The two-machine flowshop total completion time problem: A branch-and-bound based on network-flow formulation

Boris Detienne[1], Ruslan Sadykov[1], Shunji Tanaka[2]
1 : Team Inria RealOpt, University of Bordeaux, France
2 : Department of Electrical Engineering, Kyoto University, Japan

Journées GOThA/Bermudes

26-27 septembre 2017

Outline

1. Introduction
   - Problem description
   - Literature
   - Contribution
2. Lower bounds
   - Network flow formulation
   - Extended network flow formulation
3. Branch-and-bound
4. Numerical results

Two-machine flow-shop problem $F2|ST_{SI}|\sum C_i$

Input data: A set I of n jobs composed of 2 operations

- The first operation is processed on machine 1, the second on machine 2
- For all $i \in I$, s_i^2 is the sequence-independent setup time on machine 2
- Assumption: data are integer and deterministic

Constraints

- Each machine can process only one operation at a time
- Operations of a same job cannot be processed simultaneously

Objective

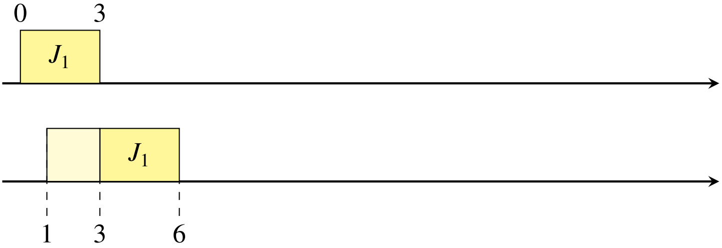
Find a schedule that minimizes the sum of the completion times of the jobs on the second machine.

Example

i	1	2	3
p_i^1	3	7	2
p_i^2	3	4	3
s_i^2	2	2	3

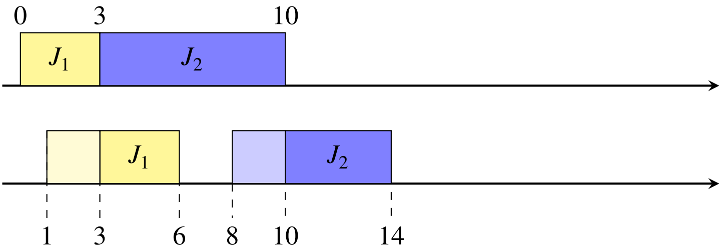
Example

i	1	2	3
p_i^1	3	7	2
p_i^2	3	4	3
s_i^2	2	2	3



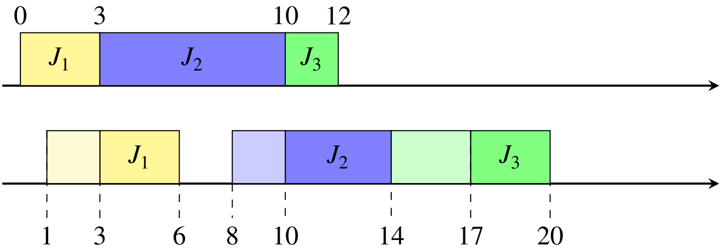
Example

i	1	2	3
p_i^1	3	7	2
p_i^2	3	4	3
s_i^2	2	2	3



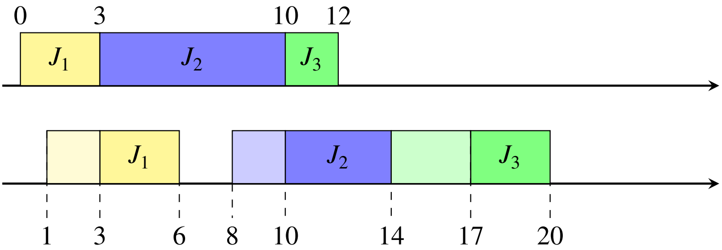
Example

i	1	2	3
p_i^1	3	7	2
p_i^2	3	4	3
s_i^2	2	2	3



Example

i	1	2	3
p_i^1	3	7	2
p_i^2	3	4	3
s_i^2	2	2	3



Cost of the schedule: $6 + 14 + 20 = 40$

Properties of the problem

Complexity

Strongly NP-hard [Conway et al., 1967]

Dominating solutions

There is a least one optimal schedule that is:

- active (operations are performed as soon as possible, no unforced idle time)
- such that the sequences of the jobs on both machines are the same (permutation schedule) [Conway et al., 1967, Allahverdi et al., 1999]

$\longrightarrow$ The problem comes to find **one** optimal sequence of jobs.

Literature

Lower bounds and exact algorithms

- **L.B.: Single machine problems**
  [Ignall and Schrage, 1965], [Ahmadi and Bagchi, 1990], [Della Croce et al., 1996], [Allahverdi, 2000]
  Branch-and-bound, up to 10, 15 and 30 jobs $(p_i \leq 20)$*, 20 jobs* $(p_i \leq 100)$
- **L.B.: Lagrangian relaxation of precedence constraints**
  [van de Velde, 1990], [Della Croce et al, 2002], [Gharbi et al., 2013]
  Branch-and-bound, up to 20 and 45 jobs $(p_i \leq 10)$
- **L.B.: linear relaxation of a positional/assignment model**
  [Akkan and Karabati, 2004], [Hoogeven et al., 2006], [Haouari and Kharbeche, 2013], [Gharbi et al., 2013] *: 35 jobs* $(p_i \leq 100)$
- **L.B.: Lagrangian relaxation of the job cardinality ctr., flow model**
  [Akkan and Karabati, 2004]
  Branch-and-bound, up to 60 jobs $(p_i \leq 10)$*, 45 jobs* $(p_i \leq 100)$

Contribution

Branch-and-bound based on the network flow model of [Akkan and Karabati, 2004]

Improvements

Stronger lower bound by using a **larger size network**

- Advantages
  - Stronger Lagrangian relaxation bound
  - Allows integration of dominance rules inside the network
- Disadvantages
  - (Too) high memory and CPU time requirements
    → Reduction of the size of the network using Lagrangian cost variable fixing

Extension to sequence-independent setup times

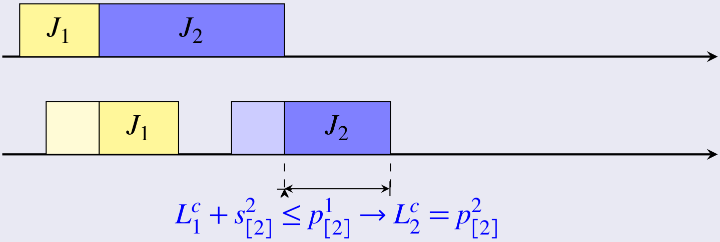
1. Introduction
2. Lower bounds
   - Network flow formulation
   - Extended network flow formulation
3. Branch-and-bound
4. Numerical results

Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

- $C^m_{[k]}$: completion time of the job in position k on machine m
- L^c_k: time **lag** elapsed between the completion of the job in position k on machines 1 and 2

$$L^c_k = C^2_{[k]} - C^1_{[k]} = \max\left\{0, L^c_{k-1} + s^2_{[k]} - p^1_{[k]}\right\} + p^2_{[k]}$$

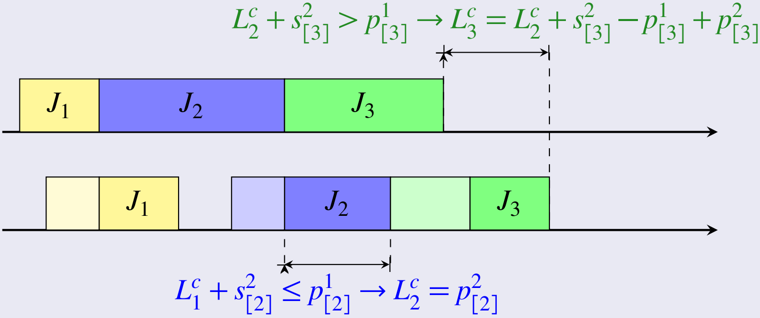


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

- $C^m_{[k]}$: completion time of the job in position k on machine m
- L^c_k: time **lag** elapsed between the completion of the job in position k on machines 1 and 2

$$L^c_k = C^2_{[k]} - C^1_{[k]} = \max\left\{0, L^c_{k-1} + s^2_{[k]} - p^1_{[k]}\right\} + p^2_{[k]}$$



Lag-based models

Formulating the objective function

Minimizing the sum of completion times:

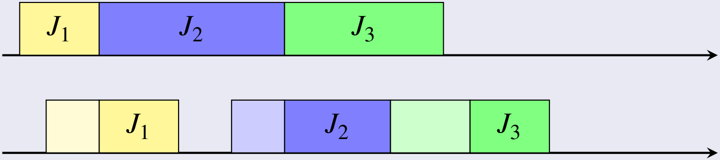
$$\sum_{k=1}^{n} C^2_{[k]} = \sum_{k=1}^{n} \left(C^1_{[k]} + L^c_k \right)$$
$$= \sum_{k=1}^{n} \left(\sum_{r=1}^{k} p^1_{[r]} + L^c_k \right)$$
$$= \sum_{k=1}^{n} \left((n-k+1) p^1_{[k]} + L^c_k \right)$$

Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

$L^c_k = C^2_{[k]} - C^1_{[k]}$: time **lag** elapsed between the completion of the job in position k on machine 1 and on machine 2

Total completion time - Similar to $1||\sum_i C_i$

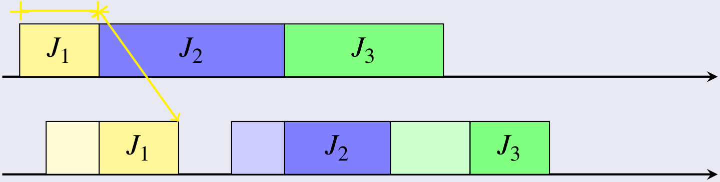


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

$L_k^c = C_{[k]}^2 - C_{[k]}^1$: time **lag** elapsed between the completion of the job in position k on machine 1 and on machine 2

Total completion time - Similar to $1||\sum_i C_i$

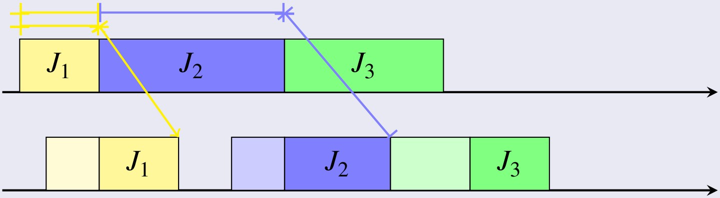


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

$L_k^c = C^2_{[k]} - C^1_{[k]}$: time **lag** elapsed between the completion of the job in position k on machine 1 and on machine 2

Total completion time - Similar to $1||\sum_i C_i$

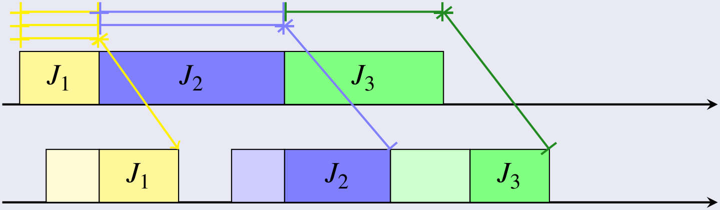


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

$L_k^c = C_{[k]}^2 - C_{[k]}^1$: time **lag** elapsed between the completion of the job in position k on machine 1 and on machine 2

Total completion time - Similar to $1||\sum_i C_i$

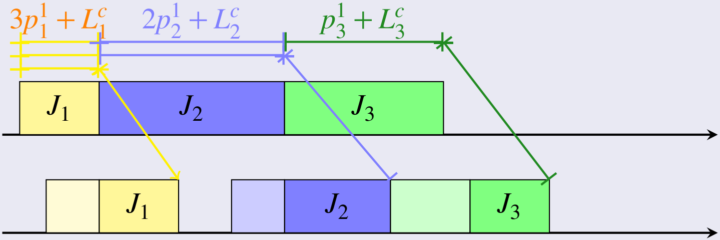


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

$L_k^c = C_{[k]}^2 - C_{[k]}^1$: time **lag** elapsed between the completion of the job in position k on machine 1 and on machine 2

Total completion time - Similar to $1||\sum_i C_i$

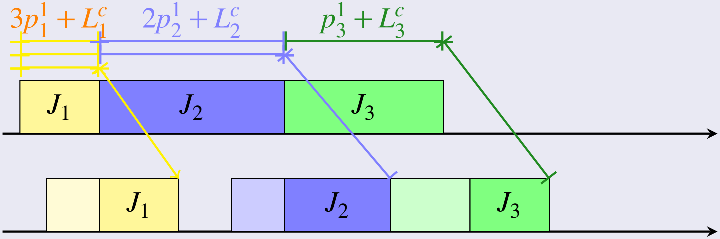


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

$L_k^c = C^2_{[k]} - C^1_{[k]}$: time **lag** elapsed between the completion of the job in position k on machine 1 and on machine 2

Total completion time

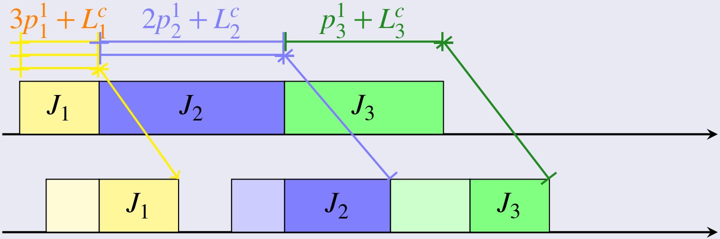


Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

Recursive formula for lag: $L_k^c = \max\left\{0, L_{k-1}^c + s_{[k]}^2 - p_{[k]}^1\right\} + p_{[k]}^2$

Total completion time



Network flow formulation [Akkan et Karabati, 2004]

Lag-based models

Cost: $\sum_{k=1}^{n} C^2_{[k]} = \sum_{k=1}^{n} \left((n-k+1)p^1_{[k]} + L^c_k\right)$

The contribution of a job to the objective function only depends on:

- Its position in the sequence
- Its lag, which is directly deduced from the lag of the preceding job

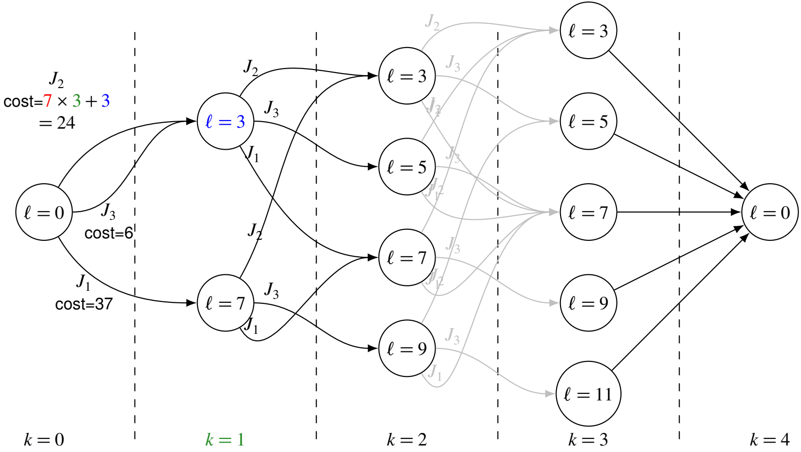
Structure of the network

- One node ≡ a pair (position, lag)
- One arc ≡ the processing of a job
  - initial node determines the position
  - terminal node determines the lag

→ The cost of an arc is the corresponding contribution to the objective function

Network flow formulation [Akkan et Karabati, 2004]: G_1

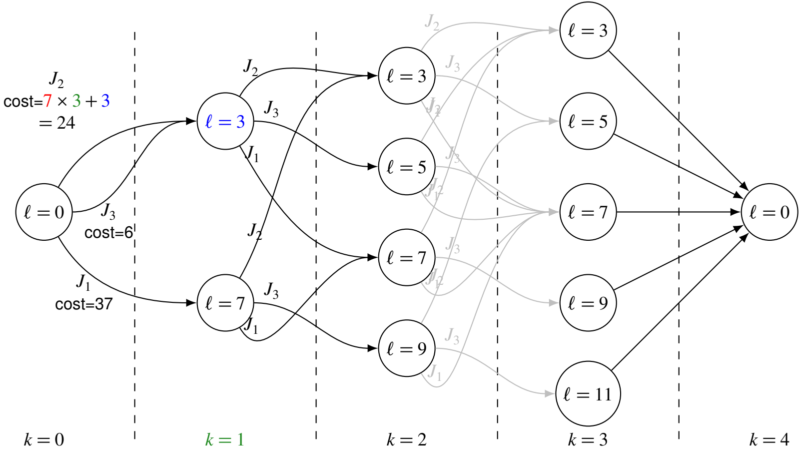
$$p_1 = (10, 7); \quad p_2 = (7, 3); \quad p_3 = (1, 3)$$



Shortest path + Each job is processed exactly once

Network flow formulation [Akkan et Karabati, 2004]: G_1

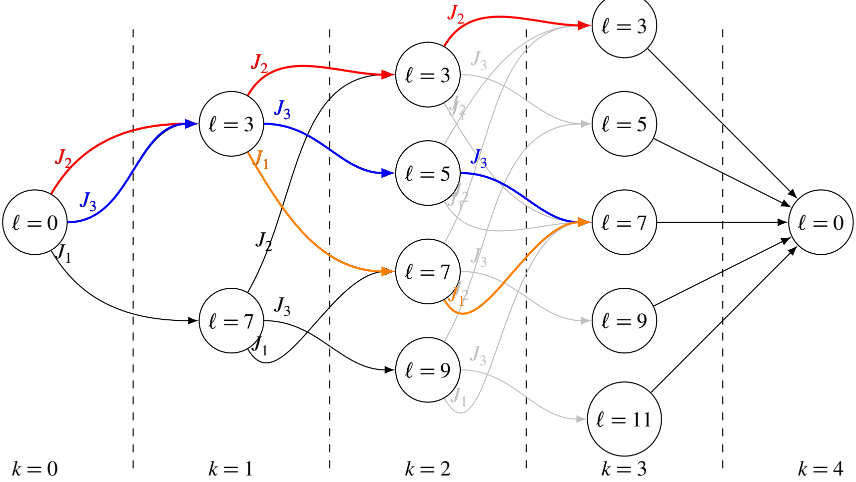
$$p_1 = (10, 7); \quad p_2 = (7, 3); \quad p_3 = (1, 3)$$



Shortest path + ~~Each job is processed once~~ → L.B. by Lagrangian relaxation

Network flow formulation [Akkan et Karabati, 2004]: G_1

Disadvantage: many infeasible paths → "weak" lower bound



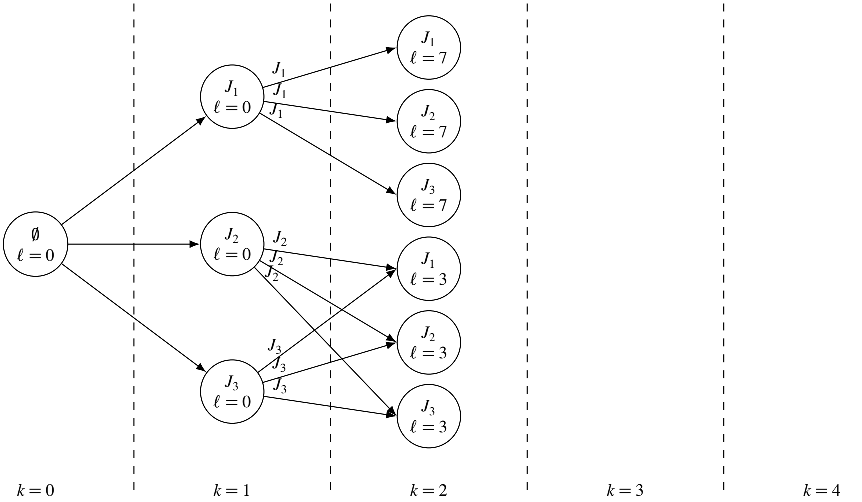
Extended network flow formulation: G_2

Structure of the network

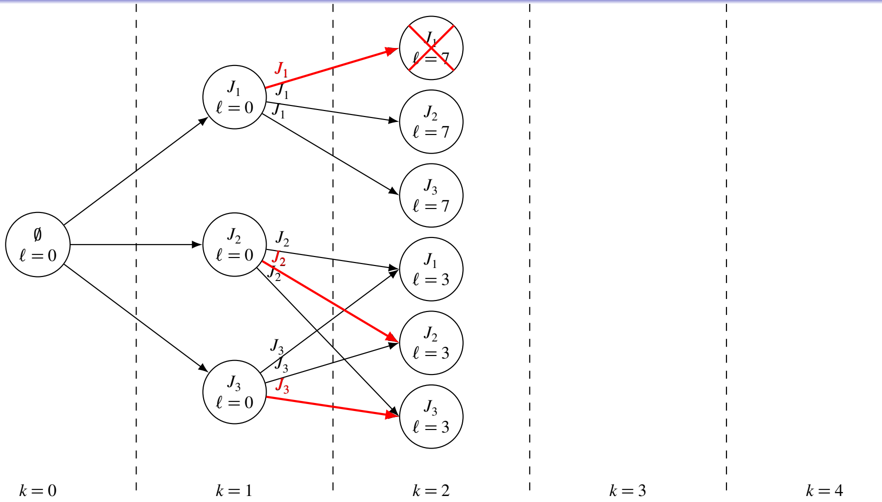
- One node ≡ a triplet (position, lag, job)
- One arc ≡ the processing of a job
  - initial node determines the position and the job
  - terminal node determines the lag and the next job

→ The cost of an arc is the corresponding contribution to the objective function

Extended network G_2

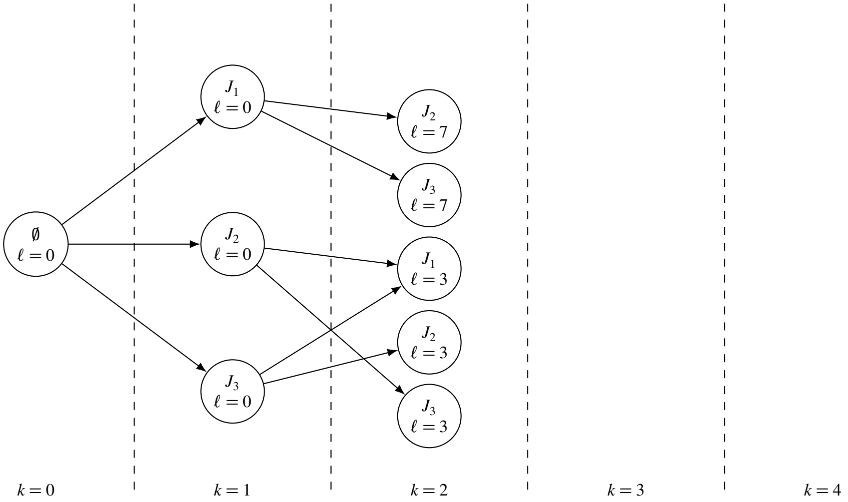


Extended network G_2 - Example of reduction

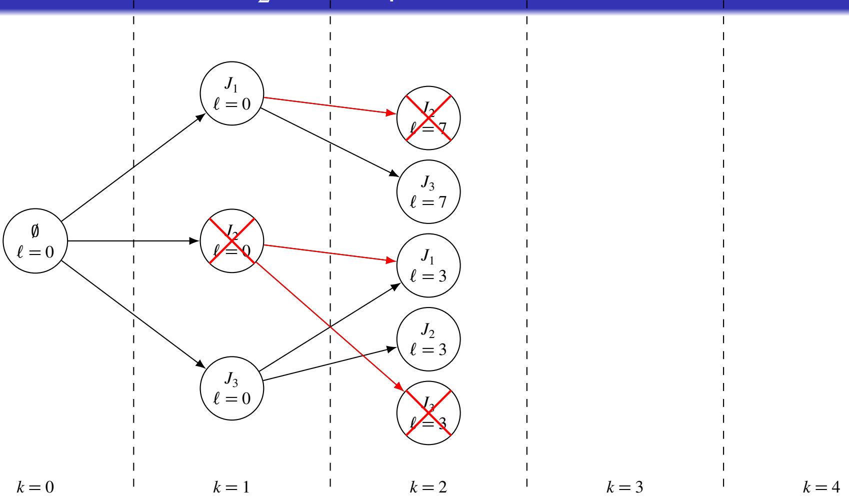


Jobs cannot be processed twice consecutively

Extended network G_2 - Example of reduction



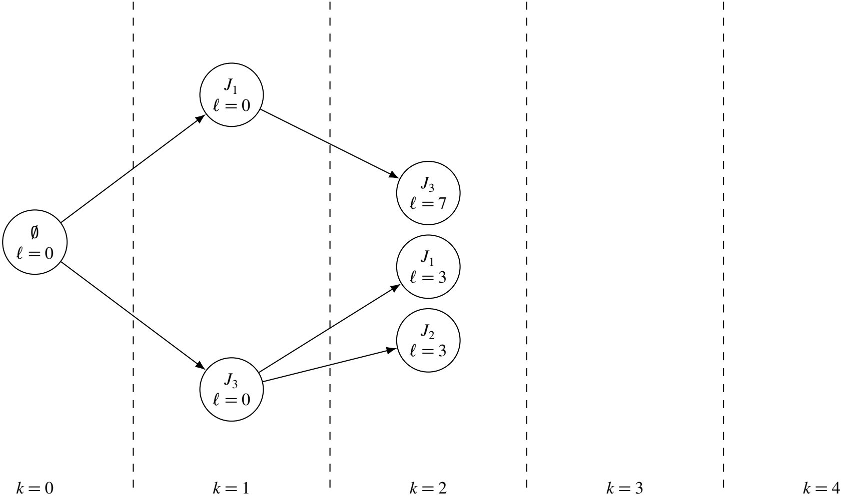
Extended network G_2 - Example of reduction



If $p_i^1 + s_j^2 \leq p_j^1 + s_i^2$, $p_i^2 + s_i^2 \leq p_j^2 + s_j^2$, and $p_j^2 \leq p_i^2$, then $i \rightarrow j$

$\Rightarrow J_3 \rightarrow J_2$ [Allahverdi, 2000]

Extended network G_2 - Example of reduction



Extended network G_2 - Example of reduction

Given a position k, a lag ℓ and a sub-sequence σ:

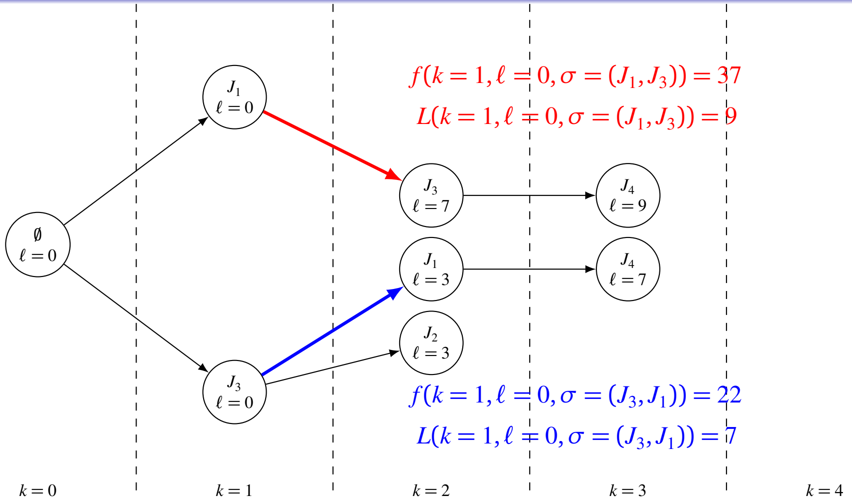
- $f(k,\ell,\sigma)$: cost of scheduling σ at (k,ℓ)
- $L(k,\ell,\sigma)$: lag of the last job of σ scheduled at (k,ℓ)

Dominance

Sub-sequence σ is dominated at (k,ℓ) by sub-sequence σ' if:

- The set of jobs in σ and σ' is the same
- $f(k,\ell,\sigma) > f(k,\ell,\sigma')$
  The partial schedule up to the end of σ' will be less costly
- $L(k,\ell,\sigma) \geq L(k,\ell,\sigma')$
  The partial schedule after σ' will not be more costly

Extended network G_2 - Example of reduction



Example: $|\sigma|=2$ allows us to remove some arcs

Lagrangian cost variable fixing

Additional input data

An upper bound UB of the optimum is known

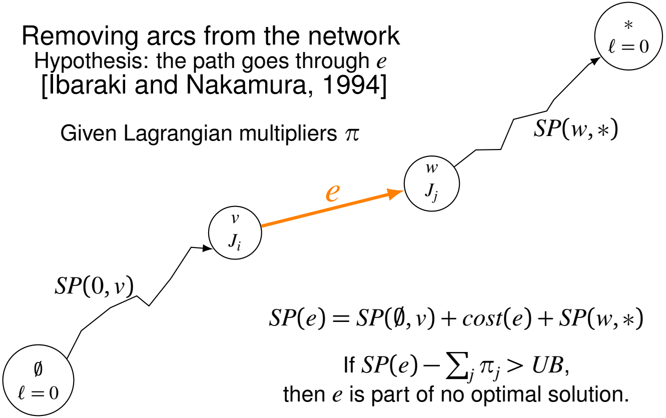
Principle

- Assume that one dominant optimal solution satisfies hypothesis h
  The optimal path goes through a given arc
- Compute a (Lagrangian) lower bound LB_h under h
- If $LB_h > UB$, then h is not satisfied in any optimal dominant solution
  The arc can be removed from the graph

Lagrangian cost variable fixing (1)

Removing arcs from the network
Hypothesis: the path goes through e
[Ibaraki and Nakamura, 1994]

Given Lagrangian multipliers π



$$SP(e) = SP(\emptyset, v) + cost(e) + SP(w, *)$$

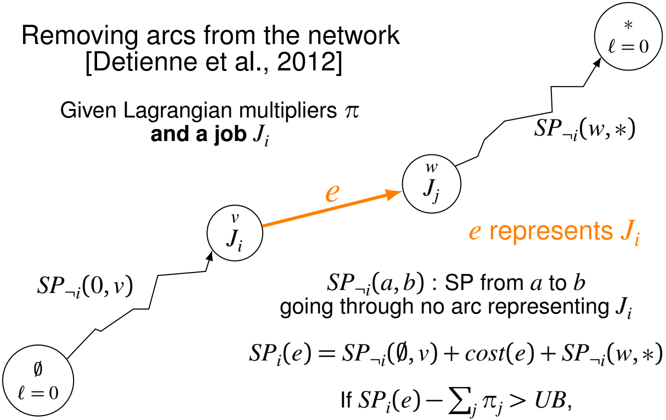
If $SP(e) - \sum_j \pi_j > UB$,
then e is part of no optimal solution.

Computing $SP(e)$ for all $e \in E$ is done in $O(|E|)$-time

Lagrangian cost variable fixing (2)

Removing arcs from the network
[Detienne et al., 2012]

Given Lagrangian multipliers π
and a job J_i



$SP_{\neg i}(a,b)$: SP from a to b
going through no arc representing J_i

$$SP_i(e) = SP_{\neg i}(\emptyset, v) + cost(e) + SP_{\neg i}(w,*)$$

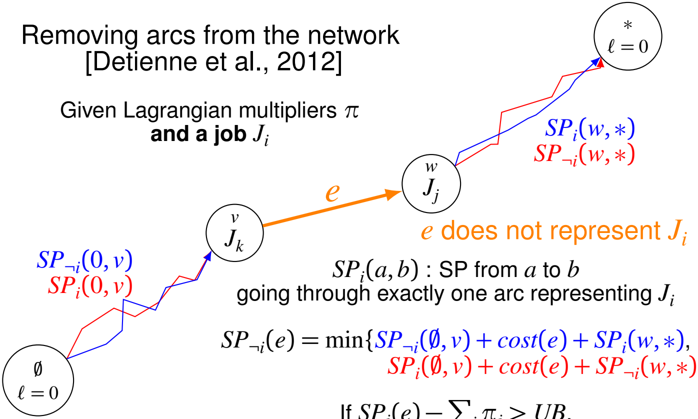
If $SP_i(e) - \sum_j \pi_j > UB$,
then e is part of no optimal solution.

Computing $SP_i(e)$ for all $e \in E$ and $i \in I$ is done in $O(n|E|)$-time

Lagrangian cost variable fixing (2)

Removing arcs from the network
[Detienne et al., 2012]

Given Lagrangian multipliers π
and a job J_i



$SP_i(a,b)$: SP from a to b
going through exactly one arc representing J_i

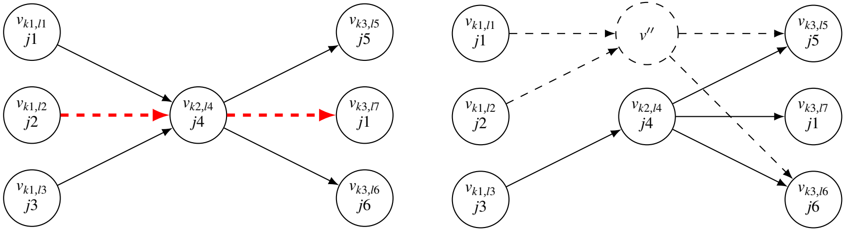
$$SP_{\neg i}(e) = \min\{SP_{\neg i}(\emptyset, v) + cost(e) + SP_i(w,*), \\ SP_i(\emptyset, v) + cost(e) + SP_{\neg i}(w,*)\}$$

If $SP_i(e) - \sum_j \pi_j > UB$,
then e is part of no optimal solution.

Computing $SP_i(e)$ for all $e \in E$ and $i \in I$ is done in $O(n|E|)$-time

Lower bound improvement using local dominance

Inspection of optimal solutions of Lagrangian subproblems: dominated or infeasible 3 job-paths are removed from the graph



Preprocessing

Initial upper bound

A good feasible solution is obtained by a local search procedure *Dynasearch* [Tanaka, 2010]

Pre-computation of lower bounds

- Construction of network G_1
- Lagrangian cost variable fixing (subgradient procedure)
- Construction of the extended network G_2 from G_1
- Lagrangian cost variable fixing (subgradient procedure)
- For the best Lagrangian multipliers, $SP_i(v,*)$ and $SP_{\neg i}(v,*)$ are stored for each $i \in I$ and $v \in V$

Branching scheme

Solution space explored

- Feasible sequences of jobs $\equiv$ Feasible constrained paths in G_2
- Depth-First Search, starting from start node $\emptyset$

Branching

Current sequence σ ($\equiv$ path) is extended with job J_i iff:

- There is a corresponding arc in G_2
- All predecessors of J_i are in σ and J_i is not in σ
- Predictive memorization?: The sequence of the last 5 jobs obtained would not be dominated by one of its permutations
- Static node memorization: The sequence is not dominated by a previously explored sequence [Baptiste et al., 2004], [T'Kindt et al., 2004], [Kao et al., 2008]

Lower bound for $\sigma \equiv$ path ending at v in G_2

Lower bound coming from jobs not sequenced yet

$$LB_1 = cost(\sigma) + \max_{i \notin \sigma} SP_i(v,*) - \sum_{i \notin \sigma} \pi_i$$

Lower bound coming from sequenced jobs

$$LB_2 = cost(\sigma) + \max_{i \in \sigma} SP_{\neg i}(v,*) - \sum_{i \notin \sigma} \pi_i$$

Computing $\max\{LB_1, LB_2\}$ is done in $\mathcal{O}(n)$-time.

Tentative upper bound

Weakness of the approach

If the initial upper bound is too large, variable fixing is not efficient.

Overall procedure

1. Build and filter G_1 using the initial upper bound (dynasearch)
2. If G_1 is *sufficiently small*, build and filter G_2 from G_1, run the Branch-and-Bound, STOP
3. Build and filter G_2 from G_1 using a tentative upper bound
4. Run the Branch-and-Bound
5. If a feasible solution is found, it is optimal, STOP
6. Otherwise, increase the tentative upper bound and go to 3

1 Introduction

2 Lower bounds

3 Branch-and-bound

4 Numerical results

Setup

Coded in C++ (MS VS 2012)

MS Windows 8 laptop with 16GB RAM and Intel Core i7 @2.7GHz

Instances of $F_2||\sum C_i$

- Randomly generated [Akkan and Karabati, 2004], [Haouari and Kharbeche 2013]
- Up to 140 jobs, p_i^1 and p_i^2 are drawn from $\mathcal{U}[1,100]$

Instances of $F_2||\sum C_i$

- Subset of the testbed of [Gharbi et al., 2013]
- Up to 100 jobs, p_i^1, p_i^2 and s_i^2 are drawn from $\mathcal{U}[1,100]$

Size of the networks - With initial upper bound

Number of nodes in G_2 (in thousands)

Duration	$F2\|\|\sum C_i$ n=40	n=60	n=80	n=100	$F2\|ST_{si}\|\sum C_i$ n=60	n=70	n=80	n=100
$[1-10]$	2.3	7.8	17.0	35.8				
$[1-100]$	26.5	92.7	212.4	391.3	246.7	426.9	608.4	1 234.1

Number of arcs in G_2 (in thousands)

Duration	$F2\|\|\sum C_i$ n=40	n=60	n=80	n=100	$F2\|ST_{si}\|\sum C_i$ n=60	n=70	n=80	n=100
$[1-10]$	12.9	68.2	217.6	642.7				
$[1-100]$	164.2	937.0	2925.4	6431.4	3818.3	8224.6	13 550.5	35 554.8

Number of nodes in G_2 after filtering (in thousands)

Duration	$F2\|\|\sum C_i$ n=40	n=60	n=80	n=100	$F2\|ST_{si}\|\sum C_i$ n=60	n=70	n=80	n=100
$[1-10]$	0.4	2.2	6.6	13.0				
$[1-100]$	5.2	35.5	92.5	166.2	163.7	284.8	396.8	766.3

Number of arcs in G_2 after filtering (in thousands)

Duration	$F2\|\|\sum C_i$ n=40	n=60	n=80	n=100	$F2\|ST_{si}\|\sum C_i$ n=60	n=70	n=80	n=100
$[1-10]$	0.8	7.6	38.6	99.2				
$[1-100]$	16.4	170.7	639.0	1465.4	1866.5	4236.0	6931.7	18 544.7

Size of the networks - With tentative upper bound

For problem $F2|ST_{SI}|\sum C_i$, using the best feasible tentative upper bound

Number of nodes in G_2 after filtering (in thousands)							
Initial upper bound				Best feasible tentative upper bound			
n=60	n=70	n=80	n=100	n=60	n=70	n=80	n=100
163.7	284.8	396.8	766.3	63.1	88.4	135.1	237.1

Number of arcs in G_2 after filtering (in thousands)							
Initial upper bound				Best feasible tentative upper bound			
n=60	n=70	n=80	n=100	n=60	n=70	n=80	n=100
1 866.5	4 236.0	6 931.7	18 544.7	344.1	544.5	1013.3	2 237.8

No setup times - $F_2||\sum C_i$

Results for 100—job instances (40 instances)

- Avg. time: 216 s., Max. time: 602 s.
- Tentative upper bound is useless
  Root gap $\approx 7 \times 10^{-4}$
- Variable fixing reduces the number of arcs by a factor 5
  Avg.: $\approx 166K$ *nodes,* $\approx 1.4M$ *arcs, Max.:* $239K$ *nodes,* $2.9M$ *arcs*

Results for 140—job instances (40 instances)

- Avg. time: 752 s., Max. time: 3006 s.
- Tentative upper bound is useless
- Small processing times: 18/20 solved in 1000 s.
- Large processing times: 12/20 solved in 1000 s.

Sequence-independent setup times - $F_2|ST_{SI}|\sum C_i$

Results for $100-$ job instances (200 instances)

- Avg. time: 935 s., Max. time: 6443 s.
- Tentative upper bound is critical
  Reduces the number of arcs from $18.5M$ to $2.2M$ at the root node
- Lagrangian Variable fixing + Tentative upper bound reduce the number of arcs by a factor 17
  Avg.: $\approx 237K$ nodes, $\approx 2.2M$ arcs, Max.: $440K$ nodes, $4.9M$ arcs
- Solves 145/200 instances in 1000 s.

Impact of static node memorization

When the rule is disabled

Problem $F_2||\sum C_i$, $100-$job instances

- Large processing times: 38/40 solved in 1000 s.
- Max. solving time: 602s. $\rightarrow$ 7700 s.
- Average computing time multiplied by a factor 4
- Average number of B&B nodes increased by a factor 45: 3.9M $\rightarrow$ 179M
- Maximum number of B&B nodes: 2.7 billions

Conclusion

Contributions

- New lower bound for $F2||\sum C_i$ and $F2|ST_{SI}|\sum C_i$
- Efficient management of the size of the extended network
- Dominance rules are embedded in the structure of the network
- The lower bound is used with success in an exact solving approach
- All 100-job instances of our test bed are solved in less than two hours
  98% are solved in less than one hour

Future directions

- Use Successive Sublimation Dynamic Programming instead of Branch-and-Bound
- Adapt for other *min-sum* objective functions?
- Adapt for more than two machines permutation flowshop?

Thank you for your attention

Network flow formulation [Akkan et Karabati, 2004]: G_1

- V_1, A_1 : sets of nodes and arcs
- $x_{v,w,j}$: amount of flow on the arc representing j between nodes v and w

$$\min \sum_{(v,w,j)\in A_1} c_{v,w,j} x_{v,w,j}$$

$$s.t. \sum_{(v,w,j)\in A_1} x_{v,w,j} = \sum_{(w,v,j)\in A_1} x_{w,v,j} \qquad \forall v \in V_1 - \{(0,0),(n+1,0)\}$$

$$\sum_{(v,w,j)\in A_1} x_{v,w,j} = 1 \qquad \forall j = 1,\ldots,n$$

$$\sum_{(0,w,j)\in A_1} x_{0,w,j} = 1$$

$$x_{v,w,j} \in \{0,1\} \qquad \forall (v,w,j) \in E_1$$

Lower bound by Lagrangian relaxation

- V_1, A_1 : sets of nodes and arcs
- $x_{v,w,j}$: amount of flow on the arc representing j between nodes v and w

$$L(\pi) = \min \sum_{(v,w,j)\in A_1} c_{v,w,j} x_{v,w,j} + \sum_{j=1}^{n} \pi_j \left(\sum_{(v,w):(v,w,j)\in A_1} x_{v,w,j} - 1 \right)$$

$$s.t. \sum_{(v,w,j)\in A_1} x_{v,w,j} = \sum_{(w,v,j)\in A_1} x_{w,v,j} \qquad \forall v \in V_1 - \{(0,0),(n+1,0)\}$$

$$\sout{\sum_{(v,w,j)\in A_1} x_{v,w,j} = 1 \qquad \forall j = 1,\ldots,n}$$

$$\sum_{(0,w,j)\in A_1} x_{0,w,j} = 1$$

$$x_{v,w,j} \in \{0,1\} \qquad \forall (v,w,j) \in A_1$$

Lower bound by Lagrangian relaxation

- V_1, A_1 : sets of nodes and arcs
- $x_{v,w,j}$: amount of flow on the arc representing j between nodes v and w

$$L(\pi) = \min \sum_{(v,w,j)\in A_1} \left(c_{v,w,j} + \pi_j\right) x_{v,w,j} - \sum_{j=1}^{n} \pi_j$$

$$s.t. \sum_{(v,w,j)\in A_1} x_{v,w,j} = \sum_{(w,v,j)\in A_1} x_{w,v,j} \qquad \forall v \in V_1 - \{(0,0),(n+1,0)\}$$

$$\sout{\sum_{(v,w,j)\in A_1} x_{v,w,j} = 1 \qquad \forall j = 1,\ldots,n}$$

$$\sum_{(0,w,j)\in A_1} x_{0,w,j} = 1$$

$$x_{v,w,j} \in \{0,1\} \qquad \forall (v,w,j) \in A_1$$

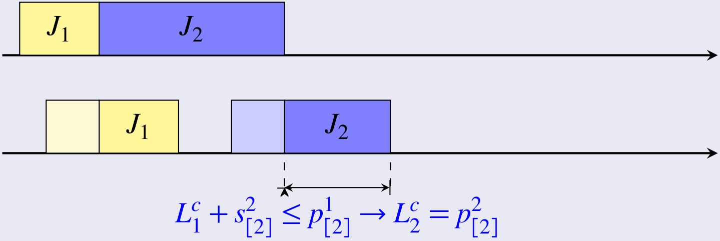
Subproblem: shortest path in the network

Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

- $C^m_{[k]}$: completion time of the job in position k on machine m
- L^c_k: time **lag** elapsed between the completion of the job in position k on machines 1 and 2

$$L^c_k = C^2_{[k]} - C^1_{[k]} = \max\left\{0, L^c_{k-1} + s^2_{[k]} - p^1_{[k]}\right\} + p^2_{[k]}$$



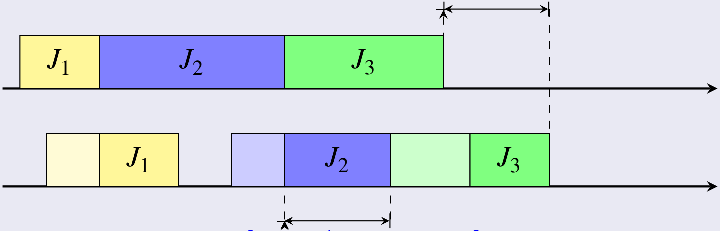
Lag-based models [Akkan and Karabati], [Gharbi et al.]

Lag variables

- $C^m_{[k]}$: completion time of the job in position k on machine m
- L^c_k: time **lag** elapsed between the completion of the job in position k on machines 1 and 2

$$L^c_k = C^2_{[k]} - C^1_{[k]} = \max\left\{0, L^c_{k-1} + s^2_{[k]} - p^1_{[k]}\right\} + p^2_{[k]}$$

$$L^c_2 + s^2_{[3]} > p^1_{[3]} \rightarrow L^c_3 = L^c_2 + s^2_{[3]} - p^1_{[3]} + p^2_{[3]}$$



$$L^c_1 + s^2_{[2]} \leq p^1_{[2]} \rightarrow L^c_2 = p^2_{[2]}$$

Lag-based models

Formulating the objective function

Minimizing the sum of completion times:

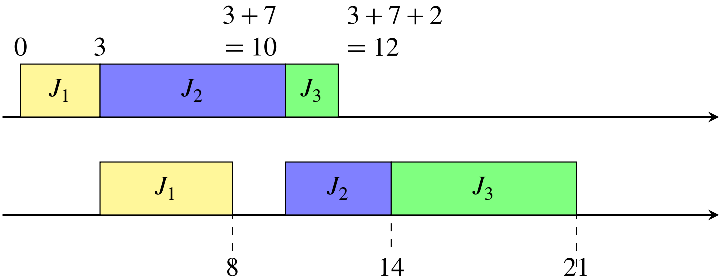
$$\sum_{k=1}^{n} C^2_{[k]} = \sum_{k=1}^{n} \left(C^1_{[k]} + L^c_k \right)$$

$$= \sum_{k=1}^{n} \left(\sum_{r=1}^{k} p^1_{[r]} + L^c_k \right)$$

$$= \sum_{k=1}^{n} \left((n-k+1) p^1_{[k]} + L^c_k \right)$$

Example

$$p_1 = (3,5); \quad p_2 = (7,4); \quad p_3 = (2,7)$$



Cost of the schedule: $(3 \times 3 + 5) + (7 \times 2 + 4) + (2 \times 1 + 9) = 43$

Lower bound for $\sigma \equiv$ path ending at v in G_2

Lower bound coming from jobs not sequenced yet

$$LB_1 = cost(\sigma) + \max_{i \notin \sigma} SP_i(v,*) - \sum_{i \notin \sigma} \pi_i$$

Lower bound coming from sequenced jobs

$$LB_2 = cost(\sigma) + \max_{i \in \sigma} SP_{\neg i}(v,*) - \sum_{i \notin \sigma} \pi_i$$

Computing $\max\{LB_1, LB_2\}$ is done in $\mathcal{O}(n)$-time.