Scheduling Unrelated Machines of Few Different Types

Vincenzo Bonifaci $^1$
joint with
Andreas Wiese $^2$

$^1$ IASI-CNR, Rome, Italy
$^2$ Sapienza U. Rome, Italy

1st Interdisciplinary Workshop on Algorithmic Challenges in Real-Time Systems
Motivation

Motivation

Suppose we are given heterogeneous processors

and some sporadic ("Liu-Layland") real-time tasks
Motivation

Suppose we are given heterogeneous processors

and some sporadic (“Liu-Layland”) real-time tasks

Each task has a utilization \textit{and} a memory requirement \textit{that could both depend on the type of processor}

Can we schedule the tasks (almost) optimally?
Problem Definition: Multidimensional Load Minimization

Input:
- set of \( n \) jobs \( J \)
- set of \( m \) machines \( M \)
- set of \( D \) dimensions \([D] = \{1, \ldots, D\}\) (\( D \) is some constant)
- size \( c_{i,j}^d \) for each \( j \in J, i \in M, d \in [D] \)
Problem Definition: Multidimensional Load Minimization

Input:
- set of $n$ jobs $J$
- set of $m$ machines $M$
- set of $D$ dimensions $[D] = \{1, \ldots, D\}$ ($D$ is some constant)
- size $c_{i,j}^d$ for each $j \in J$, $i \in M$, $d \in [D]$

Output:
- assignment $f : J \rightarrow M$
Problem Definition: Multidimensional Load Minimization

Input:

- set of $n$ jobs $J$
- set of $m$ machines $M$
- set of $D$ dimensions $[D] = \{1, \ldots, D\}$ ($D$ is some constant)
- size $c^d_{i,j}$ for each $j \in J$, $i \in M$, $d \in [D]$

Output:

- assignment $f : J \rightarrow M$

Cost:

- Let $\text{load}^d(i) := \sum_{j \in f^{-1}(i)} c^d_{i,j}$
- Goal is to minimize $\text{cost}(f) := \max_{d \in [D]} \text{load}^d(i)$
Wait... is it really consistent with the motivation?

Say $M_1$ has available utilization = 0.9, available memory = 2
$M_2$ has available utilization = 0.69, available memory = 5
Wait... is it really consistent with the motivation?

Say $M_1$ has available utilization = 0.9, available memory = 2
$M_2$ has available utilization = 0.69, available memory = 5

Scale $c_{1,j}^{util}$ by $1/0.9$, $c_{1,j}^{mem}$ by $1/2$,
$c_{2,j}^{util}$ by $1/0.69$, $c_{2,j}^{mem}$ by $1/5$
Wait... is it really consistent with the motivation?

Say $M_1$ has available utilization = 0.9, available memory = 2
$M_2$ has available utilization = 0.69, available memory = 5

Scale $c_{1,j}^{util}$ by $1/0.9$, $c_{1,j}^{mem}$ by $1/2$,
$c_{2,j}^{util}$ by $1/0.69$, $c_{2,j}^{mem}$ by $1/5$

$\exists f : cost(f) \leq 1 \iff$ there is a feasible assignment for the original instance
Recall a \( \rho \)-approximation is a solution \( f \) with \( \text{cost}(f) \leq \rho \cdot \text{OPT} \).
Recall a $\rho$-approximation is a solution $f$ with $\text{cost}(f) \leq \rho \cdot \text{OPT}$. The general problem is quite hard even for one dimension:

**Lenstra, Shmoys & Tardos 1990**

- a polynomial time 2-approximation algorithm exists, but
Related Work

Recall a \( \rho \)-approximation is a solution \( f \) with \( \text{cost}(f) \leq \rho \cdot \text{OPT} \). The general problem is quite hard even for one dimension:

<table>
<thead>
<tr>
<th>Lenstra, Shmoys &amp; Tardos 1990</th>
</tr>
</thead>
<tbody>
<tr>
<td>a polynomial time 2-approximation algorithm exists, but</td>
</tr>
<tr>
<td>no polynomial time 1.5-approximation algorithm exists unless ( P=NP ).</td>
</tr>
</tbody>
</table>
Recall a \( \rho \)-approximation is a solution \( f \) with \( \text{cost}(f) \leq \rho \cdot \text{OPT} \). The general problem is quite hard even for one dimension:

### Lenstra, Shmoys & Tardos 1990

- a polynomial time 2-approximation algorithm exists, but
- no polynomial time 1.5-approximation algorithm exists unless \( P=NP \).

Still, efficient approximation schemes (PTASs) are known for:

- arbitrary number of identical parallel machines
  (Hochbaum & Shmoys 1987; Chekuri & Khanna 2004)
Recall a \( \rho \)-approximation is a solution \( f \) with \( \text{cost}(f) \leq \rho \cdot \text{OPT} \). The general problem is quite hard even for one dimension:

**Lenstra, Shmoys & Tardos 1990**

- a polynomial time 2-approximation algorithm exists, but
- no polynomial time 1.5-approximation algorithm exists unless \( \text{P=NP} \).

Still, efficient approximation schemes (PTASs) are known for:

- arbitrary number of identical parallel machines
  (Hochbaum & Shmoys 1987; Chekuri & Khanna 2004)
- constant number of unrelated parallel machines
  (Horowitz & Sahni 1976)
In practice, we might have many processors of a few different types (a job has the same requirements on all processors of the same type). So what if the number \( K \) of types of processors is fixed?
In practice, we might have many processors of a few different types (a job has the same requirements on all processors of the same type)

So what if the number $K$ of types of processors is fixed?

**Theorem**

Let $K, D \in \mathbb{N}$. There is a PTAS for Multidimensional Load Minimization with $K$ types and $D$ dimensions.
Outline of the Algorithm

- Binary search for OPT
- Decision procedure
  - Size preprocessing
  - Guessing/enumeration
  - Iterative LP rounding

Decision procedure
Preprocessing
Enumeration
Iterative LP-rounding
Outline of the Algorithm

- Binary search for OPT
  - Decision procedure
    - Size preprocessing
    - Guessing/ enumeration
    - Iterative LP rounding

Decision procedure

Find an assignment such that

\[ \text{load}^d(i) \leq 1 + \epsilon \quad \forall d \forall i \]

or determine that no assignment exists such that

\[ \text{load}^d(i) \leq 1 \quad \forall d \forall i. \]
A job $j$ is *large* on machine $i$ if 

$$c_{i,j}^d > \epsilon \quad \text{for some } d \in [D].$$

There are at most $D/\epsilon$ large jobs on each machine.
A job $j$ is *large* on machine $i$ if

$$c_{i,j}^d > \epsilon$$

for some $d \in [D]$.

There are at most $D/\epsilon$ large jobs on each machine. But, their other dimensions might be small.
A job $j$ is *large* on machine $i$ if
\[ c^d_{i,j} > \epsilon \text{ for some } d \in [D]. \]

There are at most $D/\epsilon$ large jobs on each machine.
But, their other dimensions might be small.
So if $j$ is large on $i$, we reset
\[ c^d_{i,j} \leftarrow \max\{c^d_{i,j}, \epsilon^2/D\} \]

This increases the objective by no more than $(D/\epsilon) \cdot (\epsilon^2/D) = \epsilon$. 
There are still too many job sizes. We round them all up to powers of $1/(1 + \epsilon)$.
There are still too many job sizes.
We round them all up to powers of $1/(1 + \epsilon)$.

This increases the objective by a factor at most $(1 + \epsilon)$.
There are still too many job sizes. We round them all up to powers of $1/(1 + \epsilon)$. This increases the objective by a factor at most $(1 + \epsilon)$.

Each large pair $(i, j)$ now yields a vector $q = (c_{i,j}^1, \ldots, c_{i,j}^D)$ where each $c_{i,j}^d$ is a power of $1/(1 + \epsilon)$ between $\epsilon^2/D$ and 1.
There are still too many job sizes.
We round them all up to powers of $1/(1 + \epsilon)$.

This increases the objective by a factor at most $(1 + \epsilon)$.

Each *large* pair $(i, j)$ now yields a vector $q = (c_{i,j}^1, \ldots, c_{i,j}^D)$ where each $c_{i,j}^d$ is a power of $1/(1 + \epsilon)$ between $\epsilon^2/D$ and 1.

**Corollary**

*There are a constant number* $Q = Q(\epsilon, D)$ *of possible large job types* $q$.

*(Small pairs might still be many)*
Remember there are at most $D/\epsilon$ large jobs on a machine. For each machine there are $(D/\epsilon)^Q = O(1)$ patterns $\pi$ of large jobs of each type.
Enumeration

Remember there are at most $D/\epsilon$ large jobs on a machine. For each machine there are $(D/\epsilon)^Q = O(1)$ patterns $\pi$ of large jobs

For each type, enumerate how many machines follow which pattern

There are $(m + 1)^K = poly(m)$ combinations
Enumeration

Remember there are at most $D/\epsilon$ large jobs on a machine. For each machine there are $(D/\epsilon)^Q = O(1)$ patterns $\pi$ of large jobs

*For each type*, enumerate how many machines follow which pattern

There are $(m + 1)^{K_{\kappa}} = \text{poly}(m)$ combinations

Each pattern for $i$ creates a set of “slots” and some remaining space
Enumeration

Remember there are at most $D/\varepsilon$ large jobs on a machine.
For each machine there are $(D/\varepsilon)^Q = O(1)$ patterns $\pi$ of large jobs

For each type, enumerate how many machines follow which pattern

There are $(m + 1)^K = \text{poly}(m)$ combinations

Each pattern for $i$ creates a set of “slots” and some remaining space
$x_{i,j}: j$ assigned to remaining space on $i$
ILP Formulation and LP Relaxation

$x_{i,j}$: $j$ assigned to remaining space on $i$

$x_{s,j}$: $j$ assigned to slot $s$
$x_{i,j}$: $j$ assigned to remaining space on $i$

$x_{s,j}$: $j$ assigned to slot $s$

(we assume $x_{i,j} = 0$ if $j$ is not small on $i$ and $x_{s,j} = 0$ if $j$ does not fit $s$)
ILP Formulation and LP Relaxation

\( x_{i,j} \): \( j \) assigned to remaining space on \( i \)

\( x_{s,j} \): \( j \) assigned to slot \( s \)

(we assume \( x_{i,j} = 0 \) if \( j \) is not small on \( i \) and \( x_{s,j} = 0 \) if \( j \) does not fit \( s \))

\[
\text{(Slot-LP)} \quad \sum_{i \in M} x_{i,j} + \sum_{s \in S} x_{s,j} = 1 \quad \forall j \in J \tag{1}
\]

\[
\sum_{j \in J} x_{s,j} \leq 1 \quad \forall s \in S \tag{2}
\]

\[
\sum_{j \in J} c^{d}_{i,j} \cdot x_{i,j} \leq \text{rem}^{d}(i) \quad \forall i \in M, \forall d = 1, \ldots, D \tag{3}
\]

\[
x_{i,j} \geq 0 \quad \forall i \in M, \forall j \in J
\]

\[
x_{s,j} \geq 0 \quad \forall s \in S, \forall j \in J.
\]
We will iteratively round the LP relaxation
Iterative Rounding Framework

We will iteratively round the LP relaxation.
Observation: the LP is **sparse** (\# nontriv. constraints = $n + mD + \# \text{ slots}$)

Let $x^*$ be an optimal (fractional) extreme point solution.

Freeze (remove) all variables having zero value in $x^*$ (if any).

**Lemma**

In $x^*$ there is either

1. a machine which has at most 2D small jobs fractionally assigned to it,
2. a slot which has at most 2 jobs fractionally assigned to it.

**Proof.**

Careful counting argument.
Iterative Rounding Framework

We will iteratively round the LP relaxation

Observation: the LP is sparse (\# nontriv. constraints = $n + mD + \#$ slots)

Let $x^*$ be an optimal (fractional) extreme point solution

Freeze (remove) all variables having zero value in $x^*$ (if any)
Iterative Rounding Framework

We will iteratively round the LP relaxation
Observation: the LP is sparse (# nontriv. constraints = \( n + mD + \# \) slots)
Let \( x^* \) be an optimal (fractional) extreme point solution
Freeze (remove) all variables having zero value in \( x^* \) (if any)

Lemma

\( \text{In } x^* \text{ there is either} \)

1. a machine which has at most 2D small jobs fractionally assigned to it,
or
Iterative Rounding Framework

We will iteratively round the LP relaxation
Observation: the LP is sparse (# nontriv. constraints = $n + mD + \#$ slots)
Let $x^*$ be an optimal (fractional) extreme point solution
Freeze (remove) all variables having zero value in $x^*$ (if any)

Lemma

In $x^*$ there is either
1. a machine which has at most $2D$ small jobs fractionally assigned to it, or
2. a slot which has at most 2 jobs fractionally assigned to it.
Iterative Rounding Framework

We will iteratively round the LP relaxation
Observation: the LP is sparse ($\#$ nontriv. constraints $= n + mD + \#$ slots)
Let $x^*$ be an optimal (fractional) extreme point solution
Freeze (remove) all variables having zero value in $x^*$ (if any)

Lemma

In $x^*$ there is either

1. a machine which has at most $2D$ small jobs fractionally assigned to it,
   or
2. a slot which has at most 2 jobs fractionally assigned to it.

Proof.

Careful counting argument.
Case 1:

In $x^*$ there is a machine $i$ which has at most $2D$ small jobs fractionally assigned to it.
Handling Fractional Machines

Case 1:

In $x^*$ there is a machine $i$ which has at most $2D$ small jobs fractionally assigned to it.

Remove the space constraint for $i$

In the worst case, all jobs fractionally assigned to $i$ will be integrally assigned
Handling Fractional Machines

Case 1:

In $x^*$ there is a machine $i$ which has at most $2D$ small jobs fractionally assigned to it.

Remove the space constraint for $i$

In the worst case, all jobs fractionally assigned to $i$ will be integrally assigned

They are all small on $i$

Additional load on $i$: $2D\epsilon$ (ok)
Case 2:

1. In $x^*$ there is a slot $s$ which has 2 jobs $j_1, j_2$ fractionally assigned to it.

Assign $j_1$ or $j_2$? Too early to commit...
Case 2:

1. In $x^*$ there is a slot $s$ which has 2 jobs $j_1, j_2$ fractionally assigned to it.

Assign $j_1$ or $j_2$? Too early to commit...
Create an “artificial” job $j_0$ that is the “least common multiple” of $j_1$ and $j_2$:

- $j_0$ can go to slots where either $j_1$ or $j_2$ can go;
Case 2:

1. In $x^*$ there is a slot $s$ which has 2 jobs $j_1, j_2$ fractionally assigned to it.

Assign $j_1$ or $j_2$? Too early to commit...

Create an “artificial” job $j_0$ that is the “least common multiple” of $j_1$ and $j_2$:

- $j_0$ can go to slots where either $j_1$ or $j_2$ can go;
- $j_0$ can go to remaining space where either $j_1$ or $j_2$ can go;
Case 2:

1. In $x^*$ there is a slot $s$ which has 2 jobs $j_1, j_2$ fractionally assigned to it.

Assign $j_1$ or $j_2$? Too early to commit...

Create an “artificial” job $j_0$ that is the “least common multiple” of $j_1$ and $j_2$:

- $j_0$ can go to slots where either $j_1$ or $j_2$ can go;
- $j_0$ can go to remaining space where either $j_1$ or $j_2$ can go;
- if both $j_1$ and $j_2$ can go to $i$, we set $c_{i,j_0}$ as an appropriate convex combination of the two.
Handling Fractional Slots

Case 2:

1. In $x^*$ there is a slot $s$ which has 2 jobs $j_1, j_2$ fractionally assigned to it.

Assign $j_1$ or $j_2$? Too early to commit...

Create an “artificial” job $j_0$ that is the “least common multiple” of $j_1$ and $j_2$:

- $j_0$ can go to slots where either $j_1$ or $j_2$ can go;
- $j_0$ can go to remaining space where either $j_1$ or $j_2$ can go;
- if both $j_1$ and $j_2$ can go to $i$, we set $c_{i,j_0}$ as an appropriate convex combination of the two.

Remove $s, j_1, j_2$; add $j_0$
Lemma

After each modification (Case 1 or 2), the LP stays feasible.
LP Iterations

Lemma

*After each modification (Case 1 or 2), the LP stays feasible.*

In each iteration we either

1. find some integral variable, or
Lemma

After each modification (Case 1 or 2), the LP stays feasible.

In each iteration we either

1. find some integral variable, or
2. decrease the number of jobs and slots by one, or
Lemma

After each modification (Case 1 or 2), the LP stays feasible.

In each iteration we either

1. find some integral variable, or
2. decrease the number of jobs and slots by one, or
3. decrease the number of machine constraints by one.

So there is a polynomial number of iterations.
Replacing the Artificial Jobs

By removing the artificial jobs, we can assign all the original jobs except one. The extra one is still provided for in the final LP by artificial job $j$. With some extra rounding ($+D\epsilon$ per machine) we also handle that...
Beyond the Maximum Load

Two extremes:

- Minimize the maximum machine load: strive for maximum balance
- Minimize the sum of machine loads: assign each job where it's fastest

It makes sense to interpolate between the two

New goal: minimize 
\[ \| \text{load}(M_1), \ldots, \text{load}(M_m) \|_p, \text{ for } 1 < p < \infty \]

\[ L_p \text{ norm: } \| (a_1, \ldots, a_m) \|_p = (a_1^p + \ldots + a_m^p)^{1/p} \]

\[ p = 1: \text{Sum of machine loads} \]
\[ p = \infty: \text{Max of machine loads} \]

Theorem

Let \( K \in \mathbb{N}, 1 < p < \infty \). There is PTAS for \( L_p \)-norm Minimization with \( K \) types of processors (with unidimensional jobs).

Extends results by Alon et al. (1998) and Azar & Epstein (2005)
Beyond the Maximum Load

Two extremes:
- minimize the maximum machine load: strive for maximum balance

\[ \| \text{load}(M_1), \ldots, \text{load}(M_m) \|_p \]

\( \| \cdot \|_p \) norm:
- \( \| \cdot \|_1 \): Sum of machine loads
- \( \| \cdot \|_\infty \): Max of machine loads

Theorem
Let \( K \in \mathbb{N} \), \( 1 < p < \infty \). There is PTAS for \( L_p \)-norm Minimization with \( K \) types of processors (with unidimensional jobs).

Extends results by Alon et al. (1998) and Azar & Epstein (2005)
Beyond the Maximum Load

Two extremes:

- minimize the maximum machine load: strive for maximum balance
- minimize the sum of machine loads: assign each job where it’s fastest

It makes sense to interpolate between the two

$$\|\text{load}(M_1), \ldots, \text{load}(M_m)\|_p,$$ for $1 < p < \infty$

$L_p$ norm:

$$\| (a_1, \ldots, a_m) \|_p = (a_1^p + \ldots + a_m^p)^{1/p}$$

$p = 1$: Sum of machine loads
$p = \infty$: Max of machine loads

Theorem

Let $K \in \mathbb{N}$, $1 < p < \infty$. There is PTAS for $L_p$-norm Minimization with $K$ types of processors (with unidimensional jobs).

Extends results by Alon et al. (1998) and Azar & Epstein (2005)
Beyond the Maximum Load

Two extremes:

- minimize the maximum machine load: strive for maximum balance
- minimize the sum of machine loads: assign each job where it’s fastest

It makes sense to interpolate between the two

New goal: minimize $\|\text{load}(M_1), \ldots, \text{load}(M_m)\|_p$, for $1 < p < \infty$

$L_p$ norm: $\|(a_1, \ldots, a_m)\|_p = (a_1^p + \ldots + a_m^p)^{1/p}$

$p = 1$: Sum of machine loads

$p = \infty$: Max of machine loads
Beyond the Maximum Load

Two extremes:
- minimize the maximum machine load: strive for maximum balance
- minimize the sum of machine loads: assign each job where it’s fastest

It makes sense to interpolate between the two

New goal: minimize $\|\text{load}(M_1), \ldots, \text{load}(M_m)\|_p$, for $1 < p < \infty$

$L_p$ norm: $\|(a_1, \ldots, a_m)\|_p = (a_1^p + \ldots + a_m^p)^{1/p}$

$p = 1$: Sum of machine loads

$p = \infty$: Max of machine loads

**Theorem**

Let $K \in \mathbb{N}$, $1 < p < \infty$. There is PTAS for $L_p$-norm Minimization with $K$ types of processors (with unidimensional jobs).
Beyond the Maximum Load

Two extremes:

- minimize the maximum machine load: strive for maximum balance
- minimize the sum of machine loads: assign each job where it’s fastest

It makes sense to interpolate between the two

New goal: minimize $\|\text{load}(M_1), \ldots, \text{load}(M_m)\|_p$, for $1 < p < \infty$

$L_p$ norm: $\|(a_1, \ldots, a_m)\|_p = (a_1^p + \ldots + a_m^p)^{1/p}$

$p = 1$: Sum of machine loads

$p = \infty$: Max of machine loads

**Theorem**

Let $K \in \mathbb{N}$, $1 < p < \infty$. There is PTAS for $L_p$-norm Minimization with $K$ types of processors (with unidimensional jobs).

Extends results by Alon et al. (1998) and Azar & Epstein (2005)
Convex Program

\[(\text{Slot-CP}) \quad \min \sum_{i \in M_s} (t_i + B_i)^p + \sum_{\ell \in T} \sum_{i \in M_{vh, \ell}} (t_i^*)^p + \sum_{\ell \in T} \sum_{j \in J} (c_{\ell,j})^p \cdot x_{\ell,j} \]

\[
\sum_{i \in M_s} x_{i,j} + \sum_{s \in S} x_{s,j} + \sum_{\ell \in T} x_{\ell,j} = 1 \quad \forall j \in J
\]

\[
\sum_{j \in H_{\ell}} x_{\ell,j} \leq h_{\ell} \quad \forall \ell \in T
\]

\[
\sum_{j \in J} x_{s,j} \leq 1 \quad \forall s \in S
\]

\[
\sum_{j \in J} c_{i,j} \cdot x_{i,j} \leq t_i \quad \forall i \in M_s
\]

\[
\alpha_{\ell} \cdot c_{\max} \leq t_i \quad \forall \ell \in T, \forall i \in M_{s,\ell}
\]

\[x_{i,j} \geq 0 \quad \forall i \in M_s, \forall j \in J\]

\[x_{s,j} \geq 0 \quad \forall s \in S, \forall j \in J\]

\[x_{\ell,j} \geq 0 \quad \forall \ell \in T, \forall j \in J\]

\[t_i \geq 0 \quad \forall i \in M_s.\]
Conclusions

- The limited heterogeneity of the processors leads to drastic improvements in the approximability
- Is there a more practical PTAS?
- A “little” enumeration might go a long way
Conclusions

- The limited heterogeneity of the processors leads to drastic improvements in the approximability
- Is there a more practical PTAS?
- A “little” enumeration might go a long way

Thanks!