Introduction to Scheduling Theory

Arnaud Legrand

Laboratoire Informatique et Distribution
IMAG CNRS, France
arnaud.legrand@imag.fr

November 8, 2004

Outline

Outline

Analyzing a simple code

Solving $A.x = B$ where A is lower triangular matrix

1: **for** $i = 1$ to n **do**
2: Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
3: **for** $j = i + 1$ to n **do**
4: Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

For a given value $1 \leqslant i \leqslant n$, all tasks $T_{i,*}$ are computations done during the i^{th} iteration of the outer loop.

$<_{seq}$ is the sequential order :

$$T_{1,1} <_{seq} T_{1,2} <_{seq} T_{1,3} <_{seq} \cdots <_{seq} T_{1,n} <_{seq} T_{2,2} <_{seq} T_{2,3} <_{seq} \cdots <_{seq} T_{n,n} \cdot$$

Analyzing a simple code

Solving $A.x = B$ where A is lower triangular matrix

1: **for** $i = 1$ to n **do**
2: Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
3: **for** $j = i + 1$ to n **do**
4: Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

For a given value $1 \leqslant i \leqslant n$, all tasks $T_{i,*}$ are computations done during the i^{th} iteration of the outer loop.

$<_{seq}$ is the sequential order :

$$T_{1,1} <_{seq} T_{1,2} <_{seq} T_{1,3} <_{seq} \cdots <_{seq} T_{1,n} <_{seq} T_{2,2} <_{seq} T_{2,3} <_{seq} \cdots <_{seq} T_{n,n} .$$

Analyzing a simple code

Solving $A.x = B$ where A is lower triangular matrix

1: **for** $i = 1$ to n **do**
2: Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
3: **for** $j = i + 1$ to n **do**
4: Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

For a given value $1 \leqslant i \leqslant n$, all tasks $T_{i,*}$ are computations done during the i^{th} iteration of the outer loop.

$<_{seq}$ is the sequential order :

$$T_{1,1} <_{seq} T_{1,2} <_{seq} T_{1,3} <_{seq} \ldots <_{seq} T_{1,n} <_{seq} T_{2,2} <_{seq} T_{2,3} <_{seq} \ldots <_{seq} T_{n,n} .$$

Independence

However, some independent tasks could be executed in parallel.
Independent tasks are the ones whose execution order can be changed without modifying the result of the program.
Two independent tasks may read the value but never write to the same memory location.

For a given task T, $In(T)$ denotes the set of input variables and $Out(T)$ the set of output variables.
In the previous example, we have :

$$\begin{cases} In(T_{i,i}) = \{b(i), a(i,i)\} \\ Out(T_{i,i}) = \{x(i)\} \text{ and} \end{cases}$$

$$\begin{cases} In(T_{i,j}) = \{b(j), a(j,i), x(i)\} \\ Out(T_{i,j}) = \{b(j)\} \text{ for } j > i. \end{cases}$$

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Independence

However, some independent tasks could be executed in parallel.
Independent tasks are the ones whose execution order can be changed without modifying the result of the program.
Two independent tasks may read the value but never write to the same memory location.

For a given task T, $In(T)$ denotes the set of input variables and $Out(T)$ the set of output variables.

In the previous example, we have :

$$\begin{cases} In(T_{i,i}) = \{b(i), a(i,i)\} \\ Out(T_{i,i}) = \{x(i)\} \text{ and} \end{cases}$$

$$\begin{cases} In(T_{i,j}) = \{b(j), a(j,i), x(i)\} \\ Out(T_{i,j}) = \{b(j)\} \text{ for } j > i. \end{cases}$$

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Independence

However, some independent tasks could be executed in parallel.
Independent tasks are the ones whose execution order can be changed without modifying the result of the program.
Two independent tasks may read the value but never write to the same memory location.

For a given task T, $In(T)$ denotes the set of input variables and $Out(T)$ the set of output variables.
In the previous example, we have :

$$\begin{cases} In(T_{i,i}) = \{b(i), a(i,i)\} \\ Out(T_{i,i}) = \{x(i)\} \text{ and} \end{cases}$$

$$\begin{cases} In(T_{i,j}) = \{b(j), a(j,i), x(i)\} \\ Out(T_{i,j}) = \{b(j)\} \text{ for } j > i. \end{cases}$$

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Bernstein conditions

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Ber66].

We can check that:

- $Out(T_{1,1}) \cap In(T_{1,2}) = \{x(1)\}$ $\rightsquigarrow T_{1,1} \perp T_{1,2}$.
- $Out(T_{1,3}) \cap Out(T_{2,3}) = \{b(3)\}$ $\rightsquigarrow T_{1,3} \perp T_{2,3}$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Bernstein conditions

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Ber66].

We can check that:

- $Out(T_{1,1}) \cap In(T_{1,2}) = \{x(1)\}$ $\rightsquigarrow T_{1,1} \perp T_{1,2}$.
- $Out(T_{1,3}) \cap Out(T_{2,3}) = \{b(3)\}$ $\rightsquigarrow T_{1,3} \perp T_{2,3}$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Bernstein conditions

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Ber66].

We can check that:

- $Out(T_{1,1}) \cap In(T_{1,2}) = \{x(1)\}$ $\rightsquigarrow T_{1,1} \perp T_{1,2}$.
- $Out(T_{1,3}) \cap Out(T_{2,3}) = \{b(3)\}$ $\rightsquigarrow T_{1,3} \perp T_{2,3}$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Bernstein conditions

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Ber66].

We can check that:

- $Out(T_{1,1}) \cap In(T_{1,2}) = \{x(1)\}$ $\rightsquigarrow T_{1,1} \perp T_{1,2}$.
- $Out(T_{1,3}) \cap Out(T_{2,3}) = \{b(3)\}$ $\rightsquigarrow T_{1,3} \perp T_{2,3}$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

Precedence

If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.
More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i + 1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

A dependence graph G is used.

$(e : T \rightarrow T') \in G$ means that T' can start only if T has already been finished.
T is a predecessor of T'.

Transitivity arc are generally omitted.

Precedence

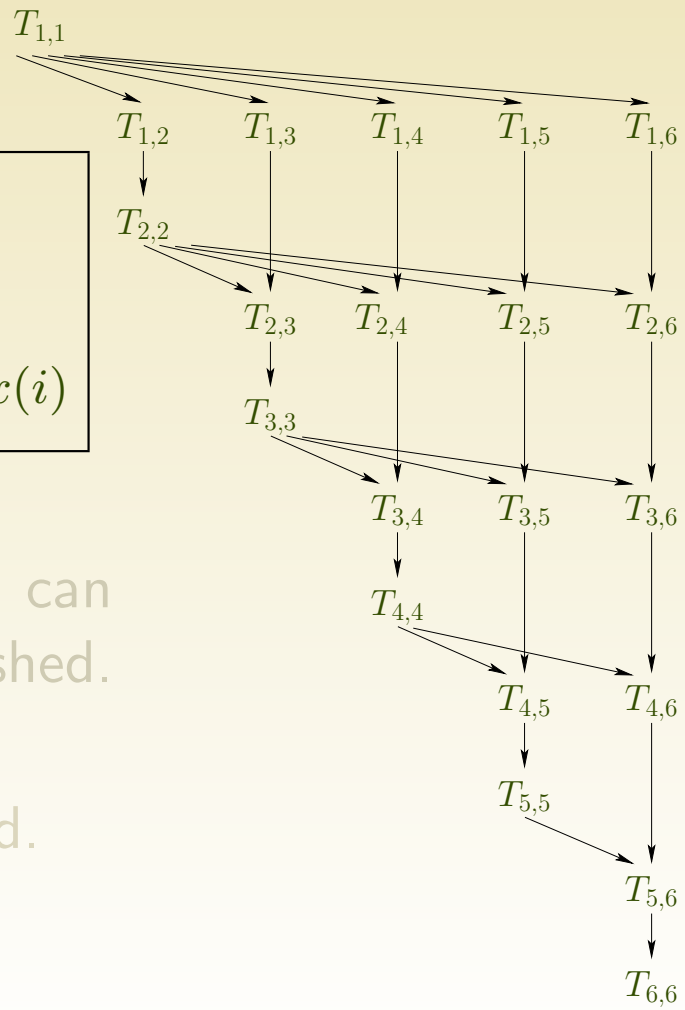
If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.
More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

A dependence graph G is used.

$(e : T \rightarrow T') \in G$ means that T' can start only if T has already been finished.
T is a predecessor of T'.

Transitivity arc are generally omitted.



Precedence

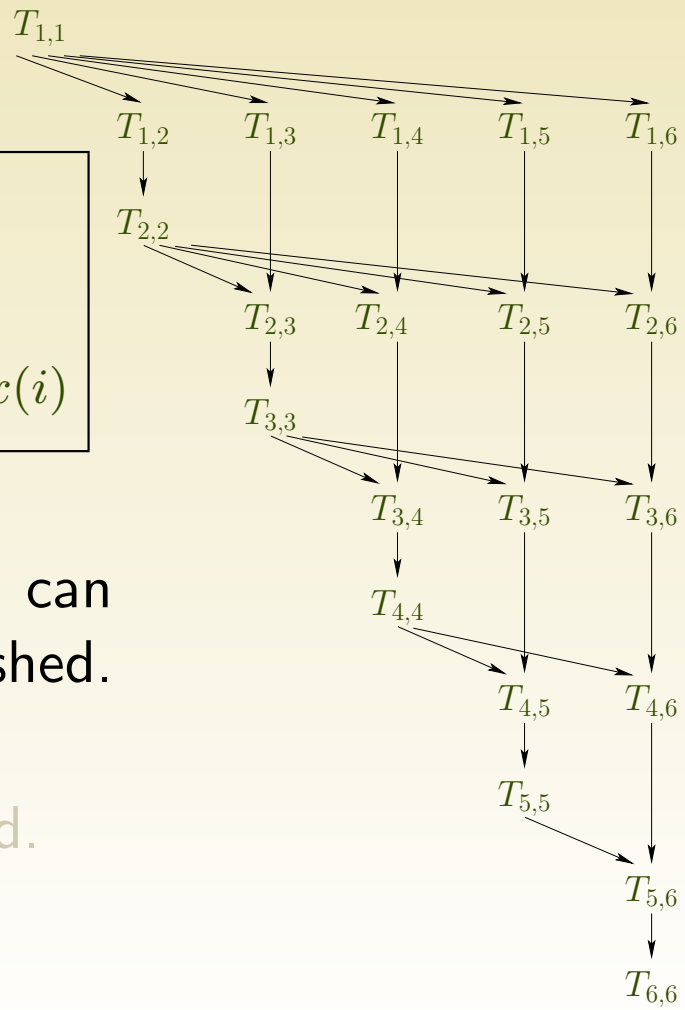
If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.
More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

A dependence graph G is used.

$(e : T \rightarrow T') \in G$ means that T' can start only if T has already been finished.
T is a predecessor of T'.

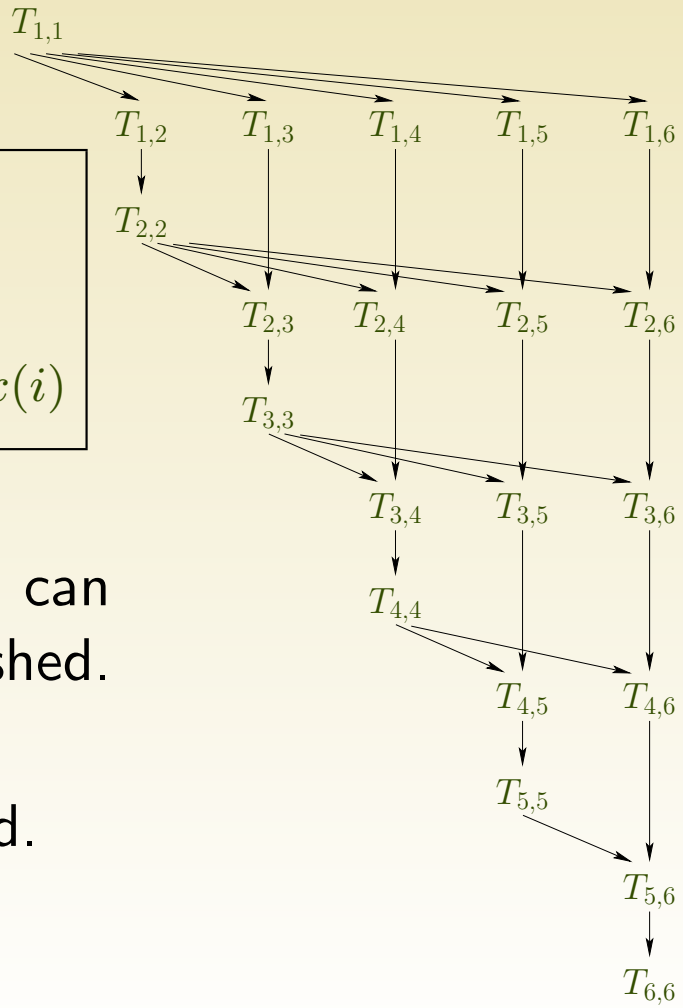
Transitivity arc are generally omitted.



Precedence

If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.
More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ to N **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i + 1$ to N **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$



A dependence graph G is used.

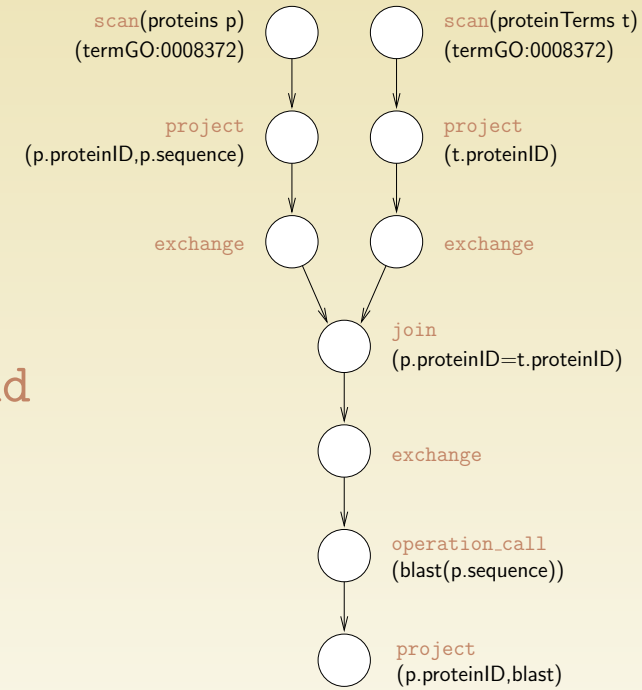
$(e : T \rightarrow T') \in G$ means that T' can start only if T has already been finished.
T is a predecessor of T'.

Transitivity arc are generally omitted.

Coarse-grain task graph

Task-graph do not necessarily come from instruction-level analysis.

```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



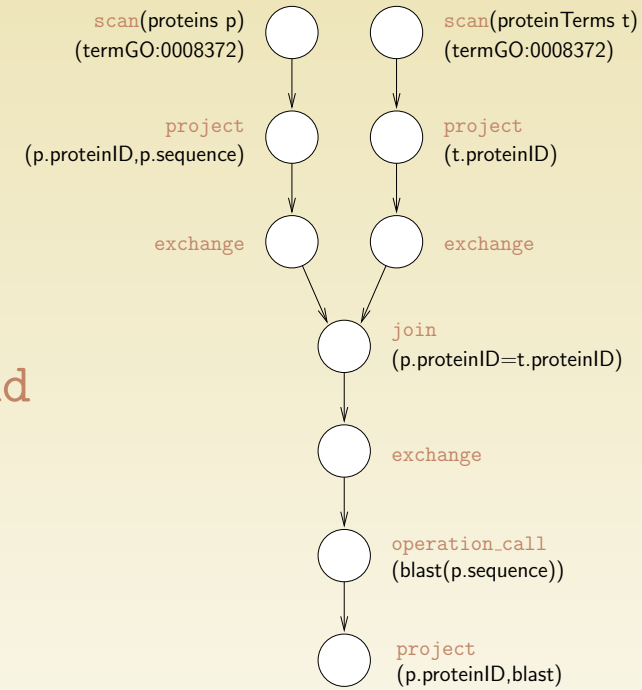
- Each task may be parallel, preemptable, divisible, . . .
- Each edge depicts a dependency i.e. most of the times some data to transfer.

In the following, we will focus on simple models.

Coarse-grain task graph

Task-graph do not necessarily come from instruction-level analysis.

```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



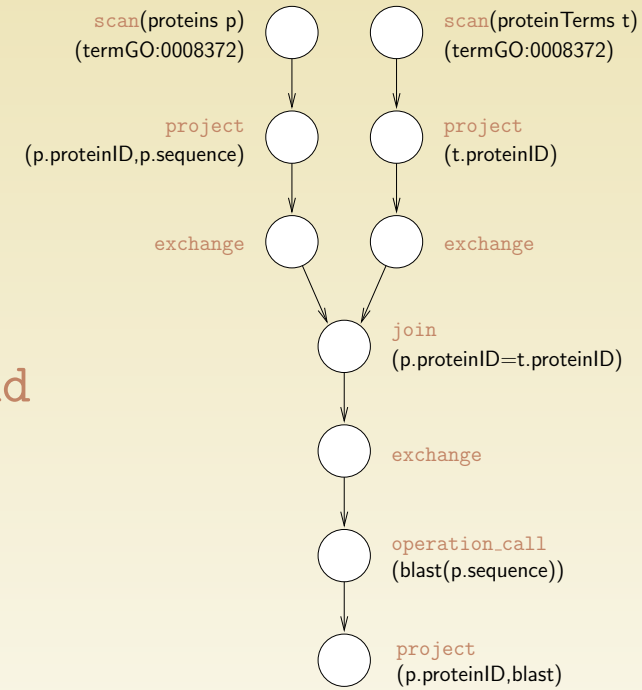
- Each task may be parallel, preemptable, divisible, . . .
- Each edge depicts a dependency i.e. most of the times some data to transfer.

In the following, we will focus on simple models.

Coarse-grain task graph

Task-graph do not necessarily come from instruction-level analysis.

```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



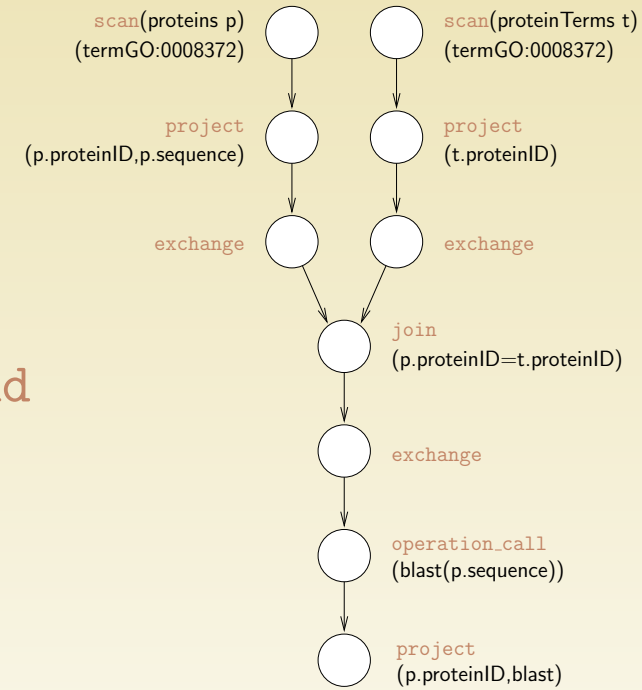
- Each task may be parallel, preemptable, divisible, . . .
- Each edge depicts a dependency i.e. most of the times some data to transfer.

In the following, we will focus on simple models.

Coarse-grain task graph

Task-graph do not necessarily come from instruction-level analysis.

```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



- Each task may be parallel, preemptable, divisible, . . .
- Each edge depicts a dependency i.e. most of the times some data to transfer.

In the following, we will focus on simple models.

Outline

Task system

Definition (Task system).

A task system is an directed graph $G = (V, E, w)$ where :

- V is the set of tasks (V is finite)
- E represent the dependence constraints:

$$e = (u, v) \in E \text{ iff } u \prec v$$

- $w : V \to \mathbb{N}^*$ is a time function that give the weight (or duration) of each task.

We could set $w(T_{i,j}) = 1$ but also decide that performing a division is more expensive than a multiplication followed by an addition.

Task system

Definition (Task system).

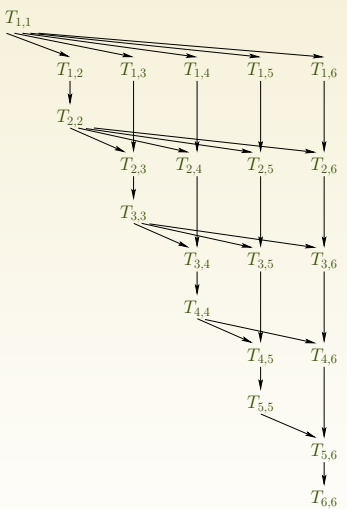
A task system is an directed graph $G = (V, E, w)$ where :

- V is the set of tasks (V is finite)
- E represent the dependence constraints:

$$e = (u, v) \in E \text{ iff } u \prec v$$

- $w : V \to \mathbb{N}^*$ is a time function that give the weight (or duration) of each task.

We could set $w(T_{i,j}) = 1$ but also decide that performing a division is more expensive than a multiplication followed by an addition.



Schedule and Allocation

Definition (Schedule).

A schedule of a task system $G = (V, E, w)$ is a time function $\sigma : V \to \mathbb{N}^*$ such that:

$$\forall (u, v) \in E,\ \sigma(u) + w(u) \leqslant \sigma(v)$$

Let us denote by $\mathcal{P} = \{P_1, \ldots, P_p\}$ the set of processors.

Definition (Allocation).

An allocation of a task system $G = (V, E, w)$ is a function $\pi : V \to \mathcal{P}$ such that:

$$\pi(T) = \pi(T') \Leftrightarrow \begin{cases} \sigma(T) + w(T) \leqslant \sigma(T') \text{ or} \\ \sigma(T') + w(T') \leqslant \sigma(T) \end{cases}$$

Depending on the application and platform model, much more complex definitions can be proposed.

Schedule and Allocation

Definition (Schedule).

A schedule of a task system $G = (V, E, w)$ is a time function $\sigma : V \to \mathbb{N}^*$ such that:

$$\forall (u, v) \in E,\ \sigma(u) + w(u) \leqslant \sigma(v)$$

Let us denote by $\mathcal{P} = \{P_1, \ldots, P_p\}$ the set of processors.

Definition (Allocation).

An allocation of a task system $G = (V, E, w)$ is a function $\pi : V \to \mathcal{P}$ such that:

$$\pi(T) = \pi(T') \Leftrightarrow \begin{cases} \sigma(T) + w(T) \leqslant \sigma(T') \text{ or} \\ \sigma(T') + w(T') \leqslant \sigma(T) \end{cases}$$

Depending on the application and platform model, much more complex definitions can be proposed.

Schedule and Allocation

Definition (Schedule).

A schedule of a task system $G = (V, E, w)$ is a time function $\sigma : V \to \mathbb{N}^*$ such that:

$$\forall (u, v) \in E,\ \sigma(u) + w(u) \leqslant \sigma(v)$$

Let us denote by $\mathcal{P} = \{P_1, \ldots, P_p\}$ the set of processors.

Definition (Allocation).

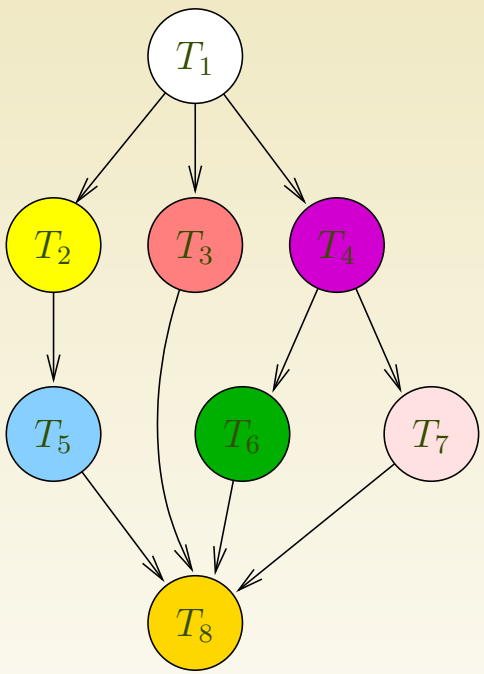
An allocation of a task system $G = (V, E, w)$ is a function $\pi : V \to \mathcal{P}$ such that:

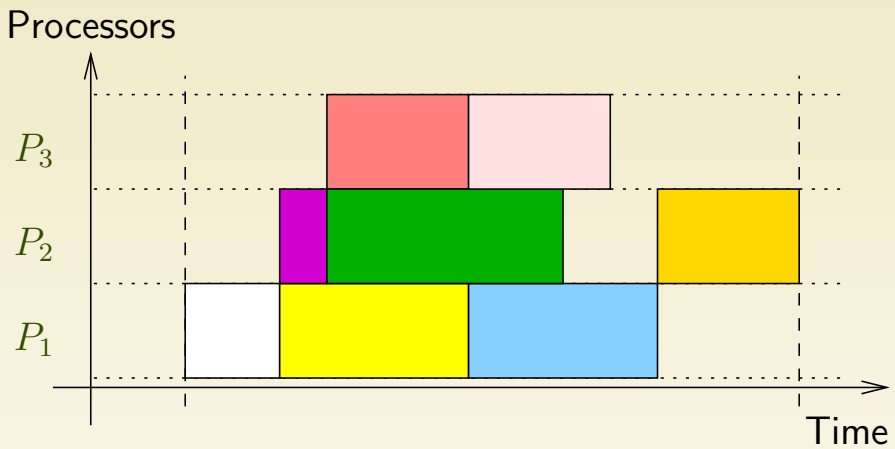
$$\pi(T) = \pi(T') \Leftrightarrow \begin{cases} \sigma(T) + w(T) \leqslant \sigma(T') \text{ or} \\ \sigma(T') + w(T') \leqslant \sigma(T) \end{cases}$$

Depending on the application and platform model, much more complex definitions can be proposed.

Gantt-chart

Manipulating functions is generally not very convenient. That is why Gantt-chart are used to depict schedules and allocations.





Basic feasibility condition

Theorem.

Let $G = (V, E, w)$ be a task system. There exists a valid schedule of G iff G has no cycle.

Sketch of the proof.

$\Rightarrow$ Assume that G has a cycle $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1$. Then $v_1 \prec v_1$ and a valid schedule σ should hold $\sigma(v_1) + w(v_1) \leqslant \sigma(v_1)$ true, which is impossible because $w(v_1) > 0$.

$\Leftarrow$ If G is acyclic, then some tasks have no predecessor. They can be scheduled first.
More precisely, we sort topologically the vertexes and schedule them one after the other on the same processor. Dependences are then fulfilled. □

Therefore all task systems we will be considering in the following are Directed Acyclic Graphs.

Basic feasibility condition

Theorem.

Let $G = (V, E, w)$ be a task system. There exists a valid schedule of G iff G has no cycle.

Sketch of the proof.

$\Rightarrow$ Assume that G has a cycle $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1$. Then $v_1 \prec v_1$ and a valid schedule σ should hold $\sigma(v_1) + w(v_1) \leqslant \sigma(v_1)$ true, which is impossible because $w(v_1) > 0$.

$\Leftarrow$ If G is acyclic, then some tasks have no predecessor. They can be scheduled first.
More precisely, we sort topologically the vertexes and schedule them one after the other on the same processor. Dependences are then fulfilled. $\square$

Therefore all task systems we will be considering in the following are Directed Acyclic Graphs.

Basic feasibility condition

Theorem.

Let $G = (V, E, w)$ be a task system. There exists a valid schedule of G iff G has no cycle.

Sketch of the proof.

$\Rightarrow$ Assume that G has a cycle $v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k \longrightarrow v_1$. Then $v_1 \prec v_1$ and a valid schedule σ should hold $\sigma(v_1) + w(v_1) \leqslant \sigma(v_1)$ true, which is impossible because $w(v_1) > 0$.

$\Leftarrow$ If G is acyclic, then some tasks have no predecessor. They can be scheduled first.
More precisely, we sort topologically the vertexes and schedule them one after the other on the same processor. Dependences are then fulfilled. $\square$

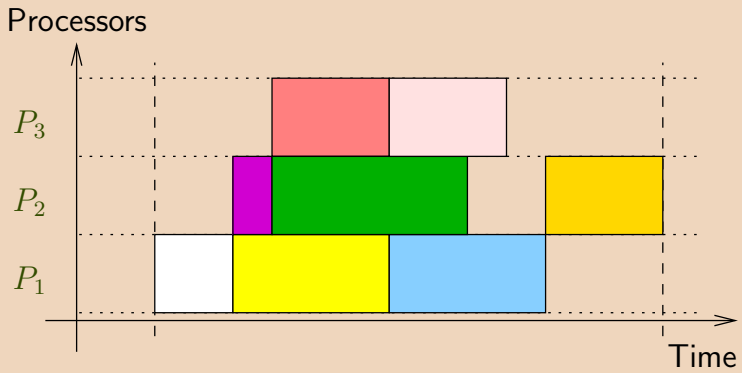
Therefore all task systems we will be considering in the following are Directed Acyclic Graphs.

Makespan

Definition (Makespan).

The makespan of a schedule is the total execution time :

$$MS(\sigma) = \max_{v \in V}\{\sigma(v) + w(v)\} - \min_{v \in V}\{\sigma(v)\} \ .$$



The makespan is also often referred as $C_{\max}$ in the literature.

$$C_{\max} = \max_{v \in V} C_v$$

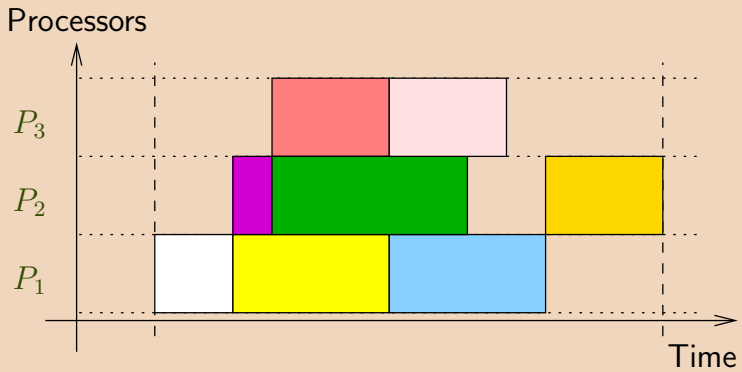
- $Pb(p)$: find a schedule with the smallest possible makespan, using at most p processors. $MS_{opt}(p)$ denotes the optimal makespan using only p processors.
- $Pb(\infty)$: find a schedule with the smallest makespan when the number of processors that can be used is not bounded.
  We note $MS_{opt}(\infty)$ the corresponding makespan.

Makespan

Definition (Makespan).

The makespan of a schedule is the total execution time :

$$MS(\sigma) = \max_{v \in V}\{\sigma(v) + w(v)\} - \min_{v \in V}\{\sigma(v)\} \ .$$



The makespan is also often referred as $C_{\max}$ in the literature.

$$C_{\max} = \max_{v \in V} C_v$$

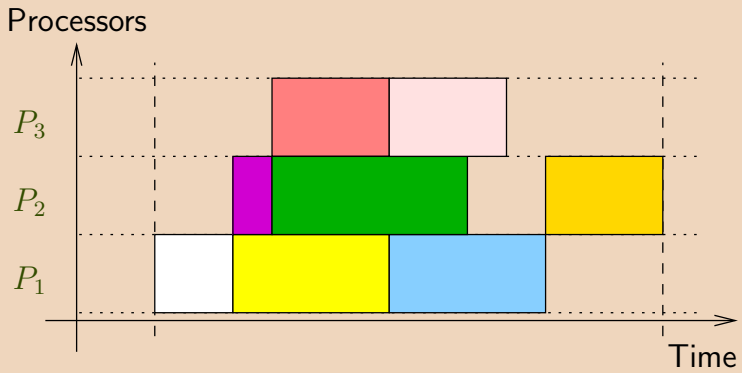
- $Pb(p)$: find a schedule with the smallest possible makespan, using at most p processors. $MS_{opt}(p)$ denotes the optimal makespan using only p processors.
- $Pb(\infty)$: find a schedule with the smallest makespan when the number of processors that can be used is not bounded.
  We note $MS_{opt}(\infty)$ the corresponding makespan.

Makespan

Definition (Makespan).

The makespan of a schedule is the total execution time :

$$MS(\sigma) = \max_{v \in V}\{\sigma(v) + w(v)\} - \min_{v \in V}\{\sigma(v)\} \ .$$



The makespan is also often referred as $C_{\max}$ in the literature.

$$C_{\max} = \max_{v \in V} C_v$$

- $Pb(p)$: find a schedule with the smallest possible makespan, using at most p processors. $MS_{opt}(p)$ denotes the optimal makespan using only p processors.
- $Pb(\infty)$: find a schedule with the smallest makespan when the number of processors that can be used is not bounded.
  We note $MS_{opt}(\infty)$ the corresponding makespan.

Critical path

Let $\Phi = (T_1, T_2, \ldots, T_n)$ be a path in G. w can be extended to paths in the following way :

$$w(\Phi) = \sum_{i=1}^{n} w(T_i)$$

Lemma.

Let $G = (V, E, w)$ be a DAG and σ_p a schedule of G using p processors. For any path Φ in G, we have $MS(\sigma_p) \geqslant w(\Phi)$.

Proof.

Let $\Phi = (T_1, T_2, \ldots, T_n)$ be a path in G: $(T_i, T_{i+1}) \in E$ for $1 \leqslant i < n$. Therefore we have $\sigma_p(T_i) + w(T_i) \leqslant \sigma_p(T_{i+1})$ for $1 \leqslant i < n$, hence

$$MS(\sigma_p) \geqslant w(T_n) + \sigma_p(T_n) - \sigma_p(T_1) \geqslant \sum_{i=1}^{n} w(T_i) = w(\Phi)\ . \quad \square$$

Critical path

Let $\Phi = (T_1, T_2, \ldots, T_n)$ be a path in G. w can be extended to paths in the following way :

$$w(\Phi) = \sum_{i=1}^{n} w(T_i)$$

Lemma.

Let $G = (V, E, w)$ be a DAG and σ_p a schedule of G using p processors. For any path Φ in G, we have $MS(\sigma_p) \geqslant w(\Phi)$.

Proof.

Let $\Phi = (T_1, T_2, \ldots, T_n)$ be a path in G: $(T_i, T_{i+1}) \in E$ for $1 \leqslant i < n$. Therefore we have $\sigma_p(T_i) + w(T_i) \leqslant \sigma_p(T_{i+1})$ for $1 \leqslant i < n$, hence

$$MS(\sigma_p) \geqslant w(T_n) + \sigma_p(T_n) - \sigma_p(T_1) \geqslant \sum_{i=1}^{n} w(T_i) = w(\Phi) \; . \qquad \square$$

Critical path

Let $\Phi = (T_1, T_2, \ldots, T_n)$ be a path in G. w can be extended to paths in the following way :

$$w(\Phi) = \sum_{i=1}^{n} w(T_i)$$

Lemma.

Let $G = (V, E, w)$ be a DAG and σ_p a schedule of G using p processors. For any path Φ in G, we have $MS(\sigma_p) \geqslant w(\Phi)$.

Proof.

Let $\Phi = (T_1, T_2, \ldots, T_n)$ be a path in G: $(T_i, T_{i+1}) \in E$ for $1 \leqslant i < n$. Therefore we have $\sigma_p(T_i) + w(T_i) \leqslant \sigma_p(T_{i+1})$ for $1 \leqslant i < n$, hence

$$MS(\sigma_p) \geqslant w(T_n) + \sigma_p(T_n) - \sigma_p(T_1) \geqslant \sum_{i=1}^{n} w(T_i) = w(\Phi) \; . \qquad \square$$

Speed-up and Efficiency

Definition.

Let $G = (V, E, w)$ be a DAG and σ_p a schedule of G using only p processors:

- Speed-up: $s(\sigma_p) = \dfrac{Seq}{MS(\sigma_p)}$, where $Seq = MS_{opt}(1) = \sum_{v \in V} w(v)$.
- Efficiency: $e(\sigma_p) = \dfrac{s(\sigma_p)}{p} = \dfrac{Seq}{p \times MS(\sigma_p)}$.

Theorem.

Let $G = (V, E, w)$ be a DAG. For any schedule σ_p using p processors:

$$0 \leqslant e(\sigma_p) \leqslant 1 \, .$$

Speed-up and Efficiency

Definition.

Let $G = (V, E, w)$ be a DAG and σ_p a schedule of G using only p processors:

- Speed-up: $s(\sigma_p) = \dfrac{Seq}{MS(\sigma_p)}$, where $Seq = MS_{opt}(1) = \sum_{v \in V} w(v)$.
- Efficiency: $e(\sigma_p) = \dfrac{s(\sigma_p)}{p} = \dfrac{Seq}{p \times MS(\sigma_p)}$.

Theorem.

Let $G = (V, E, w)$ be a DAG. For any schedule σ_p using p processors:

$$0 \leqslant e(\sigma_p) \leqslant 1 \ .$$

Speed-up and Efficiency

Definition.

Let $G = (V, E, w)$ be a DAG and σ_p a schedule of G using only p processors:

- Speed-up: $s(\sigma_p) = \dfrac{Seq}{MS(\sigma_p)}$, where $Seq = MS_{opt}(1) = \sum_{v \in V} w(v)$.
- Efficiency: $e(\sigma_p) = \dfrac{s(\sigma_p)}{p} = \dfrac{Seq}{p \times MS(\sigma_p)}$.

Theorem.

Let $G = (V, E, w)$ be a DAG. For any schedule σ_p using p processors:

$$0 \leqslant e(\sigma_p) \leqslant 1 \, .$$

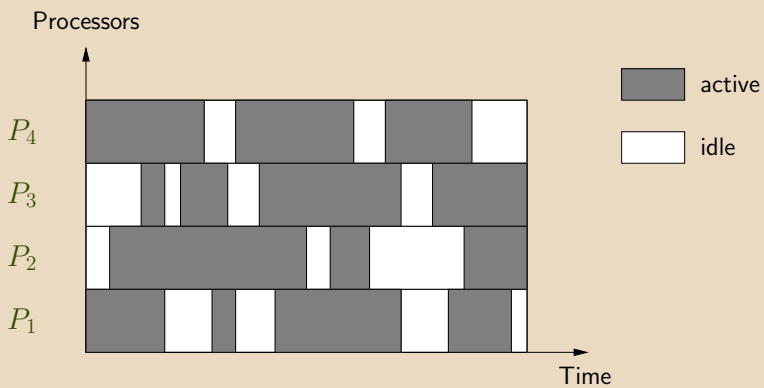
Speed-up and Efficiency (Cont'd)

Theorem.

Let $G = (V, E, w)$ be a DAG. For any schedule σ_p using p processors:

$$0 \leqslant e(\sigma_p) \leqslant 1 \ .$$

Proof.



Let $Idle$ denote the total idle time. $Seq + Idle$ is then equal to the total surface of the rectangle, i.e. $p \times MS(\sigma_p)$.

Therefore $e(\sigma_p) = \dfrac{Seq}{p \times MS(\sigma_p)} \leqslant 1.$

□

The speed-up is thus bound the number of processors. No supra-linear speed-up in our model !

A trivial result

Theorem.

Let $G = (V, E, w)$ be a DAG. We have

$$Seq = MS_{opt}(1) \geqslant \ldots \geqslant MS_{opt}(p) \geqslant MS_{opt}(p+1) \geqslant \ldots \geqslant MS_{opt}(\infty) \ .$$

Allowing to use more processors cannot hurt.

However, using more processors may hurt, especially in a model where communications are taken into account.

If we define $MS'(p)$ as the smallest makespan of schedules using exactly p processors, we may have $MS'(p) > MS'(p')$ with $p < p'$.

A trivial result

Theorem.

Let $G = (V, E, w)$ be a DAG. We have

$$Seq = MS_{opt}(1) \geqslant \ldots \geqslant MS_{opt}(p) \geqslant MS_{opt}(p+1) \geqslant \ldots \geqslant MS_{opt}(\infty) \ .$$

Allowing to use more processors cannot hurt.

However, using more processors may hurt, especially in a model where communications are taken into account.

If we define $MS'(p)$ as the smallest makespan of schedules using exactly p processors, we may have $MS'(p) > MS'(p')$ with $p < p'$.

A trivial result

Theorem.

Let $G = (V, E, w)$ be a DAG. We have

$$Seq = MS_{opt}(1) \geqslant \ldots \geqslant MS_{opt}(p) \geqslant MS_{opt}(p+1) \geqslant \ldots \geqslant MS_{opt}(\infty) \ .$$

Allowing to use more processors cannot hurt.

However, using more processors may hurt, especially in a model where communications are taken into account.

If we define $MS'(p)$ as the smallest makespan of schedules using exactly p processors, we may have $MS'(p) > MS'(p')$ with $p < p'$.

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - prmp: preemption;
  - prec: precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$: precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - pmtn: preemption;
  - prec: precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$: precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$: tasks have unit processing times, or comprised between p and $\underline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $prec$: precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$, $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$: all processing times are equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leq p_j \leq \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\alpha|\beta|\gamma$ [Bru98]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Outline

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1, \mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.

□

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.

$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.

Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.
$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.
Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.
$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.
Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.
$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.
Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.
$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.
Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.
$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.
Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

Complexity results

If we have an infinite number of processors, the "as-soon-as-possible" schedule is optimal. $MS_{opt}(\infty) = \max_{\Phi \text{ path in } G} w(\Phi)$.

- $P, 2||C_{\max}$ is weakly NP-complete (2-Partition);

Proof.

By reduction to 2-Partition: can $\mathcal{A} = \{a_1, \ldots, a_n\}$ be partitioned into two sets $\mathcal{A}_1$, $\mathcal{A}_2$ such $\sum_{a \in \mathcal{A}_1} a = \sum_{a \in \mathcal{A}_2} a$.
$p = 2$, $G = (V, E, w$ with $V = \{v_1, \ldots, v_n\}$, $E = \emptyset$ and $w(v_i) = a_i, \forall 1 \leqslant i \leqslant n$.
Finding a schedule of makespan smaller or equal to $\frac{1}{2} \sum_i a_i$ is equivalent to solve the instance of 2-Partition. □

- $P, 3|prec|C_{\max}$ is strongly NP-complete (3DM);
- $P|prec, p_j = 1|C_{\max}$ is strongly NP-complete (3DM);
- $P, p \geqslant 3|prec, p_j = 1|C_{\max}$ is open;
- $P, 2|prec, 1 \leqslant p_j \leqslant 2|C_{\max}$ is strongly NP-complete;

List scheduling

When simple problems are hard, we should try to find good approximation heuristics.
Natural idea: using greedy strategy like trying to allocate the most possible task at a given time-step. However at some point we may face a choice (when there is more ready tasks than available processors).

Any strategy that does not let on purpose a processor idle is efficient.
Such a schedule is called list-schedule.

Theorem (Coffman).

Let $G = (V, E, w)$ be a DAG, p the number of processors, and σ_p a list-schedule of G.

$$MS(\sigma_p) \leqslant \left(2 - \frac{1}{p}\right) MS_{opt}(p) \ .$$

One can actually prove that this bound cannot be improved.

Most of the time, list-heuristics are based on the critical path.

List scheduling

When simple problems are hard, we should try to find good approximation heuristics.
Natural idea: using greedy strategy like trying to allocate the most possible task at a given time-step. However at some point we may face a choice (when there is more ready tasks than available processors).
Any strategy that does not let on purpose a processor idle is efficient.
Such a schedule is called list-schedule.

Theorem (Coffman).

Let $G = (V, E, w)$ be a DAG, p the number of processors, and σ_p a list-schedule of G.

$$MS(\sigma_p) \leqslant \left(2 - \tfrac{1}{p}\right) MS_{opt}(p) \ .$$

One can actually prove that this bound cannot be improved.

Most of the time, list-heuristics are based on the critical path.

List scheduling

When simple problems are hard, we should try to find good approximation heuristics.
Natural idea: using greedy strategy like trying to allocate the most possible task at a given time-step. However at some point we may face a choice (when there is more ready tasks than available processors).
Any strategy that does not let on purpose a processor idle is efficient.
Such a schedule is called list-schedule.

Theorem (Coffman).

Let $G = (V, E, w)$ be a DAG, p the number of processors, and σ_p a list-schedule of G.

$$MS(\sigma_p) \leqslant \left(2 - \tfrac{1}{p}\right) MS_{opt}(p) \ .$$

One can actually prove that this bound cannot be improved.

Most of the time, list-heuristics are based on the critical path.

Taking communications into account

A very simple model (things are already complicated enough): the macro-data flow model. If there is some data-dependence between T and T', the communication cost is

$$c(T,T') = \begin{cases} 0 & \text{if } \mathrm{alloc}(T) = \mathrm{alloc}(T') \\ c(T,T') & \text{otherwise} \end{cases}$$

Definition.

A DAG with communication cost (say cDAG) is a directed acyclic graph $G = (V, E, w, c)$ where vertexes represent tasks and edges represent dependence constraints. $w : V \to \mathbb{N}^*$ is the computation time function and $c : E \to \mathbb{N}^*$ is the communication time function. Any valid schedule has to respect the dependence constraints.

$$\forall e = (v, v') \in E,$$

$$\begin{cases} \sigma(v) + w(v) \leqslant \sigma(v') & \text{if } \mathrm{alloc}(v) = \mathrm{alloc}(v') \\ \sigma(v) + w(v) + c(v; v') \leqslant \sigma(v') & \text{otherwise.} \end{cases}$$

Taking communications into account

A very simple model (things are already complicated enough): the macro-data flow model. If there is some data-dependence between T and T', the communication cost is

$$c(T,T') = \begin{cases} 0 & \text{if } \mathrm{alloc}(T) = \mathrm{alloc}(T') \\ c(T,T') & \text{otherwise} \end{cases}$$

Definition.

A DAG with communication cost (say cDAG) is a directed acyclic graph $G = (V, E, w, c)$ where vertexes represent tasks and edges represent dependence constraints. $w : V \to \mathbb{N}^*$ is the computation time function and $c : E \to \mathbb{N}^*$ is the communication time function. Any valid schedule has to respect the dependence constraints.

$$\forall e = (v, v') \in E,$$

$$\begin{cases} \sigma(v) + w(v) \leqslant \sigma(v') & \text{if } \mathrm{alloc}(v) = \mathrm{alloc}(v') \\ \sigma(v) + w(v) + c(v; v') \leqslant \sigma(v') & \text{otherwise.} \end{cases}$$

Taking communications into account (cont'd)

Even Pb(∞) is NP-complete !!!

You constantly have to figure out whether you should use more processor (but then pay more fore communications) or not. Finding the good trade-off is a real challenge.
$4/3$-approximation if all communication times are smaller than computation times.
Finding guaranteed approximations for other settings is really hard, but really useful.

That must be the reason why there is so much research about it !

Conclusion

Most of the time, the only thing we can done is to compare heuristics. There are three ways of doing that:

- Theory: being able to guarantee your heuristic;
- Experiment: Generating random graphs and/or typical application graphs along with platform graphs to compare your heuristics.
- Smart: proving that your heuristic is optimal for a particular class of graphs (fork, join, fork-join, bounded degree, ...).

However, remember that the first thing to do is to look whether your problem is NP-complete or not. Who knows? You may be lucky...

Bibliography


A.J. Bernstein.
Analysis of programs for parallel processing.
IEEE Transactions on Electronic Computers, 15:757–762, October 1966.


Peter Brucker.
Scheduling Algorithms.
Springer, Heidelberg, 2 edition, 1998.