DESIGNING PTAS FOR MIN-SUM SCHEDULING PROBLEMS

Foto Afrati
National Technical University of Athens

The Scheduling Problem:

Given:

-Set T of n tasks with associated processing times

-Precedence constraints among T (dag)

-Set P of m processors

Goal:

Assign tasks to processors and time slots s.t. minimize an objective function (eg. maximum completion time ≡ makespan, sum of completion times)

Example

Given:

$T = \{T_1, T_2, T_3, T_4, T_5\}$

processing times: 3, 1, 2, 3, 2

$P = \{P_1, P_2\}$

Various Models

⋆ PREEMPTION.
Non-preemptive task: once started has to be processed without interruption.

⋆ PRECEDENCE CONSTRAINTS.
Independent tasks: no precendence constraints.

⋆ MULTIPROCESSOR TASKS.
A task may need to be processed simultaneously by several processors.

Incompatible tasks: tasks that share at least one resource, hence cannot be scheduled in the same time slot.

- ⋆ RELEASE DATES.

- ⋆ WEIGHTS.

- ⋆ DEADLINES.

Three-field notation

$$X \mid Y \mid Z$$

X: machine environment e.g.,

$$1, P, R, R_m$$

Y: additional constraints on jobs e.g.,

r_j (release dates)

pmtn (preemption)

Z: objective function e.g.,

$$\sum \omega_j C_j \qquad \sum C_j$$

Example:

$$1 \mid \mid \sum w_j C_j$$

$$P \mid r_j \mid \sum w_j C_j$$

$$P \mid r_j \mid \sum C_j$$

$$Rm \mid \mid \sum w_j C_j$$

Problem:

Minimize sum of completion times or (equivalently) average (weighed) completion time

That is:

$$min \sum_{alltasks,i} w_i C_i$$

w_i: weight of task i

C_i: completion time of tasks

Known Results:

$P||\sum c_j$:PTIME, by greedy SPT rule

$1||\sum \omega_j c_j$:PTIME, by Smith's rule [Smith 56]

$R||\sum c_j$:PTIME, by matching techniques [BCS74,Horn73]

$1|r_j, pmtn|\sum c_j$:PTIME, by SRPT rule [Baker 74]

$\boldsymbol{P||\sum C_j}$

solved by the **S**hortest **P**rocessing **T**ime rule

old schedule

new schedule (better)

By moving task j to be executed before task i the following changes are done to the total cost: **decreased** by $p_i - p_j$

Thus we get a better schedule!

$\boldsymbol{1||\sum wjC_j}$

solved by **Smith**'s rule : shortest $\frac{P_j}{w_j}$ ratio first.

old schedule

new schedule (better)

By moving task j to be executed before task i the following changes are done to the total cost:

before: $w_i P_i + w_j P_i + w_j P_j$

after: $w_i P_i + w_i P_j + w_j P_j$

decreased by $w_j P_i - w_i P_j$

Thus we get a better schedule!

Constant Factor Approximation Algorithm

For a given constant ρ , they find in PTIME a solution within ρ factor of the optimum.

Polynomial **T**ime **A**pproximation **S**cheme
PTAS

For any fixed ϵ , they find in PTIME a solution within $(1+\epsilon)$ factor of the optimum.

Fully **P**olynomial **T**ime **A**pproximation **S**cheme
FPTAS

In addition to PTAS's definition, the algorithm is polynomial on $1/\epsilon$.

Previous Work

- $Pm \mid \mid \sum w_j C_j$: FPTAS, Shani 1976
- $1|prec|\sum w_j C_j$:
$O(\log n \log \sum_j w_j)$-approximation algorithm,
Ravi, Agrawal and Klein 1991 (improved later)
- $1|r_j, pmtn|\sum w_j C_j$ and $P|r_j, pmtn|\sum w_j C_j$
et al.: first constant approximation algorithms
Philips, Stein and Wein 1995
- Hall, Schulz, Shmoys and Wein 1996 and
1997 improved dramatically
Motivated by work done on polyhedral
structure of scheduling problems
(see Queyranne and Schulz, 1994)

Known Results:

$R|pmtn|\sum C_j$: open whether in PTIME or NP-Hard

$1|r_j|\sum w_jC_j$: 1.69 -approx $\overset{w.w.}{1.58}$ [CMNS 97, GQSSW 99]

$P|r_j, (pmtn)|\sum w_jC_j$: 2-approx. [SS 97]

$Rm|r_j, (pmtn)|\sum w_jC_j$:3-approx [Skutella 98, 99]

$R|r_j\,|\sum C_j$: non-approx. by PTAS unless P=NP

$R||r_j\,|\sum w_jC_j$: non-approx. by PTAS unless P=NP [HSW 98]

Techinnniques and Tools for **const-approx. algorithm**

- First formulate a PTIME-solvable relaxation of the problem.
- Use information from the relaxation to obtain an ordering of the tasks and/or an assignment of tasks to machines.

Have proven efficient in practice!

[Uma—Wein 99, SUW98]

Cannot be used and/or extended to obtain PTASs, [TU, SODA99]

Used in PTAS:

• *time-interval indexed* LP formulation, Hall, Schulz, Shmoys and Wein 1996 and 1997

• Earlier: *time indexed* LP formulation, Potts 1980, Dyer and Wosley 1990, Philips, Stein and Wein 195, Wosley 1985, Queyranne 1993.

• Subdivide the time horizon into the intervals $[1, 1], (1, 1 + \epsilon], (1 + \epsilon, (1 + \epsilon)^2], \ldots,$, where ϵ is an arbitrary small positive constant. The linear program only specifies the interval in which the job is completed.

Since all completion times within an interval are within a $(1 + \epsilon)$ factor of each other, the relative schedule within an interval will be of little consequence. This is what we call here *time partitioning.*

Why earlier methods fail:

• Their solution is not compared directly to the solution of the original problem but to the solution of the relaxed one. However, there are gaps between the solutions of the relaxed and the original problem and any algorithm obtaining such a rounded solution inherits these gaps.

• Hall, Schulz, Shmoys and Wein 1997, Schulz and Skutella 1997 Chekuri and Motwani 1999: $1|prec|\sum w_j C_j$ cannot improve

• Torng and Uthaisombut 1999: $1|r_j|\sum C_j$ cannot improve.

First PTAS: $P \mid \mid \sum M_i^2$ Alon, Azar, Woeginger and Yadid 1998

$P \mid \mid \sum w_j C_j$ Skutella and Woeginger STOC 1999

Method: *ratio partitioning*, i.e. the partitioning of the jobs into subsets according to their $\frac{w_j}{r_j}$ ratios and such that the ratios of all jobs in one subset are within a constant range.

Other PTASs

Based on time ratio partitioning :

• $Rm \mid \mid \sum w_j C_j$ Afrati, Bampis, Kenyon and Milis 2000

• $P2|fix_j| \sum C_j$ $Pm|fix_j, pmtn| \sum C_j$:Afrati, Bampis, Fishkin, Jansen Kenyon 2000

• $Pm|fix_j, r_j pmtn| \sum w_j C_j$: Fishkin Jansen Porkolab 2001

Based on time time partitioning :

• FOCS99: a series of variants with release dates.

Approach:

A series of transformations.

Each transformation:

$\star$ Either simplifies the input or considers an optimal solution and moves the jobs around to get a near-optimal solution with "nice" structure.

$\star$ Increases the objective function value by $1 + O(\epsilon)$.

Therefore, a constant number of transformations increases the objective function value by $1 + O(\epsilon)$.

Outline

- General Transformations.
- Scheduling on a constant number of unrelated machines. (**Ratio Partitioning**)
- Scheduling on a single machine with release dates and unit weights. (**Time Partitioning**)
- Scheduling dedicated tasks on a constant number of machines. (**Time Partitioning**)

[Afrati et al., FOCS99]

[Afrati, Bampis, Kenyon, Milis, 2000

[Afrati, Bampis, Fishkin, Jansen, Kenyon, 2000]

Two major categories of tranformations:

$\star$ **geometric rounding**

$\star$ **time stretching**

- We round processing times and release tasks to integral powers of $(1 + \epsilon)$.

  Thus we limit the number of distinct processing times and release dates.

- We add small amounts of idle time spread throughhout the schedule. If we stretch time by $(1+\epsilon)$ on a given schedule, the increase in the cost is negligible

Thus we use these idle times to "clean up" the schedule.

E.g. if a job is "large" or to the interval in which is scheduled, we can move it to the idle time in a later intervle, where it is "small".

Motivating Example

Given: n identical tasks, each of processing time p.

Then

$\sum_{j=1}^{n} C_j = \sum_{j=1}^{n} jp = \frac{n(n+1)}{2}p$

$max\, C_j = np$

$\frac{max\, C_j}{\sum C_j} = \frac{2}{n+1} \longrightarrow_{n\to\infty} 0$

Therefore some **"moving around tasks"**

is allowed with negligible cost. E.g.

$\star$ Take any job and move it to the end.
Additional cost: $max\, C_j$. Degradation of cost by a factor of: $\frac{2}{n+1}$

$\star$ Take $\log n$ jobs and move them to the end.
Additional cost: $max\, C_j \log n$. Degradation of cost by a factor of: $\log n \frac{2}{n+1}$

Lemma:

With $(1+\epsilon)$ loss, we can assume that all processing times are integer powers of $(1+\epsilon)$.

Proof:

1. Multiply all proc. times by $1+\epsilon$.
2. Round up to the lower integer of $(1+\epsilon)$.

Lemma:

1. With $(1+\epsilon)$ loss, we assume that there is an idle time ϵp_j before the starting time of each job j.
2. With $(1+\epsilon)$ loss, we assume that starting time of job $j\ :\ S_j \geq \epsilon P_j$

Proof:

Multiply every completion time by $(1+\epsilon)$.

Ratio Partitioning

$$R_m || \sum w_j C_j$$

Remark: The single machine version is solved polynomial using Smith's rule.

Motivating Example

Jobs with very different ratios do not affect each other.

Set of jobs: $T = \{T_1, T_2, \ldots, T_n, T_{n+1}\}$
all weights: w, processing times of all jobs except T_{n+1}: ϵp, processing time for T_{n+1}: p.

Cost= $\frac{n(n-1)}{2}\epsilon^2 pw + n\epsilon^2 pw + pw$

Two cases:

1st: $n\epsilon^2 \leq \epsilon$, then, $\frac{n\epsilon^2 pw}{pw} \leq \epsilon$

2nd: $n\epsilon^2 > \epsilon$, then, $\frac{n\epsilon^2 pw}{\frac{n(n-1)}{2}\epsilon^2 pw} = \frac{2}{n-1} < \frac{4}{n} < 4\epsilon$

Job type: the pair of its ratios

$$\left(\frac{w_j}{P_j^{(1)}}, \frac{w_j}{P_j^{(2)}}\right)$$

$$W_{(r,r')} = \sum_{j \in (r,r')} w_j$$

cost function as: $C(\mathcal{S}) = \sum_{j,k \text{ on } M_1} \mathcal{M}_1[j,k] + \sum_{j,k \text{ on } M_2} \mathcal{M}_2[j,k]$, where

$$\forall j,k \in J \qquad \mathcal{M}_i[j,k] = \begin{cases} p_k^{(i)} w_j & \text{if } j = k \\ \frac{1}{2} p_k^{(i)} w_j & \text{if } j \neq k \text{ and } \frac{w_k}{p_k^{(i)}} = \frac{w_j}{p_j^{(i)}} \\ p_k^{(i)} w_j & \text{if } \frac{w_k}{p_k^{(i)}} > \frac{w_j}{p_j^{(i)}} \\ 0 & \text{otherwise} \end{cases}$$

Job types not affecting each other

Lemma:

Whenever $r_j^{(i)} < \epsilon^2 r_k^{(i)}$, we can replace $M_i[j,k]$ by 0, with a $1 + O(\epsilon)$ loss.

Lemma:

Whenever $r_j^{(i)} < \epsilon^4 r_k^{(i)}$, we can replace both $M_1[j,k]$, $M_2[j,k]$ by 0, with a $1 + O(\epsilon)$ loss.

- Either there are constant number of jobs in each job type or we can build compound jobs so that every job of job type (r, r') has weight at least $\epsilon^2 W_{(r,r')}$.

Proof done by time stretching.

Hence, **there are constant number of jobs within each job type.**

$R_2||\sum w_jC_j$

- If $P_j^1 < \epsilon P_j^2$, then job j is M_1 decided

If $P_j^2 < \epsilon P_j^1$, then job j is M_2 decided

Geometric Rounding

All weights and processing times are of the form $(1+\epsilon)^k$.

Windows of ratios:

window $F_i = (a^i R, a^{i-1} R]$

$a = \epsilon^4$

R: the maximum ratio.

- There are constant number of undecided jobs with at least one ratio in window F_i

- Any undecided job has its two ratios either in the same window or in adjacent windows

Dynamic Programming

F_{i-1} $\quad . \quad . \quad . \quad$ F_{i-1}

$assignment_1^{i-1}$ $\quad$ $assignment_j^{i-1}$

choose best

F_i $\quad . \quad . \quad . \quad$ F_i

$assignment_1^{i}$ $\quad$ $assignment_j^{i}$

Algorithm:

1. Change $\mathcal{M}_1[j,k]$, $\mathcal{M}_2[j,k]$ according to the lemmas.

2. Create compound jobs if necessary.

3. Run dynamic programming on ratio windows.

Time Partitioning

- Divide time into geometrically increasing intervals

- In each interval, schedule tasks which ”fit nicely” into this interval

- The total problem is decomposed in a series of subproblems;

each subproblem involves a simpler problem of scheduling in an interval

- Dynamic programming

Motivating Example

Given: n identical tasks, each of processing time p.

Then

$\sum_{j=1}^{n} C_j = \sum_{j=1}^{n} jp = \frac{n(n+1)}{2} p$

$max\, C_j = np$

$\frac{max\, C_j}{\sum C_j} = \frac{2}{n+1} \longrightarrow_{n \to \infty} 0$

Therefore some **"moving around tasks"**

is allowed with negligible cost. E.g.

$\star$ Take any job and move it to the end.
Additional cost: $max\, C_j$. Degradation of cost by a factor of: $\frac{2}{n+1}$

$\star$ Take $\log n$ jobs and move them to the end.
Additional cost: $max\, C_j \log n$. Degradation of cost by a factor of: $\log n \frac{2}{n+1}$

Geometrical time partitioning:

Time intervals: $[0, (1+\epsilon)), [(1+\epsilon), (1+\epsilon)^2), [(1+\epsilon)^2, (1+\epsilon)^3) \ldots ((1+\epsilon)^{x-1}, (1+\epsilon)^x]$

We get $x = log_{1+\epsilon}(np) = O(\log n)$ intervals.

$\star$ Take one job per interval and move it to the end.

Geomewtric Rounding:

Lemma:

With $(1+\epsilon)$ loss, we can assume that all processing times are integer powers of $(1+\epsilon)$.

Proof:

1. Multiply all proc. times by $1+\epsilon$.

2. Round up to the lower integer of $(1+\epsilon)$.

Time Stretching:

Lemma:

1. With $(1+\epsilon)$ loss, we assume that there is an idle time ϵp_j before the starting time of each job j.

2. With $(1+\epsilon)$ loss, we assume that starting time of job j : $S_j \geq \epsilon P_j$

Proof:

Multiply every completion time by $(1+\epsilon)$.

Lemma:

Each job crosses at most $s := \lceil log_{1+\epsilon}(1+\frac{1}{\epsilon})\rceil$ intervals.

Proof:

The worst case is a job stretching from time ϵp_j to time $\epsilon p_j + p_j$. This is at most s intervals (i.e. const. no. of intervals)

Lemma:

With $(1+\epsilon)$ loss, we can arbitrary re-arrange jobs within an interval.

Proof:

$I_x = \sum_{all\, tasks\, completing\, in\, I_x} C_j \; > nt$ by re-arranging, we change C_{I_x} by at most $nI_x = \epsilon tn$

$$\frac{\Delta C_{I_x}}{C_{I_x}} < \frac{\epsilon tn}{tn} = \epsilon$$

- Divide time into geometrically increasing intervals

- In each interval, schedule tasks which "fit nicely" into this interval

- The total problem is decomposed in a series of subproblems;

each subproblem involves a simpler problem of scheduling in an interval

- Dynamic programming

$$1|r_j|\sum C_j$$

Remark: Preemptive version (i.e. $1|r_j, pmpn|\sum C_j$)

can be solved polynomially using the

SRPT rule

Algorithm for $1|r_j|\sum C_j$

1. Reassign release date to jobs:

   new release date $= \max\{r_j, P_j/\epsilon^2\}$

2. Run SPT until at most $3/\epsilon^7$ jobs are left

3. Enumerate all orderings of the remaining jobs to find the best one.

Computational Complexity:
$O(nlogn + (3/\epsilon^7)!)$

Lemma:

If, in the optimum schedule, all jobs are small

$$(P_j \leq \epsilon\, I_x)$$

then the algorithm gives a $(1+\epsilon)\, \times\, OPT$ solution.

Proof

$$s_j \geq R_x = \frac{I_x}{\epsilon} \geq \frac{P_j}{\epsilon^2}$$

Hence, $r_j \leq \frac{P_j}{\epsilon^2}$ does not change the feasibility of the opt. schedule

Create a preemptive opt. schedule (SPRT rule)

!!Jobs are small when released!!

Complete pre-empted jobs by $(1+\epsilon)$ time stretching

Corollary:

We have at most one preemption per interval

Lemma:

With $(1+\epsilon)$ loss, almost all jobs - except constantly many - have starting time $> \frac{1}{\epsilon^2}\,OPT$

Proof: At most $\frac{1}{\epsilon^7}$ jobs have completion time $> \epsilon^7\,OPT$

For the rest, it holds: $S_j > \frac{1}{\epsilon^2} P_j$

If it doesn't, move the job at the gap at time $\frac{C_j}{\epsilon^2}$

Two considerations:

a) The gap is large enough to accomodate all jobs landed there

b) The total added cost of the moved jobs is negligible

Correctness of the Algorithm

– There is a near-optimal schedule where almost all jobs are small within their intervals.

–By applying SRPT rule, we get at most one preemption per interval.

Scheduling of Independent multiprocessor tasks on dedicated processors

$\boldsymbol{2|multiprocessor|\sum C_j}$

We have single machine tasks or 1-tasks and 2 tasks

and 2-machine tasks or 12-tasks

Preemptive case

- this picture

  never happens

- Jobs of each type obey the SPT rule.

Hence, we have a linear number of positions to put the 12-jobs

in PTIME

n_j: # (tasks completing after time t)

Additional cost: $p_j n_j + t + p_j$

$$min_t\{p_j n_j + t + p_j\}$$

Computational Complexity

$T(n) = 2T(n/2) + O(n)$

$T(n) = O(n\,logn)$

$S_{12-tasks} =$
$\{T_1, \quad T_2, \quad T_3, \quad T_4, \quad T_5, \quad T_6, \quad T_7\}$

$$P_1 \leq P_2 \leq P_3 \leq P_4 \leq P_5 \leq P_6 \leq P_7$$

PTAS for preemptive case, with 2 processors

1. Schedule uniprocessor tasks with SPT rule

2. Schedule the $\frac{1}{\epsilon}$ longest biprocessor tasks by complete enumeration

3. For each task, j, of the rest biprocessor tasks, schedule j at time t, such that

$$\frac{1}{P_r} + \frac{1}{P_k} \leq \frac{1}{P_j}$$

Suppose $\forall j,\ P_j \leq \epsilon^7\, OPT$

Algorithm:

1. Construct a $(1+\epsilon)$-optimal preemptive schedule.

2. Move large uniprocessor tasks in long intervals.

3. Finish each preempted task at the time of preemption.

Lemma.

Supposse there is a partition of tasks
$T = L \bigcup S$ s.t.

$$max_{j \in S} \, P_j < \epsilon^5 \, min_{j \in L} \, P_j$$

Then, S is scheduled before L.

Lemma.

(there is a gap) With $1 + O(\epsilon)$ loss, we can partition

$T = L \bigcup S$ s.t. S is scheduled before L; and $|L| = O(1)$

ALGORITHM

1. Find a gap; $\quad T = L \bigcup S$

2. Find a $(1+\epsilon)$-optimal schedule for S (see Algorithm)

3. Find an optimal schedule for L exhaustively.

4. Concatenate.

Overview of recent results of PTASs minimizing average completion time

$P|r_j|\sum w_jC_j$

$P|r_j, pmtn|\sum w_jC_j$

$Rm|r_j|\sum w_jC_j$

$Rm|r_j, pmtn|\sum w_jC_j$

$P_2|fix_j|\sum C_j$

$Pm|fix_j, pmtn, r_j|\sum w_jC_j$