User Centered Scheduling for Multi-Users Systems

Erik Saule and Denis Trystram

INPG, LIG, Grenoble University
{firstname.lastname}@imag.fr

Gotha - Jan 2009
(accepted in IPDPS'09)

Motivation

- Parallel systems with multiple users: SMPs, Clusters
- Users submit tasks
- A scheduler assigns tasks to processor and time

Which function should the scheduler intent to optimize ?

- System centered objectives: min Makespan, min IdleTime
- User centered objectives: min SumFlow, min MaxStrech

Do not take user's wishes into account.

Motivation

- Parallel systems with multiple users: SMPs, Clusters
- Users submit tasks
- A scheduler assigns tasks to processor and time

Which function should the scheduler intent to optimize ?

- System centered objectives: min Makespan, min IdleTime
- User centered objectives: min SumFlow, min MaxStrech

Do not take user's wishes into account.

Motivation

- Parallel systems with multiple users: SMPs, Clusters
- Users submit tasks
- A scheduler assigns tasks to processor and time

Which function should the scheduler intent to optimize ?

- System centered objectives: min Makespan, min IdleTime
- User centered objectives: min SumFlow, min MaxStrech

Do not take user's wishes into account.

Motivation

- Parallel systems with multiple users: SMPs, Clusters
- Users submit tasks
- A scheduler assigns tasks to processor and time

Which function should the scheduler intent to optimize ?

- System centered objectives: min Makespan, min IdleTime
- User centered objectives: min SumFlow, min MaxStrech

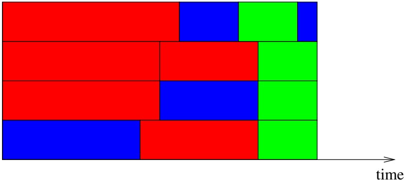
Do not take user's wishes into account.

An example

- Blue has a program to compile: Makespan
- Green uses an interactive application: Maximum flow time
- Red is running experiments: Sum of weighted completion time

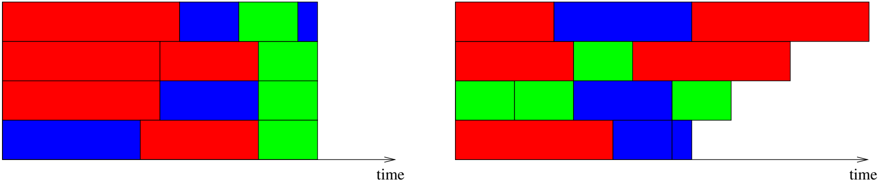
An example

- Blue has a program to compile: Makespan
- Green uses an interactive application: Maximum flow time
- Red is running experiments: Sum of weighted completion time



An example

- Blue has a program to compile: Makespan
- Green uses an interactive application: Maximum flow time
- Red is running experiments: Sum of weighted completion time



Outline of the talk

Notations

Instance

- k users
- m processors
- User u submits $n^{(u)}$ tasks
- Task $t_i^{(u)}$ of processing time $p_i^{(u)}$, belongs to u, released at $r_i^{(u)}$

Solution

- Function π, processor allocation.
- Function σ time allocation.

$C_i^{(u)}$ is the completion time of $t_i^{(u)}$

Notations

Instance

- k users
- m processors
- User u submits $n^{(u)}$ tasks
- Task $t_i^{(u)}$ of processing time $p_i^{(u)}$, belongs to u, released at $r_i^{(u)}$

Solution

- Function π, processor allocation.
- Function σ time allocation.

$C_i^{(u)}$ is the completion time of $t_i^{(u)}$

Notations

Objective functions

Each user chooses an objective function among:

- makespan: $C_{max}^{(u)} = \max_i C_i^{(u)}$
- sum of (weighted) completion time: $\sum C_i^{(u)} = \sum_i C_i^{(u)}$
- max flow time: $F_{max}^{(u)} = \max_i C_i^{(u)} - r_i^{(u)}$

The Multi-User Scheduling Problem

The problem will be denoted by:

- $MUSP(k : C_{max})$: all users are interested in the makespan
- $MUSP(k' : \sum C_i; k'' : C_{max})$: k' users are interested in the sum of completion time and k'' in the makespan

Notations

Objective functions

Each user chooses an objective function among:

- makespan: $C_{max}^{(u)} = \max_i C_i^{(u)}$
- sum of (weighted) completion time: $\sum C_i^{(u)} = \sum_i C_i^{(u)}$
- max flow time: $F_{max}^{(u)} = \max_i C_i^{(u)} - r_i^{(u)}$

The Multi-User Scheduling Problem

The problem will be denoted by:

- $MUSP(k : C_{max})$: all users are interested in the makespan
- $MUSP(k' : \sum C_i; k'' : C_{max})$: k' users are interested in the sum of completion time and k'' in the makespan

Related Works: Linear Aggregation (Baker & Smith, 2003)

One machine. Two users. Investigate C_{max} and $\sum \omega_i C_i$.
This is polynomial:

- The tasks of a makespan user are merged into a single task which is scheduled as a 'sum of completion time' task.
- Optimizing the sum of weighted completion time is polynomial (WSPT).

Can be extended to more users by the same technique.
However:

- Does not work with flowtime
- Can not extend to several processors
- Linear combination is bad.

Related Works: Linear Aggregation (Baker & Smith, 2003)

One machine. Two users. Investigate C_{max} and $\sum \omega_i C_i$.
This is polynomial:

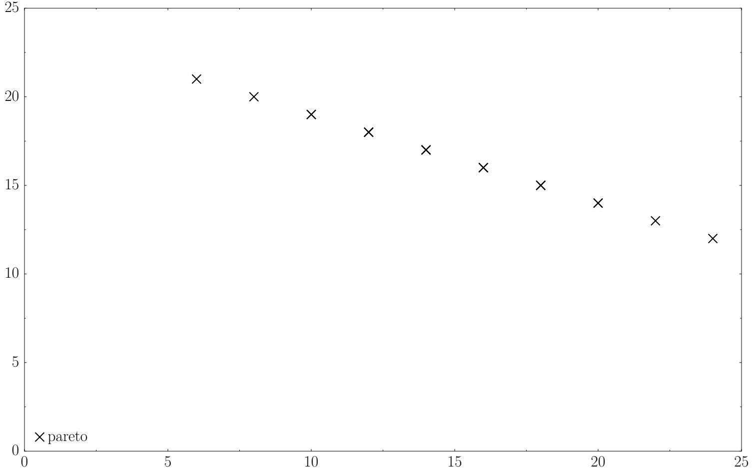
- The tasks of a makespan user are merged into a single task which is scheduled as a 'sum of completion time' task.
- Optimizing the sum of weighted completion time is polynomial (WSPT).

Can be extended to more users by the same technique.
However:

- Does not work with flowtime
- Can not extend to several processors
- Linear combination is bad.

Linear combination is unfair!

It is easy to construct instances of $MUSP(2 : \sum C_i)$ where the Pareto set is like:



Only two points can be reached. Both of them are unfair.

Linear combination is not truthful!

For makespan users

Selecting the sum of completion time objective always leads to better performance than selecting the makespan.

For sum of completion time users

Spliting tasks into two can help, even if it increases the total load.

Objective Function

A bag-of-tasks objective function

In bag-of-tasks scheduling, several applications compete for the computing resources. The common objective function is the (maximum or sum) stretch : $s_i = \frac{C_i}{p_i}$.
s_i is degradation factor for not being the only application in the system.

The Degradation objective function

Similarly, we define $d^{(u)}(S) = \frac{f^{(u)}(S)}{f^{(u)*}}$, the degradation of user u for not being the only user of the system.
The objective function is a norm of degradation (*e.g.* $\sum_u d^{(u)}(S)$, ...)

Optimizing Degradation

A difficult objective

Stretch based objective are difficult to tackle. Degradation are even worse. The (reachable) lower bound on $d^{(u)}(S) = \frac{f^{(u)}(S)}{f^{(u)*}}$ is 1. However, $f^{(u)*}$ is unknown.

Going multi-objective

An interesting property of norms is that they are monotone according to the component wise order. The optimal solution for a norm is a Pareto optimal solution of the $(d^{(1)}, \ldots, d^{(k)})$ **multi-objective optimization problem**.

The $(f^{(1)}, \ldots, f^{(k)})$ multi-objective optimization problem is **equivalent**.

Optimizing Degradation

A difficult objective

Stretch based objective are difficult to tackle. Degradation are even worse. The (reachable) lower bound on $d^{(u)}(S) = \frac{f^{(u)}(S)}{f^{(u)*}}$ is 1. However, $f^{(u)*}$ is unknown.

Going multi-objective

An interesting property of norms is that they are monotone according to the component wise order. The optimal solution for a norm is a Pareto optimal solution of the $(d^{(1)}, \ldots, d^{(k)})$ **multi-objective optimization problem**.
The $(f^{(1)}, \ldots, f^{(k)})$ multi-objective optimization problem is **equivalent**.

Related Works: Complexity (Agnetis *et al.* , 2004)

One machine. Two users.

Decision version

- $MUSP(2 : \sum C_i)$ is weakly NP-Complete
- $MUSP(2 : C_{max})$ and $MUSP(1 : C_{max}; 1 : \sum C_i)$ are polynomial
- $MUSP(2 : F_{max})$, $MUSP(1 : F_{max}; 1 : \sum C_i)$ and $MUSP(1 : F_{max}; 1 : C_{max})$ are polynomial

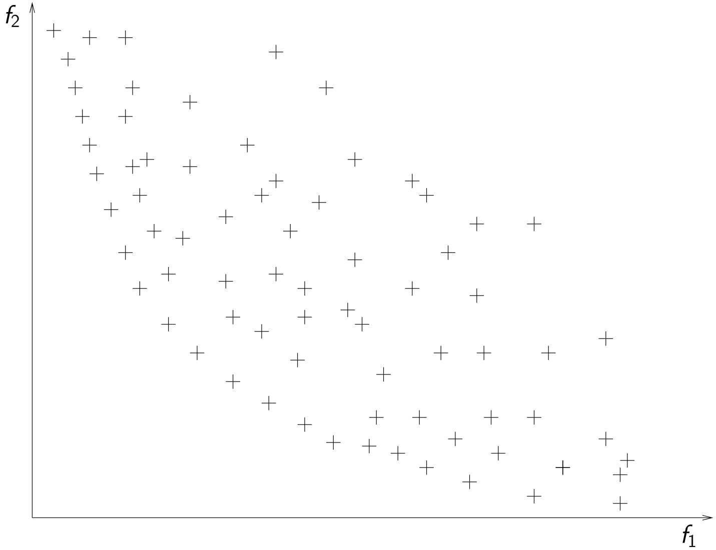
Thus, on a arbitrary number of processors. Everything is NP-Complete.

Multi-Objective Approximation

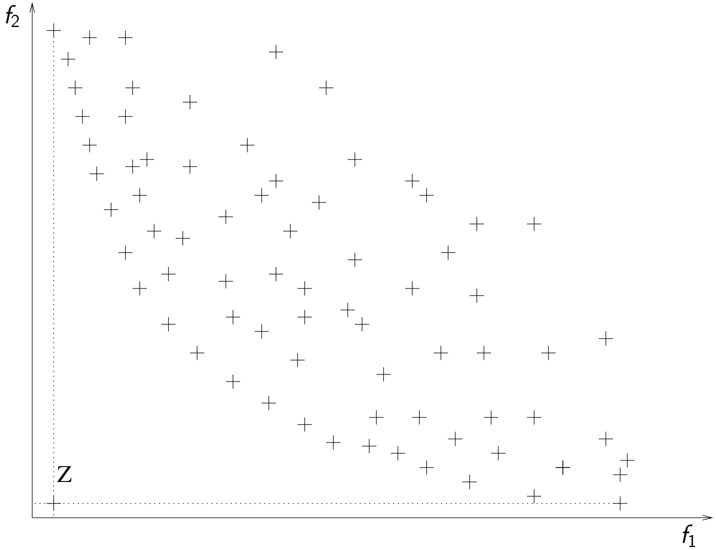
Two kinds of approximation techniques

- Zenith approximation : find a solution that approximates all objectives at the same time
- Pareto set approximation : find solutions that cover the Pareto set

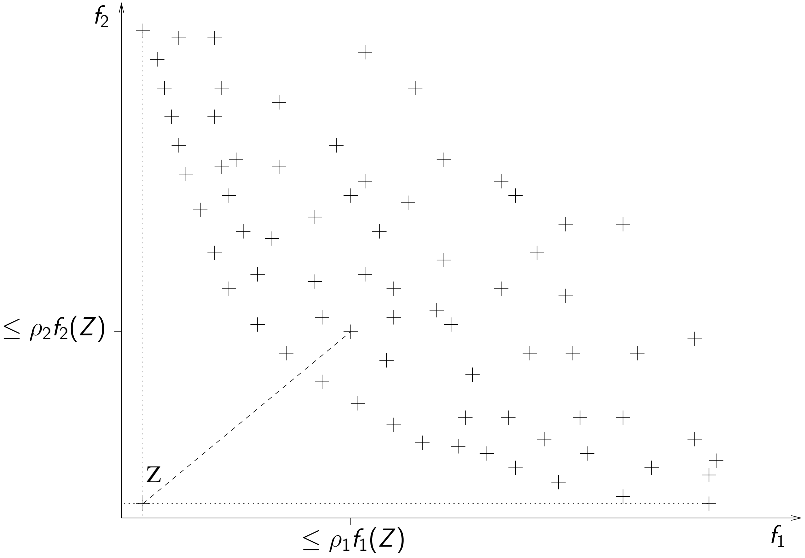
Multi-Objective Approximation



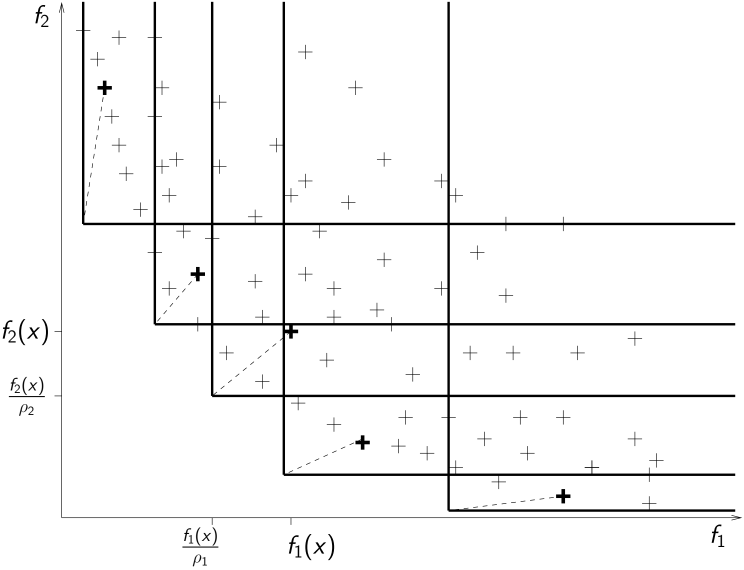
Multi-Objective Approximation



Multi-Objective Approximation

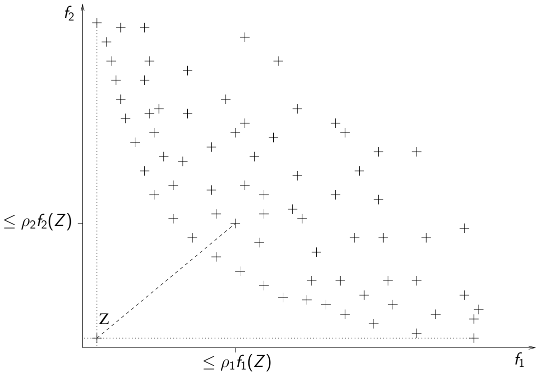


Multi-Objective Approximation



Which approximation to choose ?

A Degradation is a ratio to the optimal, given by the single user case.
A Zenith approximation ratio is an upper bound on degradations.



We study Zenith approximation.

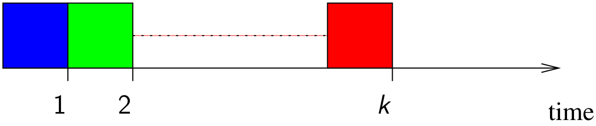
Outline

For $MUSP(k : C_{max})$

- One machine
- Each user has 1 task and chooses $C_{max}^{(u)}$
- $\forall u, p_1^{(u)} = 1$

$\Rightarrow \forall u, C_{max}^{(u)*} = 1$

No other choices than:



Theorem

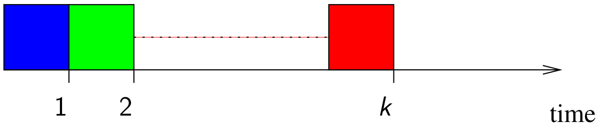
No algorithm approximates $MUSP(k : C_{max})$ better than $(1, 2, \ldots, k)$.

For $MUSP(k : C_{max})$

- One machine
- Each user has 1 task and chooses $C_{max}^{(u)}$
- $\forall u, p_1^{(u)} = 1$

$\Rightarrow \forall u, C_{max}^{(u)*} = 1$

No other choices than:



Theorem

No algorithm approximates $MUSP(k : C_{max})$ better than $(1, 2, \ldots, k)$.

Inapproximability for $MUSP(k : \sum C_i)$

- One machine
- Each user has x tasks and chooses $\sum C_i^{(u)}$
- $\forall u, \forall i, p_i^{(u)} = 1$

$\Rightarrow \forall u, \sum C_i^{(u)*} = \frac{x(x+1)}{2}$
Over all the tasks, $\sum C_i^* = \sum_{i=1}^{kx} i = \frac{kx(kx+1)}{2}$
A fair algorithm will not serve a user better than another one :
$\forall u, \sum C_i^{(u)} = Cst$.
Recall that, $\sum C_i = \sum_u \sum C_i^{(u)}$.
$\sum C_i^{(u)} \geq \frac{\sum C_i^*}{k} = \frac{kx^2+x}{2}$

Theorem

No algorithm approximates $MUSP(k : \sum C_i)$ better than $(k, \ldots, k)$.

Inapproximability for $MUSP(k : \sum C_i)$

- One machine
- Each user has x tasks and chooses $\sum C_i^{(u)}$
- $\forall u, \forall i, p_i^{(u)} = 1$

$\Rightarrow \forall u, \sum C_i^{(u)*} = \frac{x(x+1)}{2}$
Over all the tasks, $\sum C_i^* = \sum_{i=1}^{kx} i = \frac{kx(kx+1)}{2}$
A fair algorithm will not serve a user better than another one :
$\forall u, \sum C_i^{(u)} = Cst$.
Recall that, $\sum C_i = \sum_u \sum C_i^{(u)}$.
$\sum C_i^{(u)} \geq \frac{\sum C_i^*}{k} = \frac{kx^2+x}{2}$

Theorem

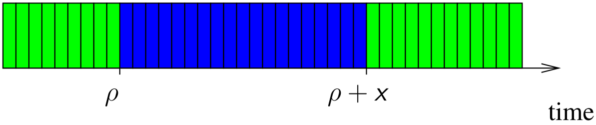
No algorithm approximates $MUSP(k : \sum C_i)$ better than $(k, \ldots, k)$.

For $MUSP(2 : F_{max})$

- One machine
- 2 users with x jobs
- $\forall i$ and u, $r_i^{(u)} = i - 1, p_i^{(u)} = 1$

$\Rightarrow \forall u, F_{max}^{(u)*} = 1.$

Let suppose a ρ approximated solution for user 1 (blue):



$$F_{max}^{(2)} = x$$

Theorem

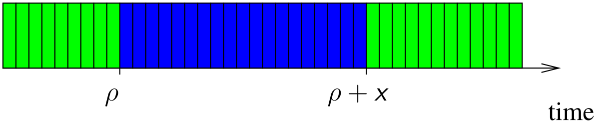
No algorithm approximates $MUSP(2 : F_{max})$ within a constant factor.

For $MUSP(2 : F_{max})$

- One machine
- 2 users with x jobs
- $\forall i$ and u, $r_i^{(u)} = i - 1, p_i^{(u)} = 1$

$\Rightarrow \forall u, F_{max}^{(u)*} = 1.$

Let suppose a ρ approximated solution for user 1 (blue):



$$F_{max}^{(2)} = x$$

Theorem

No algorithm approximates $MUSP(2 : F_{max})$ within a constant factor.

Outline

$MUSP(k : C_{max})$

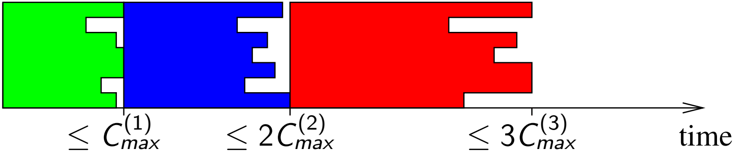
MULTICMAX

Given a ρ-approximation algorithm for the single user case

For each user u, compute $S^{(u)}$ such as $C_{max}(S^{(u)}) \leq \rho C_{max}^{(u)*}$

Group tasks of each user u according to $S^{(u)}$

Schedule them in increasing order of $C_{max}(S^{(u)})$



Theorem

MULTICMAX is a $(\rho, 2\rho, \ldots, k\rho)$-approximation algorithm (for an unknown order of users)

$MUSP(k : C_{max})$

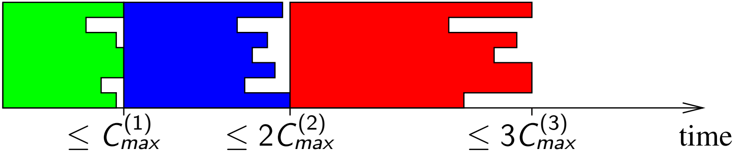
MultiCmax

Given a ρ-approximation algorithm for the single user case

For each user u, compute $S^{(u)}$ such as $C_{max}(S^{(u)}) \leq \rho C_{max}^{(u)*}$

Group tasks of each user u according to $S^{(u)}$

Schedule them in increasing order of $C_{max}(S^{(u)})$



Theorem

MultiCmax is a $(\rho, 2\rho, \ldots, k\rho)$-approximation algorithm (for an unknown order of users)

Distance to the Pareto Set

The *Lex* solution

Given an order of the users. *Lex* is a solution optimal for the first user, optimal for the second under constraint that the first is optimal, and recursively.
$Lex(u)$ denotes the solution on the u first users.
By definition, *Lex* is Pareto-optimal.

The *List* solutions

Sort users according to the optimal makespan of their job.
Schedule all the jobs user one after the other with List Scheduling.

Properties

List can be worsened to match MULTICMAX: $(2, \ldots, 2k)$-approximation
$\forall i, C_i \leq \frac{\sum_{i' \leq i} p_{i'}}{m} + (1 - \frac{1}{m})p_i$ [Graham 66]

Distance to the Pareto Set

The *Lex* solution

Given an order of the users. *Lex* is a solution optimal for the first user, optimal for the second under constraint that the first is optimal, and recursively.
$Lex(u)$ denotes the solution on the u first users.
By definition, *Lex* is Pareto-optimal.

The *List* solutions

Sort users according to the optimal makespan of their job.
Schedule all the jobs user one after the other with List Scheduling.

Properties

List can be worsened to match MULTICMAX: $(2, \ldots, 2k)$-approximation
$\forall i, C_i \leq \frac{\sum_{i' \leq i} p_{i'}}{m} + (1 - \frac{1}{m})p_i$ [Graham 66]

Distance to the Pareto Set

The *Lex* solution

Given an order of the users. *Lex* is a solution optimal for the first user, optimal for the second under constraint that the first is optimal, and recursively.
Lex(u) denotes the solution on the u first users.
By definition, *Lex* is Pareto-optimal.

The *List* solutions

Sort users according to the optimal makespan of their job.
Schedule all the jobs user one after the other with List Scheduling.

Properties

List can be worsened to match MULTICMAX: $(2, \ldots, 2k)$-approximation
$\forall i, C_i \leq \frac{\sum_{i' \leq i} p_{i'}}{m} + (1 - \frac{1}{m}) p_i$ [Graham 66]

Distance between *Lex* and *List*

Hypothesis

Each user u submits a fair amount of load: $\sum p_i^{(u)} > \frac{mC_{max}^{(u)*}}{2}$.

Property

$Idle(Lex(u)) < mC_{max}^{(u)*}$ (by local improvement)

Lemma

$\forall u > 2$, if $C_{max}^{(u-1)}(Lex) < C_{max}^{(u-2)}(Lex)$ then $C_{max}^{(u)}(Lex) > C_{max}^{(u-2)}(Lex)$ (from the Hypothesis and Property)

Theorem

$\forall u, C_{max}^{(u)}(List) \leq (3 - \frac{1}{m})C_{max}^{(u)}(Lex)$ (from Lemma and [Graham 66])

Distance between *Lex* and *List*

Hypothesis

Each user u submits a fair amount of load: $\sum p_i^{(u)} > \frac{mC_{max}^{(u)*}}{2}$.

Property

$Idle(Lex(u)) < mC_{max}^{(u)*}$ (by local improvement)

Lemma

$\forall u > 2$, if $C_{max}^{(u-1)}(Lex) < C_{max}^{(u-2)}(Lex)$ then $C_{max}^{(u)}(Lex) > C_{max}^{(u-2)}(Lex)$ (from the Hypothesis and Property)

Theorem

$\forall u, C_{max}^{(u)}(List) \leq (3 - \frac{1}{m})C_{max}^{(u)}(Lex)$ (from Lemma and [Graham 66])

Distance between *Lex* and *List*

Hypothesis

Each user u submits a fair amount of load: $\sum p_i^{(u)} > \frac{mC_{max}^{(u)*}}{2}$.

Property

$Idle(Lex(u)) < mC_{max}^{(u)*}$ (by local improvement)

Lemma

$\forall u > 2$, if $C_{max}^{(u-1)}(Lex) < C_{max}^{(u-2)}(Lex)$ then $C_{max}^{(u)}(Lex) > C_{max}^{(u-2)}(Lex)$ (from the Hypothesis and Property)

Theorem

$\forall u, C_{max}^{(u)}(List) \leq (3 - \frac{1}{m})C_{max}^{(u)}(Lex)$ (from Lemma and [Graham 66])

Distance between *Lex* and *List*

Hypothesis

Each user u submits a fair amount of load: $\sum p_i^{(u)} > \frac{mC_{max}^{(u)*}}{2}$.

Property

$Idle(Lex(u)) < mC_{max}^{(u)*}$ (by local improvement)

Lemma

$\forall u > 2$, if $C_{max}^{(u-1)}(Lex) < C_{max}^{(u-2)}(Lex)$ then $C_{max}^{(u)}(Lex) > C_{max}^{(u-2)}(Lex)$ (from the Hypothesis and Property)

Theorem

$\forall u, C_{max}^{(u)}(List) \leq (3 - \frac{1}{m})C_{max}^{(u)}(Lex)$ (from Lemma and [Graham 66])

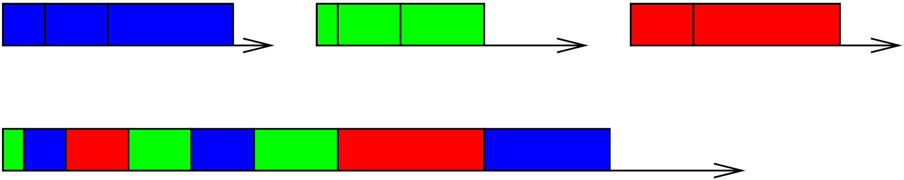
$MUSP(k : \sum C_i)$

On a single machine.

AGGREG

Let $S^{(u)}$ be a schedule of user u's jobs

Construct a schedule S of all the jobs in increasing order of $C_i^{(u)}(S^{(u)})$



Theorem

Schedule S is such that $\forall u, \forall i \leq n^{(u)}, C_i^{(u)}(S) \leq kC_i^{(u)}(S^{(u)})$

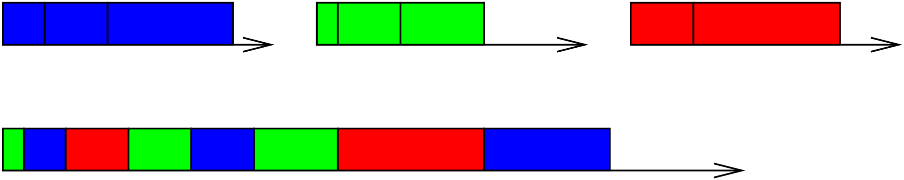
$MUSP(k : \sum C_i)$

On a single machine.

AGGREG

Let $S^{(u)}$ be a schedule of user u's jobs

Construct a schedule S of all the jobs in increasing order of $C_i^{(u)}(S^{(u)})$



Theorem

Schedule S is such that $\forall u, \forall i \leq n^{(u)}, C_i^{(u)}(S) \leq kC_i^{(u)}(S^{(u)})$

$MUSP(k : \sum C_i)$: extensions

Going to m processors

On several processors, the idea also works. Each processor is considered individually. The same property holds.
Using SPT AGGREG is a $(k, k, \ldots, k)$-approximation algorithm for $MUSP(k : \sum C_i)$

Parametric

Given a vector λ such as $\sum_u \lambda_u = 1$, the algorithm can be changed to schedule the tasks in increasing order of $\lambda_u C_i^{(u)}(S^{(u)})$
The property becomes: Schedule S is such that
$\forall u, \forall i \leq n^{(u)}, C_i^{(u)}(S) \leq \frac{kC_i^{(u)}(S^{(u)})}{\lambda_u}$

$MUSP(k : \sum C_i)$: extensions

Going to m processors

On several processors, the idea also works. Each processor is considered individually. The same property holds.

Using SPT AGGREG is a $(k, k, \ldots, k)$-approximation algorithm for $MUSP(k : \sum C_i)$

Parametric

Given a vector λ such as $\sum_u \lambda_u = 1$, the algorithm can be changed to schedule the tasks in increasing order of $\lambda_u C_i^{(u)}(S^{(u)})$

The property becomes: Schedule S is such that

$\forall u, \forall i \leq n^{(u)}, C_i^{(u)}(S) \leq \frac{kC_i^{(u)}(S^{(u)})}{\lambda_u}$

$MUSP(k' : \sum C_i; k'' : C_{max})$

A first idea

Consider makespan users as sum of completion users. This leads to a $(k, k, \ldots, k)$-approximation algorithm
However, the tasks of makespan users are not totally ordered

Merge MULTICMAX and AGGREG into MULTIMIXED

Build a schedule $S^{(C_{max})}$ for all the makespan users only using MULTICMAX.
Build a schedule $S^{(u)}$ for each sum of completion time user u.
Apply AGGREG with high priority for C_{max} : $\lambda_{C_{max}} = \frac{k''}{k}$ and standard priority the sum of completion users: $\lambda_u = \frac{1}{k}$.

- no (theoretical) overhead on the sum of completion time users
- does not mix makespan users' jobs

MULTIMIXED is a $(k, \ldots, k, \frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$-approximation algorithm

$MUSP(k' : \sum C_i; k'' : C_{max})$

A first idea

Consider makespan users as sum of completion users. This leads to a $(k, k, \ldots, k)$-approximation algorithm
However, the tasks of makespan users are not totally ordered

Merge MULTICMAX and AGGREG into MULTIMIXED

Build a schedule $S^{(C_{max})}$ for all the makespan users only using MULTICMAX.
Build a schedule $S^{(u)}$ for each sum of completion time user u.
Apply AGGREG with high priority for C_{max} : $\lambda_{C_{max}} = \frac{k''}{k}$ and standard priority the sum of completion users: $\lambda_u = \frac{1}{k}$.

- no (theoretical) overhead on the sum of completion time users
- does not mix makespan users' jobs

MULTIMIXED is a $(k, \ldots, k, \frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$-approximation algorithm

$MUSP(k' : \sum C_i; k'' : C_{max})$

A first idea

Consider makespan users as sum of completion users. This leads to a $(k, k, \ldots, k)$-approximation algorithm
However, the tasks of makespan users are not totally ordered

Merge MULTICMAX and AGGREG into MULTIMIXED

Build a schedule $S^{(C_{max})}$ for all the makespan users only using MULTICMAX.
Build a schedule $S^{(u)}$ for each sum of completion time user u.
Apply AGGREG with high priority for C_{max} : $\lambda_{C_{max}} = \frac{k''}{k}$ and standard priority the sum of completion users: $\lambda_u = \frac{1}{k}$.

- no (theoretical) overhead on the sum of completion time users
- does not mix makespan users' jobs

MULTIMIXED is a $(k, \ldots, k, \frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$-approximation algorithm

Outline

To sum up

In general

- Linear combination is bad !
- A new metric has been proposed (norm of degradation)

Zenith Approximation

- $MUSP(k : C_{max})$:
  - no algorithm better than $(1, 2, \ldots, k)$.
  - MULTICMAX reaches $(\rho, 2\rho, \ldots, k\rho)$.
- $MUSP(k : \sum C_i)$:
  - no algorithm better than $(k, k, \ldots, k)$.
  - AGGREG reaches it.
- $MUSP(k' : \sum C_i; k'' : C_{max})$:
  - no known lower bound.
  - MULTIMIXED $(k, \ldots, k, \frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$.

To sum up

In general

- Linear combination is bad !
- A new metric has been proposed (norm of degradation)

Zenith Approximation

- $MUSP(k : C_{max})$:
  - no algorithm better than $(1, 2, \ldots, k)$.
  - MULTICMAX reaches $(\rho, 2\rho, \ldots, k\rho)$.
- $MUSP(k : \sum C_i)$:
  - no algorithm better than $(k, k, \ldots, k)$.
  - AGGREG reaches it.
- $MUSP(k' : \sum C_i; k'' : C_{max})$:
  - no known lower bound.
  - MULTIMIXED $(k, \ldots, k, \frac{k}{k''}\rho, \frac{2k}{k''}\rho, \ldots, k\rho)$.

Perspective

A new kind of analysis (zenith approximation + pareto set distance)

- Currently given for a subset of $MUSP(k : C_{max})$.
- Should be generalized to the other objectives
- Can this analysis be applied on different problems ?

Some problems

- Flow time can not be tackled this way
- Add more constraints such as precedence or rigid tasks
- Extend to other objectives

Zenith approximation will be tough to adapt. Would Pareto set approximation be easier ?

Thank You

Questions ?

Agnetis, A., Mirchandani, P. B., Pacciarelli, D., & Pacifici, A. 2004.
Scheduling Problems with Two Competing Agents.
Operations Research, **52**(2), 229–242.

Baker, K., & Smith, J.C. 2003.
A Multiple-Criterion Model for Machine Scheduling.
Journal of Scheduling, **6**, 7–16.