal?

Proposed by Björn Andersson

Problem Statement: Consider n constrained-deadline sporadic tasks to be scheduled by online non-preemptive EDF without inserted-idle time. We say that an algorithm is an online algorithm if the algorithm knows the task parameters (T, D, C) of each task but it does not know the release times of each job before the job has been released. We say that an algorithm is an offline algorithm if the algorithm knows the task parameters (T, D, C) of each task and it also knows the release times of each job before the job has been released. We say that a task set is schedulable by an online algorithm A if deadlines of all jobs released by the task set meets deadlines when scheduled by algorithm A; and this should hold for every possible release that is possible according to the sporadic model.

It is well-known that there is an off-line feasible task sets which is not schedulable by non-preemptive EDF without inserted-idle time. We are interested however in the following two questions:

- If a task set is online feasible does it imply that the task set is schedulable by non-preemptive EDF without inserted-idle time;
- If a task set is offline feasible and the task set has implicit-deadlines (that is, $D_i = T_i$) does it imply that the task set is schedulable by non-preemptive EDF without inserted-idle time.

Online preemptive routing in general graphs

Proposed by Yossi Azar

Problem Statement: We are given a graph with large capacities (at least $\log m$ where m is the graph size) and a sequence of requests (paths from s_i to t_i). The problem is to accept or reject each path as to maximize the number of accepted paths (throughput) while maintaining the capacity constraints. Find a constant competitive algorithm or non-constant lower bound. The best known algorithm (achieved by non-preemptive algorithm) is $O(\log m)$ competitive so even to get below $O(\log m)$ is open.

Related Results and Comments:

- Offline (preemption is meaningless): Constant approximation or even $1 + \epsilon$ assuming the capacities are at least $\log m/\epsilon 2$ can be found by solving the factional problem (multi-commodity flow) and rounding (Raghavan and Thompson Combinatorica 1988).
- On-line - no preemption. A tight $O(\log m)$ competitive algorithm is achieved in Awerbuch, Azar and Plotkin (Focs 1993). The lower bound (as well as upper bound for special graphs with low capcities) is achieved by Awerbuch, Bartal, Fiat, and Rosen (Soda 1994), Lipton and Tomkins (Soda 1994), Awerbuch, Gawlick, Leighton, and Rabani (Focs 1994) Garay, Gopal, Kutten, Mansour, and Yung (ICTCS 1993).
- Special graphs - with preemption. Constant competitive algorithm is achieved for the line in Adler and Azar (SODA 99) and for trees in Azar, Feige, and Glasner (Swat 2008).
- Small capacities. For capacities which are 1 (i.e. disjoint path problem) $\Omega(n^\epsilon)$ lower bound on the competitive ratio was shown in Bartal, Fiat, and Leonardi (Stoc 1996). Even for the offline version (polynomial time algorithm) it is hard to get small approximation Andrews, Chuzhoy, Khanna, and Zhang (Focs 2005).
- Comments: for large capacities randomization does not seem to help for online algorithms (where it may help for small capacities).

Coping with non-preemption

Proposed by Nikhil Bansal

Problem Statement: We are given n jobs with arbitrary release times and arbitrary sizes. There is a single machine and we wish is to find a non-preemptive schedule with minimum total flow time. Here, flow time of a job is the difference between its completion time and its release time.

The problem is known to be hard to approximate within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$ [2]. The result is based on a reduction from 3-Partition and is carefully based on exploiting that numbers just "fit right". Thus it is natural to expect that much better guarantees may be possible with $(1+\epsilon)$-speed. [3] gave an $O(\log n)$-machine, $(1+\epsilon)$-approximation polynomial time algorithm. However this result is somewhat unsatisfying for two reasons: the resource augmentation is not constant, and the algorithm is quite naive as it merely uses one machine for all jobs of about the same size.

One reason why it is hard to obtain better bounds is that any natural time indexed LP (that we can think of) is quite weak: It does not distinguish between preemptive and non-preemptive schedules, which can have huge gaps, see [3] for an example. Recently, [1] considered a somewhat stronger LP that captured (a limited) effect of non-preemption and used it to obtain a 12-speed, $O(1)$-approximation. However, they also show that if only $2-\epsilon$ speedup is allowed, then the LP has an integrality gap of n^c for some constant c. In particular the LP is useless with only $(1+\epsilon)$-speed.

Open Question: Is there an algorithm with "reasonable" approximation guarantee with only $(1+\epsilon)$ speedup.

I conjecture that this is true, and believe that designing such an algorithm should give useful insights into coping with non-preemption.

References:

[1] N. Bansal, H.L. Chan, R. Khandekar, K. Pruhs, B. Schieber and C. Stein. Non-Preemptive min-sum scheduling with resource augmentation. FOCS 2007.

[2] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. STOC 1996.

[3] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. Algorithmica, 32, 2001.

Optimal design of an EDF task set

Proposed by Enrico Bini

Schedulability analysis requires to check whether a real-time task set will miss or not any deadline. When designing a system, however, the designer has often to face the problem of choosing these values. This leads to the problem of optimal design of an EDF task set. This problem is extremely common in control systems.

Let the task τ_i be modeled by a computation time C_i, a period T_i, and a deadline D_i. We formulate the problem as follows.

$$\begin{aligned}
\text{given } & n, C_i \\
\text{find } & T_i, D_i \\
\text{minimize } & \max_i F_i(T_i, D_i) && (1) \\
\text{subject to } & \text{task set } \{\tau_i\} \text{ is EDF schedulable.} && (2)
\end{aligned}$$

The function F_i of Eq. (1) is the cost of task τ_i. We assume it is differentiable and $\frac{\partial F_i}{\partial T_i} \geq 0$, $\frac{\partial F_i}{\partial D_i} \geq 0$, since it is often the case that by reducing the periods/deadlines we also improve the quality of the system. The overall system cost (1) can be sometime modeled also as

$$\sum_i F_i(T_i, D_i).$$

The constraint of EDF schedulability (2) can be expressed in one of the following equivalent ways

$$\forall t \geq 0 \quad \sum_{i=1}^{m} \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i \leq t,$$

or

$$\forall \mathbf{k} \in \mathbb{N}^n \setminus \{\mathbf{0}\} \quad \exists i : k_i \geq 1 \quad \sum_{j \neq i} C_j k_j - (T_i - C_i) k_i \leq D_i - T_i.$$

In the case of constrained deadlines (for all i, $D_i \leq T_i$), the EDF schedulability condition can be simplified, and it becomes one of the following two equivalent relations.

$$\forall t \geq 0 \quad \sum_{i=1}^{m} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i \leq t$$

or

$$\mathbb{N} \setminus \{\mathbf{0}\} \subseteq \bigcup_{i=1}^{m} \mathsf{domK}_i$$

with

$$\mathsf{domK}_i = \{\mathbf{k} \in \mathbb{Z}^m : \sum_{j \neq i} C_j k_j - (T_i - C_i) k_i \leq D_i - T_i\}.$$

IRS audit scheduling

Proposed by Marek Chrobak

An IRS auditor needs to schedule interviews with n taxpayers. Each interview can be scheduled in a unit time interval $[a, a+1)$, for some integer a. Each taxpayer j has an interval $[r_j, d_j)$ when he/she is available, for some integers r_j, d_j. If taxpayers i, j are married, they can be scheduled at the same time slot, if their availability intervals overlap. If i, j are not married, they must be scheduled at different time slots. Is there a polynomial-time algorithm to determine if all taxpayers can be scheduled?

Suppose that all availability intervals have length K (that is, $d_j - r_j = K$ for al j). Can this special case be solved in polynomial time?

Related Results and Comments:

- **Motivation:** It's a cute puzzle. Also, the problem arises in the design of approximation algorithms for broadcast scheduling. A positive answer (even for the special case above) would give a randomized 1.75-approximation algorithm for broadcast scheduling.
- **Related Results:** If $|r_j - r_i| = D$ for all married couples i, j, then the problem can be solved in polynomial time, by expressing it as a integer linear program and observing that the matrix of this LP is totally unimodular [unpublished].

Weighted completion time and selfish scheduling

Proposed by José Correa

We are given n jobs which can be processed in any of the m available machines. If job j is processed on machine i it takes p_{ij} time units to complete. Also each job j has a weight w_j. In our context, jobs are players who seek to minimize their own completion time. To this end, job j selects as strategy a probability distribution over the machines $(\pi_i^j)_{i=1}^m$, meaning that it will choose machine i with probability π_i^j. On the other hand, each machine i will process the jobs that end up being assigned to it according to Smith rule (i.e., in nonincreasing order of w_j/p_{ij}).

A Nash equilibrium is a situation in which the expected completion time C_j of every job j under its current strategy is minimum given the strategies of all other jobs, and its social cost is $E[\sum w_j C_j]$. An optimal solution is a centralized schedule minimizing $\sum w_j C_j$ (which is NP-hard to compute but can be approximated within a factor of $3/2 + \epsilon$ by a randomized rounding approach of Schulz and Skutella).

Question: Does there exist a constant α such that the social cost of any Nash equilibrium is at most α times the optimal cost? In other words, is the price of anarchy of this scheduling game constant?

Note 1: The game may not have pure strategy Nash equilibria.

Note 2: If $p_{ij} \in \{p_j \cdot s_i, +\infty\}$ the answer is positive and the price of anarchy is exactly 4 (Correa and Queyranne 2009).

Open Problem

Proposed by Liliana Cucu-Grosjean

Problem Statement: Periodic tasks with release times, deadlines and (probabilistic) variable execution times. The problem is to determine if there is an optimal fixed-priority algorithm that can schedule these tasks with arbitrary preemption on one processor. To the best of my knowledge this open problem arises from the paper:

D. Maxim and L. Cucu-Grosjean, Towards optimal priority assignment for probabilistic real-time systems with variable execution times, Proceedings of the 3rd Junior Researcher Workshop on Real-Time Computing (JRWRTC 2009)

Related Results and Comments:

- Optimality for fixed-priority algorithms: such algorithm is optimal in the sense that if there is (at least) a priority assignment that satisfies the constraints then the algorithm will find it. No (general) optimal algorithm is known for this problem.
- The satisfaction of the constraints is verified using results provided in J.L Diaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, J.M. Lopez and O. Mirabella, Stochastic Analysis of Periodic Real-Time Systems, Proceedins of 23rd of the IEEE Real-Time Systems Symposium (RTSS 2002)

Open Problem

Proposed by Rob Davis

Problem: What is the pattern of job arrivals that leads to the longest response time (from arrival to completion) of any job of task τ_k?

This problem can be posed for a number of different task models. These are:

(i) Concrete periodic tasks with a synchronous arrival sequence: By periodic, we mean that the next job of task τ_k arrives exactly T_k time units after the arrival of the previous job of that task, by concrete, we mean that there is a fixed relationship between the arrival times of the first jobs of each task, in this case a synchronous arrival sequence, where the first job of each task arrives at time 0.

(ii) Non-concrete periodic tasks, where we do not know the relationship between the arrival times of the first job of each task.

(iii) Sporadic tasks, where the next job of a task τ_k may arrive at any time T_k or greater since the arrival of the previous job of that task.

The problem can also be posed for different constraints on task deadlines, for example:

(a) implicit deadlines $D_k = T_k$,
(b) constrained deadlines $D_k \leq T_k$, and
(c) arbitrary deadlines.

What we know: For the equivalent single processor problem (for tasksets complying with models (i), (ii), and (iii)) the interval during which the processor is busy executing jobs of priority k or higher, starting with synchronous arrival of jobs of all tasks, defines (for implicit or constrained deadline tasks) or includes (for arbitrary deadline tasks) the longest response time for any job of task k. However, this is known not to be the case for the multiprocessor problem.

In the multiprocessor case for concrete periodic tasks with a synchronous arrival sequence, we could simulate the schedule to the Least Common Multiple of task periods and thus find the longest response time of any job of task τ_k. Note, as global fixed priority pre-emptive scheduling is *predictable* [1] reducing job execution times to less than the maximum allowed cannot result in increased response times. Hence we only need simulate the schedule for jobs assuming the maximum possible execution times.

In the multiprocessor case (non-concrete periodic tasks / sporadic tasks) we can show that the worst-case occurs when task τ_k arrives at some time t when all m processors have just become busy with higher priority tasks (i.e. at time $t-1$ at most $m-1$ processors were busy with higher priority tasks, and at time t, all m processors are busy with higher priority tasks - assuming integer time). I have a simple proof of this, inspired by the work of Guan [2].

Related problems: Optimal priority assignment: How to find a priority assignment that results in a schedulable taskset (all worst-case response times less than or equal to deadlines) whenever such an ordering exists.

[1] Ha, R., and Liu, J.W-S., 1994. Validating timing constraints in multiprocessor and distributed real-time systems. In proceedings of the International conference on Distributed Computing Systems, pp. 162-171, 1994.

[2] Guan, N., Stigge, M., Yi, W., and Yu, G., 2009. New Response Time Bounds for Fixed Priority Multiprocessor Scheduling. In proceedings of the Real-Time Systems Symposium, 2009.

Hardness of compositional schedulability analysis

Proposed by Arvind Easwaran

Problem Statement: Jobs with release time, deadlines and known computation time arrive over time (set $\mathcal{J}$). Suppose these jobs are prioritized using the Earliest Deadline First (EDF) strategy and scheduled on a single machine. Define $D_{\mathcal{J}}(t)$ to be a function that gives the total computational requirement of $\mathcal{J}$ in the interval $(0, t]$ and under EDF. The problem is to determine if, for any given ϵ, there exists another set of jobs ($\mathcal{J}'$) such that:

- Computational restriction: $D_{\mathcal{J}'}(t)$ is a $1+\epsilon$-approximation of $D_{\mathcal{J}}(t)$, and
- Size restriction: $|\mathcal{J}'| = \mathcal{O}\left(\frac{1}{\epsilon}\log|\mathcal{J}|\right)$.

To the best of my knowledge this open problem arises from the following paper: [EALS] Arvind Easwaran, Madhukar Anand, Insup Lee, and Oleg Sokolsky, "On the Complexity of Generating Optimal Interfaces for Hierarchical Systems", Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2008).

Related Results and Comments:

EALS showed that for $\epsilon = 0$, it is feasible to generate a polynomial-sized (not poly-log) set of jobs $\mathcal{J}'$. This is indeed a trivial result for the chosen scheduling strategy.

EALS also demonstrated the hardness of a related problem through an example. Considering yet another popular scheduling strategy, it was shown that for $\epsilon = 0$, it is not possible to generate even a polynomial-sized job set. To the best of my knowledge, it is still an open problem to classify this hardness result.

Comments: We, as a community, have so far focused on $O(1)$-sized approximations, without considering any computational restrictions (i.e., $\epsilon = \infty$). Addressing the aforementioned problem is expected to open a pandora's box in the area of compositional schedulability analysis, leading to some very efficient solutions for long standing open problems.

Open Problem

Proposed by Jeff Edmonds

Problem Statement: The goal is to prove a surprising lower bound for resource augmented nonclairvoyant algorithms for scheduling jobs with sublinear nondecreasing speedup curves on multiple processors with the objective of average response time. Edmonds and Pruhs in SODA09 prove that for every $\epsilon > 0$, there is an algorithm Alg_ϵ that is $(1{+}\epsilon)$-speed $O(\frac{1}{\epsilon 2})$-competitive. A problem, however, is that this algorithm Alg_ϵ depends on ϵ. The goal is to prove that every fixed deterministic nonclairvoyant algorithm has a suboptimal speed threshold, namely for every (graceful) algorithm Alg, there is a threshold $1+\beta_{\mathrm{Alg}}$ that is $\beta_{\mathrm{Alg}} > 0$ away from being optimal such that the algorithm is $\Omega(\frac{1}{\epsilon\beta_{\mathrm{Alg}}})$ competitive with speed $(1{+}\beta_{\mathrm{Alg}}){+}\epsilon$ and is $\omega(1)$ competitive with speed $1{+}\beta_{\mathrm{Alg}}$. I have worked very hard on it and have felt that I was close. The proof technique is to use Brouwer's fixed point theorem to break the cycle of needing to know which input will be given before one can know what the algorithm will do and needing to know what the algorithm will do before one can know which input to give. Every thing I have can be found at

`http://www.cse.yorku.ca/ jeff/research/schedule/lowerbound.pdf` and
`http://www.cse.yorku.ca/ jeff/research/kirk/laps/laps.ppt`

Open Problem

Proposed by Shelby Funk

Problem Statement: Set of independent periodic sporadic tasks $\tau = \{T_1, T_2, \ldots, T_n\}$ and a uniform multiprocessor $\pi = [s_1, s_2, \ldots, s_m]$. Each $T_i = (p_i, e_i, D_i)$, where p_i is the period, e_i is the worst case execution time (WCET), and D_i is the deadline. If T_i generates a job at time a, then

- the job must be allowed to complete e_i units of work by time $a + D_i$,
- if T_i has multiple jobs with outstanding work, these jobs are executed in FIFO order,
- T_i cannot generate another job before time $a + p_i$ (if T_i is periodic, next job arrives exactly at time $a + p_i$), and
- T_i can only be executing on one processor at any point in time.

We want a set of rules such that any algorithm satisfying these rules can schedule any such system whenever it is possible to do so. Also, use these rules to generate new algorithms. Ideally, the algorithm will satisfy the following properties: (i) running time is at most $O(n)$, (ii) algorithm runs as infrequently as possible, (iii) preemptions and migrations are kept to a minimum.

Related Results and Comments: A task set is feasible if and only if

$$\begin{aligned}\sum_{i=1}^{k} u_i &\leq \sum_{i=1}^{k} s_i \text{ for all } k \leq m \\ U(\tau) &\leq \sum_{i=1}^{n} s_i,\end{aligned}$$

where $U(\tau)$ is the total utilization of τ (Funk, Goossens and Baruah, "On-line Scheduling On Uniform Multiprocessors", Proceedings of the IEEE Real-Time Systems Symposium, 2001).

Optimal algorithms already exist on identical multiprocessors such as Pfair (Baruah, Cohen, Plaxton and Varvel, "Proportionate progress: A Notion of Fairness in Resource Allocation", Algorithmica, 1996), LLREF (Cho, Ravindran and Jensen "An Optimal Real-Time Scheduling Algorithm for Multiprocessors", Proceedings of RTSS 2006), and BF (Zhu, Mosse and Melhem, "Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?"). These all guarantee that each task T_i has executed for $u_i \cdot t$ at certain points in time.

Rules for tasks executing on identical multiprocessors (i.e., $s_i = 1$ for all i). We divide the time into consecutive intervals, where we start a new interval whenever a task has a deadline. For periodic tasks with deadlines equal to periods, at the beginning of each interval $[t_0, t_f)$ assign each task T_i to execute for $u_i \cdot (t_f - t_0)$ time units and schedule the tasks as follows

- Always run any jobs that have zero laxity.
- Never run any jobs that have zero remaining execution time.
- Do not allow more than $(m - U(\tau)) \times (t_f - t_0)$ discretionary idle time.

Idle time is discretionary if processors idle while tasks are waiting to execute. We can show that if task idle for the specified amount of time then there will be at least m uncompleted jobs for the remainder of time.

Sporadic tasks may generate jobs to arrive within an interval at time $t \in [t_0, t_f)$. In this case, we add the following two rules

- If $t + p_i \geq t_f$, assign the newly arrived job to execute for $u_i \cdot (t_f - t)$.
- If $t + p_i < t_f$, assign the newly arrived job to execute for e_i, split the time slice into 2 pieces so that T_i's deadline coincides with the end of the first piece and divide other task's remaining execution time in proportion to the lengths of the two pieces.

When $D_i \neq p_i$, we use task density, $\delta_i = e_i / \min\{D_i, p_i\}$, instead of utilization to determine run times. We also use the density to determine feasibility, but this test is sufficient only when density is used. There is no optimal online multiprocessor scheduling algorithm when deadlines are less than periods (Fisher, Goossens and Baruah, "Optimal

Online Multiprocessor Scheduling of Sporadic Real-Time Tasks is Impossible", Real-Time Systems, to appear. 2010). Currently, we set boundaries at times $a_{i,k} + \min\{D_i, p_i\}$, where $a_{i,k}$ is the time when T_i releases its k^{th} job. Thus, when $D_i > p_i$, we have artificial boundaries (i.e., boundaries that do not coincide with any deadlines). This effectively forces all tasks to have $D_i \leq p_i$.

A simple algorithm: In each time interval, we can use McNaughton's wrap-around algorithm (McNaughton, "Scheduling with deadlines and loss functions", Machine Science, 1959) to determine a schedule for the entire the interval. This algorithm runs in $O(n)$ time at the beginning of each interval. For uniform multiprocessors, we can develop a schedule using the level algorithm (Horvath, Lam and Sethi, "A Level Algorithm for Preemptive Scheduling", Journal of the ACM, 1977).

The results presented above have been submitted to the EuroMicro Conference on Real-Time Systems. It is joint work with Greg Levin, Caitlin Sadowski, Ian Pye and Scott Brandt at the University of California at Santa Cruz.

Questions:

1. Given that the algorithm runs at the beginning of each interval, we clearly benefit from reducing the number of intervals. How much can we reduce the number of intervals? It seems clear we can allow some jobs to have deadlines within an interval. Any such job must be allowed to execute immediately and non-preemptively. The question is how many such jobs can we allow at any point in time?

2. Can we reduce preemptions and migrations in the uniform multiprocessor case? The level algorithm shares processors between jobs, which can cause numerous preemptions and migrations. Is there an algorithm that can reduce this overhead?

3. Can we schedule tasks with $D_i > p_i$ more efficiently? Removing artificial boundaries could also reduce the total number of intervals. In particular, if a task T_i with $D_i > p_i$ releases a job at time a, and some other task has at some time $t \in [a + p_i, a + D_i)$, then the T_i will have completed its work by time t. This would mean we would not have to set a boundary at time $a + D_i$. It's conceivable that we would rarely need to set boundaries for tasks with deadlines significantly larger than periods.

A sharp threshold for rate monotonic schedulability of real-time tasks

Proposed by Sathish Gopalakrishnan

Setting: For a set of n known task periods $T_1, T_2, \ldots, T_n$, let A_u represent the set of all *implicit-deadline task sets* of utilization u that are schedulable using the rate monotonic scheduling policy. In other words, every task set $\tau \in A_u$ is schedulable using RM and can be represented by a vector of utilizations $\{u_i\}$ such that $0 \leq u_i \leq u$ and $\sum_{i=1}^{n} u_i = u$. For large n and any given set of task periods, there exists a u^* such that, for any $\epsilon, 0 < \epsilon < 1$,

$$\mu(A_u) = \begin{cases} 0 & \text{if } u > (1+\epsilon)u^* \\ 1 & \text{if } u < (1-\epsilon)u^* \end{cases},$$

where $\mu(A_u)$ is the uniform probability measure of the set A_u on the n-dimensional simplex $\sum_{i=1}^{n} u_i = u, u_i \geq 0$ (i.e., the Lebesgue measure of A_u suitably normalized). This result can be obtained by an application of sharp threshold results for random graphs due to Friedgut and Kalai, and Bourgain.

Open problem: As $n \to \infty$, for a given set of task periods $T_1, \ldots, T_n$, what is the sharp threshold u^*?

Open Problem

Proposed by Han Hoogeveen

Problem Statement: We are looking at a single machine scheduling problem with release dates and equal processing times: the objective function is assumed to be regular. Then you know that there is an optimal schedule in which each job either starts at its release date, or starts at the completion time of some other job. This implies that there are at most $O(n^2)$ execution intervals for all jobs. Verma and Dessouky have presented an ILP formulation for a similar problem with binary variables that indicate whether a job gets assigned to an execution interval. They have shown that, if a double-nested solution is either suboptimal, or can be rewritten to a non-double-nested solution, then there exists an integral solution with equal cost to the value of the LP-relaxation. Here double-nested means that there are two jobs j and k that are partially assigned to four intervals completing at times t_1, t_2, t_3, and t_4, with $t_1 < t_2 < t_3 < t_4$, such that one of the jobs is assigned to t_1 and t_3 and the other one to t_2 and t_4. For our problem, we can show that the result by Verma and Dessouky can be applied. Hence, we solve the LP-relaxation. If the double-nested solution is sub-optimal, then the solution to the LP will be integral. If the solution to the LP-relaxation is fractional, then we know that there exists an integral solution with equal value, but we still have to find it. The open question is whether there exists an easy algorithm to find it. An early version of the paper can be found at

`http://www.cs.uu.nl/research/techreps/UU-CS-2005-054.html`

The final version has been accepted by Journal of Scheduling.

Open Problem

Proposed by Claire Mathieu

Along with Moses Charikar and Howard Karloff, I worked unsuccessfully on the following conjecture. Consider the problem $PM|p_j = 1, prec|C_{max}$ of scheduling unit-time single-processor jobs on M identical processors to minimize the makespan, when there are precedence constraints (so that the input is simply a DAG describing the precedence constraints.)

Conjecture 1 *Consider $PM|p_j = 1, prec|C_{max}$. Fix M and ϵ. Then there exists $k = k(M, \epsilon)$ such that the linear program below (whose size is polynomial in n^k) has integrality gap less than $1 + \epsilon$.*

Here is the linear program for checking feasibility of T. There is one variable x_p for each partial assignment of $\ell \leq k$ slots to jobs:

$$\{(j_1, t_1, m_1), (j_2, t_2, m_2), \ldots, (j_\ell, t_\ell, m_\ell)\}$$

where $t_i \leq T$ is a timestep, $m_i \leq M$ is a machine, and j_i is either a job or "idle".

Feasibility constraints: if p is not feasible then $x_p = 0$. (This can happen either because the same job is assigned to two different slots, or because the same slot is assigned two different jobs, or because job j is scheduled before or at the same time as job j' even though j' must precede j according to the input precedence constraints.)

By convention we define a special variable $x_\emptyset = 1$.

Covering constraints: Given a partial assignment p of $\ell < k$ slots, for every job j, the extension of p schedules j somewhere:

$$\sum_{(t,m)} x_{p \cup \{(j,t,m)\}} = x_p.$$

Packing constraints: Given a partial assignment p of $\ell < k$ slots, for every slot (t, m), the extension of p schedules exactly one job (including the "idle" possibility) in the slot:

$$\sum_{j} x_{p \cup \{(j,t,m)\}} = x_p.$$

And of course, every x_p must be in $[0, 1]$.

Remark: It can be verified that this is an encoded form of the Sherali-Adams lifting of the usual LP relaxation of the problem. Moreover, the LP can be enriched if desired by adding positive semi-definite constraints; in particular, the matrix that has one row for every partial assignment p of $\leq k/2$ slots, one column for every partial assignment q of $\leq k/2$ slots, and (p, q) entry equal to $x_{p \cup q}$, must be positive semi-definite.

Open Problem

Proposed by Nicole Megow

Problem Statement: Jobs arrive online over time at their release dates. At a job's arrival, its deadline and processing time are revealed. We are interested in online algorithms that schedule these jobs on m parallel identical machines with arbitrary preemption and migration. Does there exist an online algorithm for m speed-s machines, with $s < 2 - 1/m$, that finds a feasible solution for any instance for which there exists a feasible schedule on m speed-1 machines. This is an open question that arises from

[PSTW] Cynthia A. Phillips, Clifford Stein, Eric Torng, Joel Wein: Optimal Time-Critical Scheduling via Resource Augmentation. Algorithmica 32(2): 163-200, 2002.

Related Results and Comments:

- Speed $s = 1$: EDF and LLF are well-known to guarantee a feasible schedule on one machine, if such a feasible schedule exists. On two or more machines, no online algorithm can guarantee a feasible schedule, if such a schedule exists. (See references in [PSTW].)

- Lower bound: [PSTW] show that there is no s-speed online algorithm with $s < 6/5$.
- Upper bounds: [PSTW] show that LLF and EDF are $2-1/m$-speed algorithms. This result is tight for EDF; for LLF we do not know. With S. Anand (IIT Delhi) we can show that LLF needs speed

$$s \geq \frac{1+\sqrt{1+4x2}}{2x}, \text{ with } x = \frac{m}{m-1}.$$

This value is $(1+\sqrt{17})/4 \approx 1.28$ for $m = 2$ and goes up to the Golden ratio $\Phi = (1+\sqrt{5})/2 \approx 1.618$ for $m \to \infty$.

Maximizing throughput in real-time scheduling

Proposed by Seffi Naor

Consider the following real-time scheduling problem in which the goal is to maximize the throughput. The input to the problem consists of n jobs, where each of the jobs is associated with a release time, a deadline, a weight, and a processing time. The goal is to find a non-preemptive schedule that maximizes the weight of the jobs meeting their deadline. Garey and Johnson have shown that deciding whether a given set of jobs can all be scheduled is already NP-hard in the strong sense.

Let us focus on the discrete version of the problem. In this version the possible instances of a job are given explicitly as a set of time intervals. The goal then is to pick a set of maximum weight non-intersecting time intervals such that at most one interval from each set of job instances is picked. Spieksma (On the approximability of an interval scheduling problem, Journal of Scheduling, Vol. 2, pp. 215-227, 1999) considered the unweighted version of the interval scheduling problem and proved that it is MAX-SNP hard.

There is a natural linear programming formulation for this problem. Each time interval of a job is associated with a variable and there are two types of constraints: the *time* constraints require that at any time at most one interval is scheduled, and the *job* constraints require that at most one interval from each job is chosen. It is easy to show that the integrality gap of this linear program is at least 2. Bar-Noy et al. (A. Bar-Noy et al., Approximating the throughput of multiple machines in real-time scheduling, SIAM Journal on Computing, Vol. 31, pp. 331–352, and A. Bar-Noy et al., A unified approach to approximating resource allocation and scheduling, Journal of the ACM, Vol. 48, pp. 1069-1090) gave a 2-approximation algorithm for the problem based on this linear programming formulation.

The open question is whether this approximation factor can be improved. A step in this direction was taken by Chuzhoy et al. (J. Chuzhoy et al., Approximation algorithms for the job interval selection problem and related scheduling problems, Mathematics of Operations Research, Vol. 31, pp. 730-738) who gave an approximation algorithm achieving a factor of $e/(e-1)+\varepsilon$, for all $\varepsilon > 0$, for the case where all jobs have the same weight. However, the question still remains open for weighted instances. It would be very challenging to add more constraints to the above linear programming formulation to decrease the integrality gap.

Open Problem

Proposed by Kirk Pruhs

Problem Statement : Jobs with release dates, deadlines and known sizes arrive over time. The problem is to determine if there is an online algorithm that can schedule these jobs, with arbitrary preemption and migration, on $O(m)$ machines, if there is a feasible schedule on m machines. To the best of my knowledge this open problem arises from the paper: Cynthia A. Phillips, Clifford Stein, Eric Torng, Joel Wein (PSTW): Optimal Time-Critical Scheduling via Resource Augmentation. STOC 1997.

Related Results and Comments:

- No Augmentation: EDF and LLF will guarantee a feasible schedule on one machine, if such a feasible schedule exists. There is no online algorithm that will guarantee a feasible schedule on two machines (or more machines), if such a schedule exists. So you need some sort of augmentation for more than one machine.

- Speed Augmentation: PSTW shows that LLF and EDF are 2-speed algorithms for this problem, that is, they guarantee an optimal schedule on m machines if there is a feasible schedule on m unit speed machines. In the SODA 2006 paper "Extra unit-speed machines are almost as powerful as speedy machines for competitive flow time scheduling" H. L. Chan, T. W. Lam, and K. S. Liu showed that there is a $(1+\epsilon)$-speed $O(1/\epsilon 2)$-machine algorithm.

- Machine Augmentation: PSTW showed that there is no $(1+\epsilon)$-machine algorithm when $\epsilon < 1/4$. PSTW also showed that none of the standard algorithms, e.g. LLF and EDF, are $O(f(m))$-machine algorithms for any function f.

- Comments: To my knowledge, not even an $O(f(m))$ machine algorithm is known for any function f; although it is not clear how interesting this would be if f is $\omega(\log m)$. The core issue seems to be that machine augmentation doesn't excuse the online algorithm from having to understanding the nesting structure of the jobs in the optimal schedule (as does speed augmentation).

Parallel machines scheduling to preemptively minimize the weighted sum of mean busy dates

Proposed by Maurice Queyranne

Problem Statement: What is the complexity of the scheduling problem $P|pmtn|\sum_j w_j M_j$ of preemptively minimizing a weighted sum of mean busy dates on identical parallel machines?

Instance input data are:

- a set $N = \{1, \ldots, n\}$ of n jobs, all available at date 0 and with given processing times $p_j > 0$ and weights $w_j > 0$ $(j \in N)$;

- m identical parallel machines.

Recall some definitions: for a given schedule and every job $j \in N$:

- At any date $t \in \mathbb{R}$ $(t \geq 0)$, the (actual processing) *speed* $\sigma_j(t)$ of job j at date t is $\sigma_j(t) = 1$ if j is being processed at date t, and 0 otherwise. If the schedule is feasible then
  - it entirely processes job j: $\int_{t=0}^{+\infty} \sigma_j(t)\, dt = p_j$; and
  - every machine can process at most one job at every date $t \geq 0$: $\sum_{j \in N} \sigma_j(t) \leq m$.
- The *mean busy date* M_j of job j is the average date at which j is being processed in the schedule, i.e., $M_j = \int_{t=0}^{+\infty} t\, \sigma_j(t)\, dt$.

Related Results and Comments:

1. Mean busy dates were introduced by Michel Goemans (SODA 1997) and are commonly used to define relaxations of certain (usually nonpreemptive) scheduling problems, and derive structural and/or analyze approximation results (e.g., Goemans et al., SIAM J. Disc. Math. 2002; Chou et al., Math. Prog. 2006). They are also of independent interest in scheduling problems where jobs accrue costs (or revenue), continuously over time, while they are being processed.

2. An optimal schedule for our problem $P|pmtn|\sum_j w_j M_j$ may use "strategic" preemptions, which occur at dates that may vary (continuously) not only as the "physical" data (processing times, number of machines) vary , but also as the "economic" data (the weights) vary.
   Example: $m = 2$ machines; $n = 3$ unit jobs (all $p_j = 1$) with weights $w_1 = 11$, $w_2 = 10$ and $w_3 = 9$. The unique (up to swapping the two machines) optimum schedule is to process job 1 between dates 0 and 1 on the first machine; job 2 between 0 and 0.55 on the second machine, and between 1 and 1.45 on the first machine; and job 3 between 0.55 and 1.55 on the second machine; for an optimum objective value of 21.975. The optimum preemption date (currently, 0.55) of job 2 varies continuously as the weights w_j vary by small nonzero amounts around their initial values (11, 10, 9).

3. The same problem, but with a *fixed* number m of machines, i.e., problem $Pm|pmtn|\sum_j w_j M_j$, can be solved in polynomial time (for rational data), but with running time growing exponentially with m. This follows from the fact (Queyranne, manuscript notes, July 20, 2010) that this problem (and more general versions with unrelated machines and time-dependent processing times $p_{ij}(t)$ given as piecewise constant functions; note that this allows modeling release dates, deadlines, planned job and/or machine unavailability periods, etc.) can be formulated and solved as a convex quadratic programming problem (with decision variables associated with subsets of at most m jobs forming "feasible patterns" of jobs).

References

[1] Chou, C.-F., Queyranne, M., and Simchi-Levi D.: The asymptotic performance ratio of an on-line algorithm for uniform parallel scheduling with release dates. Mathematical Programming 106, 137–157 (2006).

[2] Goemans, M. X.: Improved Approximation Algorithms for Scheduling with Release Dates. Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms, 591-598 (1997).

[3] Goemans, M. X., Queyranne, M., Schulz, A. S., Skutella, M., and Wang,Y.: Single Machine Scheduling with Release Dates. SIAM J. Discrete Mathematics 15, 165-192 (2002).

[4] Queyranne, M.: Manuscript notes. July 20, 2010.

Open Problem

Proposed by Adi Rosén

We consider directed linear communication networks. The linear network consists of n nodes $\{1, \ldots, n\}$, and $n-1$ directed edges, $(i, i+1)$, for $1 \leq i \leq n-1$. The system is synchronous, and in each time step each edge can transmit one message. Each node can store at any time an infinite number of messages. We are given a set $\mathcal{M}$, $|\mathcal{M}| = M$ of messages. Each message $m = (s_m, t_m, r_m, d_m) \in \mathcal{M}$ consists of a *source node* s_m, a *target node* t_m, a release time r_m, and a deadline d_m. For a message m, we define the *slack* of m, σ_m, to be $\sigma_m = (d_m - r_m) - (t_m - s_m)$ (this is the number of steps the message can be idle and still make it to its destination by its deadline.). We define $\Sigma = \max_{m \in \mathcal{M}} \sigma_m$.

We want to find a schedule for the messages that maximizes the number of messages that arrive to their respective destinations by their respective deadlines.

The open problem is whether there exists a polynomial-time algorithm with constant approximation ratio.

The problem is NP-hard [2]. A polynomial-time algorithm with approximation ratio $O(\min\{\log^* n, \log^* \Sigma, \log^* M\})$ is known [3].

References

[1] Micah Adler, Sanjeev Khanna, Rajmohan Rajaraman, and Adi Rosén. Time-constrained scheduling of weighted packets on trees and meshes. *Algorithmica*, 36(2):123–152, 2003.

[2] Micah Adler, Arnold L. Rosenberg, Ramesh K. Sitaraman, and Walter Unger. Scheduling time-constrained communication in linear networks. *Theory of Computing Systems*, 35(6):599–623, 2002.

[3] H. Räcke, A. Rosén, Approximation Algorithms for Time-Constrained Scheduling on Line Networks. In *Proc. of the 21st ACM Symposium on Parallel Algorithms and Architectures* (SPAA), pp. 337–346, August 2009.

Non-clairvoyant scheduling with precedence constraints: *Towards a measure of the worst case degree of parallelism within a precedence constraints DAG structure*

Proposed by Nicolas Schabanel

This looks technical, but it isn't as much as it looks ;-).

The model. We consider the non-clairvoyant preemptive setting by Edmonds [1] augmented with precedence constraints as defined in [5]: each job J_i consists in an unknown DAG of unknown subjobs J_{ij}. Each subjob J_{ij} consists in an unknown sequence of unknown phases J_{ij}^k. Each phase consists in certain amount of work w_{ij}^k and a speed-up function Γ_{ij}^k (sublinear and non-decreasing). The amount of work accomplished during dt by subjob J_{ij} during phase k when given $\rho_{ij} \in \mathbb{R}_+$ processors is: $dw_{ij}^k = \Gamma_{ij}^k(\rho_{ij})\,dt$. We say that a subjob in phase k progresses at rate $\Gamma_{ij}^k(\rho_{ij})$ when given ρ_{ij} processors. A subjob is released as soon as all of its predecessors are completed, or when its corresponding job is released, if it has no predecessors. A subjob completes when all of its phases are completed. A job is completed when all of its subjobs are completed.

Non-clairvoyant algorithms. We consider non-clairvoyant algorithms, that is to say algorithms that are unaware of the DAG structure within each job, nor of the phases etc. Non-clairvoyant algorithms cannot forecast the structure of the DAG and only discover the subjobs at the time of their release. The only available informations to the algorithm are the release dates of each jobs and subjobs and to which job belongs each currently active subjobs.

Previous results. When the DAG consists in a unique chain of subjobs, Edmonds and Pruhs have shown in [2] that the algorithm LAPS_β that shares equally the processors between the fraction β of the most recent active jobs, is $1+\beta+\epsilon$-speed $4(1+\beta+\epsilon)/\beta\epsilon$-competitive. The first step of the proof in [1, 2] consists in showing that one can only consider two types of phases: SEQ, for which $\Gamma(\rho) = 1$ for all $\rho \geq 0$ and on which any processor given is wasted; and PAR, for which $\Gamma(\rho) = \rho$ for all $\rho \geq 0$, on which any processor given is used at 100%. Any non-clairvoyant algorithm which is competitive for SEQ/PAR jobs, is competitive for arbitrary sublinear non-decreasing speed-up functions. We have shown in [5] that this reduction to SEQ/PAR subjobs by [1] still applies in the setting with precedence constraints.

Scattering coefficient. This coefficient is the key in [5] to obtain our upper bound on the competitive ratio of non-clairvoyant schedulers in presence of precedence constraints, because it allows to get rid of the precedence constraint without modifying the supported load.

We consider $A \circ B$ algorithms where algorithm A allots some processors ρ_i to each alive job J_i and algorithm B splits among the alive subjobs J_{ij} of each given job J_i the ρ_i processors it received from A. New difficulties arise from precedence constraints because the way that the algorithm unfolds the DAG, has a huge impact on the number of processors wasted on SEQ phases (see [5, 4]). We bound from above the extend of this waste as follows. Given a DAG structure D, the scattering coefficient $\alpha(B, D)$ for algorithm B on

DAG D, is the maximum of the following ratio overall possible job J_i with DAG structure D and all allocation of processors ρ_i:

$$\alpha(B,D) = \max_{\substack{\text{all functions } \rho_i, \\ \text{all jobs } J_i \text{ with DAG } D}} \frac{\int_0^\infty \big(\% \text{ procesors wasted by } B \text{ on SEQ phases of subjobs of } J_i\big)\, dt}{\text{SEQ}(J_i)}$$

where $\text{SEQ}(J_i)$ denotes the maximum amount of SEQ work along a chain of subjobs of J_i. Surprisingly, because it is a worst case coefficient, this scattering coefficient can be easily computed for various precedence constraints structures [5] such as: independent chains, in-trees, out-trees, serial-parallel DAG,... (for instance by using dynamic programming).

Now, since, by definition of $\alpha(B,D)$, the sequential work in any DAG cannot be stretched by a factor more than $\alpha(B,D)$ in any DAG D, and since the progress in parallel work is not impacted by the precedence constraints, the competitive ratio of algorithm $A \circ B$ on D at speed s is at most α times the competitive ratio of A at speed s on jobs consisting in a unique chain of subjobs (as in [1, 2]). It follows that $\text{LAPS}_\beta \circ \text{EQUI}$ is $(1+\beta+\epsilon)$-speed $(k+1)(1+\beta+\epsilon)/\beta\epsilon$-competitive where k is the size of the largest number of independent jobs in a DAG of the instance, because EQUI is a $(k+1)/2$-scatterer, which is optimal (see [5]). A surprising consequence is that precedence constraints *do not decrease* the maximum supported load by non-clairvoyant algorithms!

Open questions. The scattering coefficient thus yields upper bounds on the competitive ratio of non-clairvoyant scheduler with precedence constraints. A natural question is thus: is it the right parameter to measure the degree of parallelism in a precedence constraints structure? In other words, can the scattering coefficient yield lower bounds on the best achievable competitive ratio by a given algorithm on a given DAG? Is there a way to define this coefficient independently of the algorithm B used? We need to be able to construct worst case instances for a given DAG structure from the worst value of the scattering coefficient. We do not know how to do that yet, nor if it is possible in general. If true, this would yield an interesting measure of the worst degree of parallelism achievable in a precedence constraints structure and thus may help designing dependancies between softwares so as to minimize the risk of wasting performances due to bad precedence constraints between them.

A preliminary result we have in that direction, is that for trees D, one can construct worst case instances on a DAG structure D' close to D with competitive ratio $\Omega(\alpha(D))$. D' is simply an "hairy" version of D where each leaf is prolongated by q leaves [5, 3].

References

[1] Jeff Edmonds. Scheduling in the dark. In *Proc. of ACM Symp. on Theory of Computing (STOC)*, pages 179–188, 1999.

[2] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *Proc. of ACM/SIAM Symp. On Discrete Algorithms (SODA)*, pages 685–692, 2009.

[3] Julien Robert. *Ordonnancement non-clairvoyant avec contraintes de dépendances.* PhD thesis, École normale supérieure de Lyon, Dec. 2009.

[4] Julien Robert and Nicolas Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *Proc. of European Symp. on Algorithms (ESA)*, volume LNCS 4698, pages 742–753, Oct. 2007.

[5] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Proc. of ACM/SIAM Symp. On Discrete Algorithms (SODA)*, pages 491–500, Jan. 2008.

Maximizing the throughput of parallel jobs on two machines

Proposed by Jiří Sgall

$P2|r_j, p_j = 1, size_j| \sum U_j$, i.e., maximizing he throughput of parallel jobs on two machines.

Setting: We are given two machines and a set of unit processing time jobs with release times and deadlines. Each job can be either parallel, then it needs to be run on both machine in the same single timeslot, or sequential, then it needs to be run on one arbitrary machine for a single timeslot. The objective is to maximize the number of jobs that can be scheduled in their respective time windows.

Open problem: Can this be solved in polynomial time? What about an approximation?

Related results and notes:

Feasibility: Checking if all the jobs can be scheduled (and finding the schedule) can be done in polynomial time. See the following papers. In the first one, the dynamic programming proof is correct, the other has a correctable gap. The other paper has a shorter proof using total unimodularity.

Ph. Baptiste, B. Schieber. A Note on Scheduling Tall/Small Multiprocessor Tasks with Unit Processing Time to Minimize Maximum Tardiness. Journal of Scheduling 6(4): 395-404, 2003.

C. Dürr and M. Hurand. Finding total unimodularity in optimization problems solved by linear programs. In Proc. 13th European Symp. on Algorithms (ESA), LNCS 4168, pages 53-64. Springer, 2006.

Special cases: It is possible to solve various special cases, e.g. if the jobs are nested or the parallel jobs have agreeable deadlines and release times. See:

O. Zajíček. A note on scheduling parallel unit jobs on hypercubes. Int. J. on Found. Comput. Sci., 20(2):341-349, 2009.

Tamás Kis. Scheduling multiprocessor UET tasks of two sizes. Theor. Comput. Sci. 410(47-49):4864-4873, 2009.

Approximation: We have a 1.5-competitive algorithm. See:

O. Zajíček, J. Sgall, T. Ebenlendr: Online scheduling of parallel jobs on hypercubes: Maximizing the throughput To appear in Proc. of the Parallel Processing and Applied Mathematics (PPAM'09), Lecture Notes in Comput. Sci., Springer, 2010.

`http://iti.mff.cuni.cz/series/files/2009/iti481.pdf`

No better approximation seems to be known, even offline.

Other notes: For sequential jobs only the problem is an easy matching problem. If we know which pairs of sequential jobs are scheduled in the same timeslot, it is easy as well. We may assume that both the optimum and any algorithm schedules all the sequential jobs.

2-approximation for $1|r_j, prec|\sum w_j C_j$

Proposed by René Sitters

Problem Statement: Is there a polynomial time 2-approximation for minimizing total weighted completion time on the single machine with release dates and arbitrary precedence constraints ($1|r_j, prec|\sum w_j C_j$)?

Related Results

- An e-approximation is given by Schulz and Skutella (Paper: Random based scheduling, 1997).
- For the case that all release dates are zero several 2-approximation algorithms are known. It is generally believed that this is best possible in polynomial time.

Conjecture: Rounding an optimal solution of the following LP gives a 2-approximation. Define a variable x_{jt} for each job j and time t.

$$\begin{array}{ll} \min & \sum_j w_j(M_j + p_j/2) \\ s.t. & M_j = \sum_t x_{jt}(t + p_j/2) \\ & \sum_j \sum_{t':t-p_j+1 \le t' \le t} x_{jt'} \le 1, \text{ for all } t \textit{ (packing constr.)} \\ & \sum_{t' \le t-p_j} x_{jt'} \ge \sum_{t' \le t} x_{kt} \text{ for all } t \text{ and } j \prec k \textit{ (prec. constr.)} \\ & x_{jt} \ge 0 \text{ for all } j, t \end{array}$$

First part of the conjecture is that for a given optimal LP-solution there always is solution such each job completes within time $2M_j^{LP} + p_j$. (This is not true for the LP using variables y_{jt}, where y_{jt} is the fraction of j processed between t and $t+1$). Second part of the conjecture is that we can find it in polynomial time.

Remarks: - The LP-is only of polynomial size of input numbers are polynomially bounded. Hence, in general this should give a $2+\epsilon$-approximation.

Increasing speed scheduling

Proposed by Sebastian Stiller

Given an n tuple of pairs of natural numbers $(d, w)_{1 \le i \le n}$ and a weakly monotonically increasing, integrable function $s : \mathbb{R}^+ \to \mathbb{R}^+$. The Increasing Speed Scheduling (ISS) problem asks for a permutation σ, such that $\sum w_i C_i$ is minimal with

$$C_i := \min \left\{ t \in \mathbb{R}^+ : \int_0^t s(x)dx \ge \sum_{\sigma(j) \le \sigma(i)} d_j \right\}.$$

We conceive of this as n jobs with demand and weight to be scheduled on a single machine with increasing speed.

We are interested in the complexity of the problem, in particular for s being piecewise constant, even with a constant number of non-differentiable points. Among other positive results, a (straight forward) dynamic program is known for the latter case. Still, even for s taking exactly two distinct values we conjecture weak NP-hardness.

For details, background, and some of our current results on the matter please confer `http://www.math.tu-berlin.de/coga/publications/techreports/2010/Report-007-2010.xhtml`.

Deliberate idleness problem

Proposed by Marc Uetz

The *deliberate idleness problem* is a problem in stochastic machine scheduling. In stochastic machine scheduling, we are concerned with the question how to optimally schedule n jobs with stochastic processing requirements on m machines. More specifically, the processing times follows distributions $p_j \sim X_j$, $j = 1 \ldots, n$. The jobs are nonpreemptive, all available at time 0, and have to be scheduled on m parallel, identical machines. Each machine can only do one job at a time, and a job can go on any of the m machines. Moreover, each job has a weight w_j, and we want to find a scheduling policy that minimizes the expected value of the weighted sum of completion times, $E[\sum_j w_j C_j]$. An instance consists of the input of jobs (w_j, X_j), $j = 1, \ldots, n$, and the encoding of the number of machines m.

Noticeable about stochastic scheduling is that the solution is not a schedule, but a scheduling *policy* which essentially tells us, at any point in time t (typically when a machine falls idle, but possibly also at other points in time), which job(s) to schedule next. This decision may depend on the input of the problem, and the state of the system at time t. The latter is given by time t, the set of jobs already completed, the set of jobs currently running together with their conditional distribution of remaining processing time, and the set of jobs not yet started.

The **question is this**. Assume $m \geq 3$ machines, and assume that all jobs follow an exponential distribution, $p_j \sim exp(\lambda_j)$, that is, the processing times are memoryless. Does there always exist an optimal policy that avoids deliberate idleness? (That is, as long as there are unprocessed jobs, it would never leave a machine idle.)

Some background information.

- For arbitrary distributions $p_j \sim X_j$ there are simple examples showing that deliberate idleness can be necessary, even on $m = 2$ machines. See

  M. Uetz, When Greediness Fails: Examples from Stochastic Scheduling, Operations Research Letters 31, 2003, pp. 413-419.
- For $m = 2$ machines and $p_j \sim exp(\lambda_j)$ an optimal policy always exists that avoids deliberate idleness. This is not totally trivial, but not too difficult either, using an inductive argument.
- The WSEPT rule (greedily schedule jobs in order of ratios w_j/Ep_j) has a performance guarantee of $2 - 1/m$, for any distributions with coefficient of variation at most 1. Thus, in particular for exponential distributions. See

  R.H. Möhring, A.S. Schulz, M. Uetz, Approximation in stochastic scheduling: the power of LP-based priority policies, Journal of the ACM 46 (1999), 924-942.
- For all $w_j = 1$ the problem is solved optimally by the SEPT rule, greedily schedule jobs with shortest expected processing time first.

Complexity of local search

Proposed by Tjark Vredeveld

Problem Statement: We consider the problem of finding a local optimum for the following problem. Given are n jobs, with processing times $p_1, \ldots, p_n$, each of which needs to be scheduled on one of m identical parallel machines. The goal is to schedule the jobs in such a way that the *makespan* is minimized.

The class PLS (polynomial-time local search) [6] contains the (local search) problems whose neighborhood can be search in polynomial time. Several important local search problems are complete for PLS under an appropriately defined reduction, see e.g. [8].

The simplest form of an local search algorithm is *iterative improvement*: starting from a feasible solution, move from one solution to an neighboring solution that is better. Iterative improvement stops in the first local optimum that its encounters.

Consider the k-Opt neighborhood, i.e., we are allowed to relocate k jobs. For which k can we prove that we can find a local optimum by iterative improvement in a polynomial number of steps and for which k can we show that it is PLS-complete?

Another question is whether the Push neighborhood, defined in [7], is PLS-complete.

Related Results and Comments:

- Recently, Dumrauf, Monien, and Tiemann [4] showed that the scheduling problem is PLS-complete for a neighborhood in which 33 jobs are allowed to be relocated.
- On the other hand, Brucker, Hurink, and Werner [1, 2] showed that if the neighborhood consists of all schedules that can be obtained by relocating only one job (jump neighborhood), then a local optimum can be found in $\mathcal{O}(n^2)$ steps by iterative improvement, if always a job is jumped to a machine with minimum load.
- The Push neighborhood is a variable depth search neighborhood. To define it unambigously, one also needs to define a machine selection rule, for a job that needs to be relocated. See [7] for more details. Therefore, the complexity of the push neighborhood also depends on this rule.
- Performance guarantees for the quality of local optima for this scheduling problem are studied in [5, 7, 3].

References

[1] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems I. *Discrete Applied Mathematics*, 65:97–122, 1996.

[2] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems II. *Discrete Applied Mathematics*, 72:47–69, 1997.

[3] T. Brueggemann, J. L. Hurink, T. Vredeveld, and G. J. Woeginger. Very large-scale neighborhoods with performance guarantees for minimizing makespan on parallel machines. In C. Kaklamanis and M. Skutella, editors, *Approximation and Online Algorithms (WAOA 2007)*, volume 2909 of *Lecture Notes in Computer Science*, pages 41–55. Springer, Berlin, 2008.

[4] D. Dumrauf, B. Monien, and K. Tiemann. Multiprocessor scheduling is PLS-complete. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–10, 2009.

[5] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1979.

[6] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[7] P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. *Informs Journal on Computing*, 19:52–63, 2007.

[8] M. Yannakakis. Computational complexity. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 2, pages 19–55. Wiley, Chichester, 1997.

The airplane refueling problem

Proposed by Gerhard J. Woeginger

Problem Statement: Suppose you have to deliver a bomb in some distant point of the globe, the distance being much greater than the range of the airplanes you are going to use. Thus you have to use the technique of refueling in the air. Starting with several planes which refuel one another, and gradually drop out of the flight until the last plane reaches the target, how would you plan the refueling program?

Here is a mathematical formulation of this problem. There are n airplanes $A_1, \ldots, A_n$ with tank volumes v_j and gas consumption rates c_j ($1 \le j \le n$). The goal is to find a drop out permutation $\pi(1), \ldots, \pi(n)$ for the planes that maximizes the distance traveled by the last plane. Since the tank volume v_j of airplane $A_{\pi(j)}$ is consumed by airplanes $A_{\pi(j)}, \ldots, A_{\pi(n)}$ the traveled distance of the last plane is given by

$$\sum_{j=1}^{n} \left(v_{\pi(j)} \ / \ \sum_{k=j}^{n} c_{\pi(k)} \right).$$

The cases where all planes have identical tank volumes and where all planes have identical consumption rates can be solved by sorting. The computational complexity of the general case is open.

Related Results and Comments: This problem is motivated by a problem in the aeronautica chapter of the book *"Puzzle-Math"* (Macmillan, London, 1958) by George Gamow and Marvin Stern.