A Local Search Based Metaheuristic for a Class of Earliness and Tardiness Scheduling Problems

Arthur Kramer[1,2]
Anand Subramanian[2]

[1]Università degli Studi di Modena e Reggio Emilia - Italy
[2] Universidade Federal da Paraíba - Brazil

March 31, 2016

Conselho Nacional de Desenvolvimento Científico e Tecnológico

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Outline

1. Introduction
   - Problem
   - Motivation
   - Literature
2. Method
   - Iterated Local Search
   - Neighborhoods
   - Move Evaluation
3. Computational Experiments
4. Conclusions and Future Work

Outline

1 Introduction
- Problem
- Motivation
- Literature

2 Method
- Iterated Local Search
- Neighborhoods
- Move Evaluation

3 Computational Experiments

4 Conclusions and Future Work

Outline

1. Introduction
   - Problem
   - Motivation
   - Literature

2. Method
   - Iterated Local Search
   - Neighborhoods
   - Move Evaluation

3. Computational Experiments

4. Conclusions and Future Work

Outline

1 Introduction
  - Problem
  - Motivation
  - Literature

2 Method
  - Iterated Local Search
  - Neighborhoods
  - Move Evaluation

3 Computational Experiments

4 Conclusions and Future Work

Outline

1 Introduction
- Problem
- Motivation
- Literature

2 Method
- Iterated Local Search
- Neighborhoods
- Move Evaluation

3 Computational Experiments

4 Conclusions and Future Work

Problem

- Class of Just-In-Time scheduling problems
- Penalties if a job is completed before or after its due date
- Usually $\mathcal{NP}-$Hard
- Additional features
  - Nonzero release dates
  - Multiple machines
  - Sequence-dependent setup times

Problem definition

- $J = \{1, \ldots, n\}$ is a set of jobs
- $M = \{1, \ldots, m\}$ is a set of **unrelated** parallel machines
- For each job $j \in J$:
  - p_j^k: processing time of j in machine $k \in M$
  - d_j: due date of j
  - r_j: release date of j
  - w_j': earliness penalty weight
  - w_j: tardiness penalty weight
  - s_{ij}^k: setup time required before starting to process j immediately after i in machine k

Problem definition

- Schedule all the jobs over the machines
- Each machine can process only one job at time
- The jobs are ready to be processed after their release date
- Setup times are required when changing the jobs being processed
- Preemption is not allowed
- Idle time can be inserted between two consecutive jobs
- Objective: minimize the sum of the weighted earliness and tardiness

Problem definition

Objective Function

$$\text{Minimize} \sum_{j \in J} w'_j E_j + w_j T_j$$

- $E_j = \max\{d_j - C_j, 0\}$: earliness
- $T_j = \max\{C_j - d_j, 0\}$: tardiness

Classification $(\alpha|\beta|\gamma)$

$$R|r_j, s_{ij}^k| \sum w'_j E_j + w_j T_j$$

Considered problems

- Particular cases of the $R|r_j, s_{ij}^k|\sum w_j' E_j + w_j T_j$

Considered problems

Single Machine	Identical Parallel Machines	Uniform Parallel Machines	Unrelated Parallel Machines
$1\|\|\sum w_j T_j$	$P\|\|\sum w_j T_j$	$Q\|\|\sum w_j T_j$	$R\|\|\sum w_j T_j$
$1\|r_j\|\sum w_j T_j$	$P\|r_j\|\sum w_j T_j$	$Q\|r_j\|\sum w_j T_j$	$R\|r_j\|\sum w_j T_j$
$1\|s_{ij}\|\sum w_j T_j$	$P\|s_{ij}\|\sum w_j T_j$	$Q\|s_{ij}^k\|\sum w_j T_j$	$R\|s_{ij}^k\|\sum w_j T_j$
$1\|r_j, s_{ij}\|\sum w_j T_j$	$P\|r_j, s_{ij}\|\sum w_j T_j$	$Q\|r_j, s_{ij}^k\|\sum w_j T_j$	$R\|s_{ij}^k\|\sum w_j T_j$
$1\|\|\sum w_j' E_j + w_j T_j$	$P\|\|\sum w_j' E_j + w_j T_j$	$Q\|\|\sum w_j' E_j + w_j T_j$	$R\|\|\sum w_j' E_j + w_j T_j$
$1\|r_j\|\sum w_j' E_j + w_j T_j$	$P\|r_j\|\sum w_j' E_j + w_j T_j$	$Q\|r_j\|\sum w_j' E_j + w_j T_j$	$R\|r_j\|\sum w_j' E_j + w_j T_j$
$1\|s_{ij}\|\sum w_j' E_j + w_j T_j$	$P\|s_{ij}\|\sum w_j' E_j + w_j T_j$	$Q\|s_{ij}^k\|\sum w_j' E_j + w_j T_j$	$R\|s_{ij}^k\|\sum w_j' E_j + w_j T_j$
$1\|r_j, s_{ij}\|\sum w_j' E_j + w_j T_j$	$P\|r_j, s_{ij}\|\sum w_j' E_j + w_j T_j$	$Q\|r_j, s_{ij}^k\|\sum w_j' E_j + w_j T_j$	$R\|r_j, s_{ij}^k\|\sum w_j' E_j + w_j T_j$
$1\|r_j\|\sum w_j C_j$	$P\|\|\sum w_j C_j$	$Q\|\|\sum w_j C_j$	$R\|\|\sum w_j C_j$
$1\|s_{ij}\|\sum w_j C_j$	$P\|r_j\|\sum w_j C_j$	$Q\|r_j\|\sum w_j C_j$	$R\|r_j\|\sum w_j C_j$
$1\|r_j, s_{ij}\|\sum w_j C_j$	$P\|s_{ij}\|\sum w_j C_j$	$Q\|s_{ij}^k\|\sum w_j C_j$	$R\|s_{ij}^k\|\sum w_j C_j$
	$P\|r_j, s_{ij}\|\sum w_j C_j$	$Q\|r_j, s_{ij}^k\|\sum w_j C_j$	$R\|r_j, s_{ij}^k\|\sum w_j C_j$

Motivation

- $\mathcal{NP}$-Hard problems
- Real life problems
- Even harder when idle time is allowed (timing problem)

Literature Review

- More than 120 relevant papers in the last 25 years
- Exact approaches (B&B, DP, LR...)
- (Meta)-heuristics Methods (GA, TS, ILS...)
- Efficient local search methods

Outline

1 Introduction
- Problem
- Motivation
- Literature

2 Method
- Iterated Local Search
- Neighborhoods
- Move Evaluation

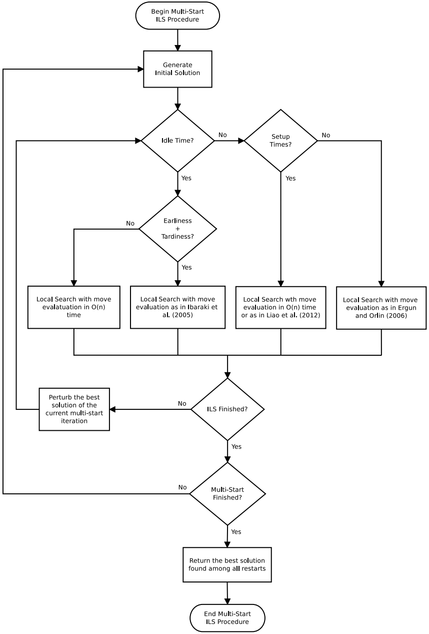
3 Computational Experiments

4 Conclusions and Future Work

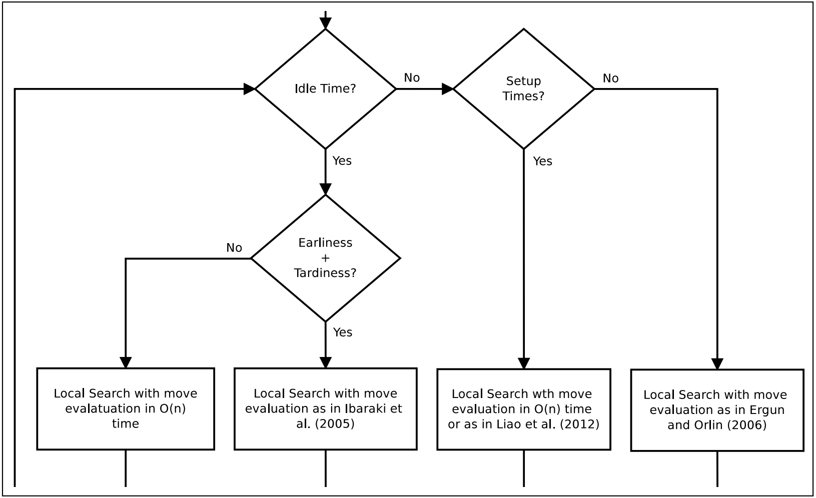
Iterated Local Search

- Multi-start local search based meta-heuristic
- High quality results on many kind of problems (e.g. routing)
- Devised to solve a large class of ET scheduling problems
- The method considers specif features of the different problems
- Components:
  - Initial solutions
  - Local search
  - Perturbation

Iterated Local Search



Iterated Local Search



Initial Solution

- Two simple procedures:
  - Random iterative insertions
  - EDD based on GRASP constructive phase
- The initial solutions did not significantly affect, on average, neither the final solution quality nor the speed of convergence.

Local search

- Performed by the Randomized Variable Neighborhood Descent (RVND) procedure
- RVND consists of:
  - List of neighborhoods
  - Selected at random
  - If a neighborhood improves the best solution, all the neighborhoods come back to the list
  - Repeat until the list becomes empty

Perturbation mechanisms

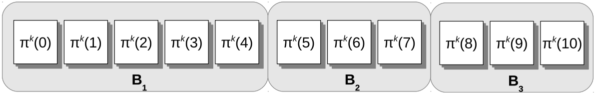
- **(l, l')-block swap intra-machine**: one (l, l')-block swap intra-machine move is performed at random with l and $l' \in \{2, \ldots, \lfloor n/4 \rfloor\}$.
- **Multiple $(1, 1)$-insertion inter-machine**: a job j from a machine k is moved to a machine k', while a job j' from k' is moved to k. Such procedure is repeated one, two or three consecutive times.
- **Multiple (l, l')-block insertion inter-machine**: this perturbation generalizes the previous one but in this case blocks of jobs of sizes l and l', respectively, are involved in the move with $l \in \{1, 2\}$ and $l' \in \{2, 3\}$.

Neighborhoods

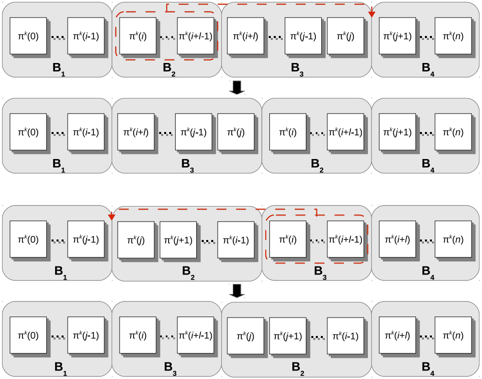
- Intra machine
  - l-block insertion
  - (l, l')-block swap
- Inter machines
  - l-block insertion
  - (l, l')-block swap

Neighborhoods

- Solution representation
  - Let $\pi_k = (\pi_k(0), \pi_k(1), \ldots, \pi_k(n))$ be a sequence of $n+1$ jobs associated to a machine $k \in M$
  - A block B consists of subsequence of consecutive jobs

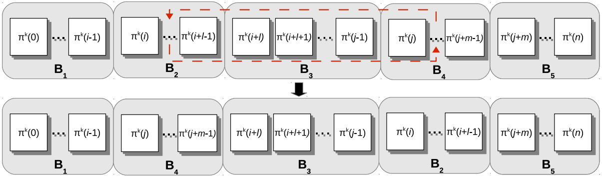


- ***l*-block insertion intra machine**
  - Consists of moving a block of size l forward or backward in the sequence



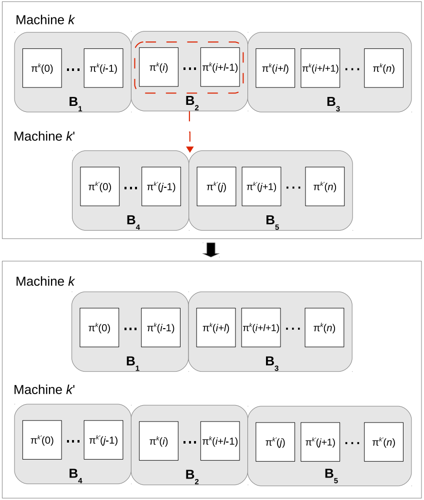
Neighborhoods

- **(l, l')-block swap intra machine**
  - Consists of interchanging a block of size l with another one with size l' of the same sequence

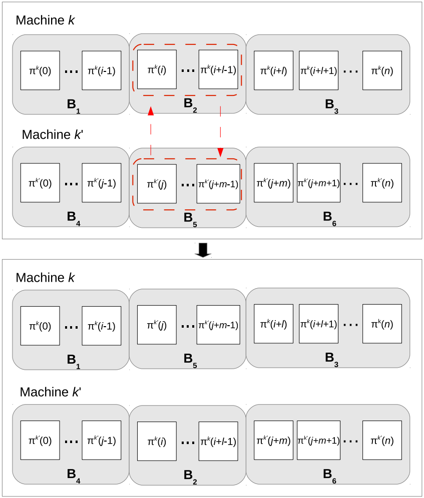


Neighborhoods

- **l-block insertion inter machines**
  - Consists of removing a block of size l from a machine k and inserting it in machine k'



- **(l, l')-block swap inter machines**
  - Consists of interchanging a block of size l of machine k with a block of size l' of machine k'



Neighborhoods

- The size(s) of the block(s) in each type of neighborhood is an input parameter
- Size of each neighborhood: $\mathcal{O}(n^2)$
- For problems without idle time, a straightforward move evaluation can be performed in $\mathcal{O}(n)$
- Overall complexity of $\mathcal{O}(n^3)$
- For problems with idle time the timing problem should be solved

Move Evaluation

- **Without IT**: straightforward move evaluation → $\mathcal{O}(n)$. Overall complexity: $\mathcal{O}(n^3)$
- **Without both IT and ST**: amortized $\mathcal{O}(1)$ → following the ideas of Ergun and Orlin (2006). Overall complexity: $\mathcal{O}(n^2)$
- **Without IT and with ST**: amortized $\mathcal{O}(1)$ → following the ideas of Liao *et al.* (2012). Overall complexity: $\mathcal{O}(n^2 \log n)$
- **With IT**: Timing problem should be solved for each sequence: $\mathcal{O}(n \log n)$. Overall complexity: $\mathcal{O}(n^3 \log n)$. (Ibaraki *et al.*, 2008) reduce the overall complexity

Move Evaluation

- **Without IT**: straightforward move evaluation $\rightarrow \mathcal{O}(n)$. Overall complexity: $\mathcal{O}(n^3)$
- **Without both IT and ST**: amortized $\mathcal{O}(1) \rightarrow$ following the ideas of Ergun and Orlin (2006). Overall complexity: $\mathcal{O}(n^2)$
- **Without IT and with ST**: amortized $\mathcal{O}(1) \rightarrow$ following the ideas of Liao *et al.* (2012). Overall complexity: $\mathcal{O}(n^2 \log n)$
- **With IT**: Timing problem should be solved for each sequence: $\mathcal{O}(n \log n)$. Overall complexity: $\mathcal{O}(n^3 \log n)$. (Ibaraki *et al.*, 2008) reduce the overall complexity

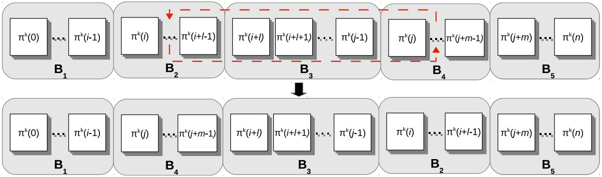
Without idle times and without setup times

- Ergun and Orlin (2006) method
  - Ergun and Orlin (2006) proposed a move evaluation in $\mathcal{O}(1)$ for the $1||w_j T_j$
    - Insertion intra machine (1-block insertion intra)
    - Swap intra ((1-1)-block swap intra)
    - Block reverse – Twist
  - We extend to the unrelated parallel machines cases without both idle times and setup times
  - Intra and inter neighborhoods
  - This method uses auxiliary functions and data structures

W_j^k: cost of the block $B = (\pi_k(0), \ldots, \pi_k(j))$ in a sequence π_k

$$W_j^k = \begin{cases} \sum_{i=1}^{j} w'_{\pi_k(i)}(d_{\pi_k(i)} - C_{\pi^k(i)}), & \text{if} \quad C_{\pi^k(i)} \leq d_{\pi_k(i)} \\ \sum_{i=1}^{j} w_{\pi_k(i)}(C_{\pi^k(i)} - d_{\pi_k(i)}), & \text{if} \quad C_{\pi^k(i)} \geq d_{\pi_k(i)} \end{cases}$$

- Example - (l, l')-block swap intra



- New cost: $C(B_1) + C(B'_4) + C(B'_3) + C(B'_2) + C(B'_5)$

Without idle times and without setup times

Penalty function of a job

- $\rho_j^k(t)$: penalty of starting to process j in machine k at time t

$$\rho_j^k(t) = \begin{cases} M(r_j - t) + w_j'(d_j - p_j^k - t), & t \in [0, r_j] \\ w_j'(d_j - p_j^k - t), & t \in [r_j, d_j - p_j^k] \\ w_j(d_j - p_j^k + t), & t \in [d_j - p_j^k, +\infty) \end{cases}$$

- M: big number to allow the r_j violation

Without idle times and without setup times

- Characteristics of $\rho_j^k(t)$:
  - Non-negative, convex and piecewise linear function
  - At most three segments defined by: r_j, d_j and p_j^k
  - Without $r_j \rightarrow$ two segments

Without idle times and without setup times

Penalty function of a block of tasks

- $g_j^k(t)$: cost of a block $(\pi_k(j), \ldots, \pi_k(n_k))$ sequenced in machine k in this order where job $\pi_k(j)$ starts to be processed at time t

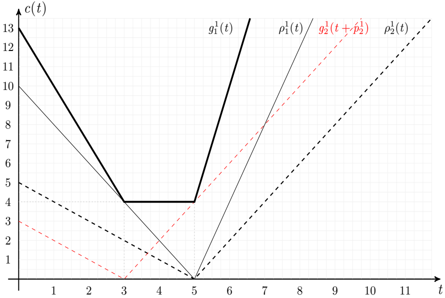
$$g_j^k(t) = \begin{cases} +\infty, & j = n_k + 1, \quad t \in (-\infty, 0) \\ 0, & j = n_k + 1, \quad t \in [0, +\infty) \\ g_{j+1}^k(t + p_j^k + s_{j,\,j+1}^k) + \rho_j^k(t), & n_k \geq j > 0, \quad -\infty < t < +\infty \end{cases}$$

Without idle times and without setup times

- Example $\rho_k^k(t)$ e $g_j^k(t)$

- Assuming $r_j = 0, \forall j \in J$
- Given $\pi_1 = (0, 1, 2, 0)$

j	r_j	p_j	d_j	w'_j	w_j
1	0	3	8	2	4
2	0	2	7	1	2
3	0	1	4	2	4
4	0	3	3	1	3
5	0	1	13	1	1



$$\rho_1^1(t) = \begin{cases} 10 - 2t, & t \in [0, 5] \\ 4(t-5), & t \in [5, +\infty) \end{cases}$$

$$g_2^1(t+p_2^1) = \begin{cases} 3 - 1t, & t \in [0, 3] \\ 2(t-3), & t \in [3, +\infty) \end{cases}$$

$$\rho_2^1(t) = \begin{cases} 5 - 1t, & t \in [0, 5] \\ 2(t-5), & t \in [5, +\infty) \end{cases}$$

$$g_1^1(t) = \begin{cases} 13 - 3t, & t \in [0, 3] \\ 4, & t \in [3, 5] \\ 4 + 6(t-5), & t \in [5, +\infty) \end{cases}$$

Without idle times and without setup times

- $\rho_j^k(t)$ e $g_j^k(t)$ can be stored in memory by means of linked lists
- Each element of the list is associated with one of the segments of the piecewise linear function
  - $\mathtt{b}_1$: beginning of the segment's interval;
  - $\mathtt{b}_2$: end of the segment's interval;
  - $\mathtt{c}$: value of the function at time b_1;
  - α: slope of the segment.

$$\boxed{[b_1, b_2]: c - \alpha(t - b_1)}$$

- Cost of a block $(\pi_k(j), \pi_k(j+1), \ldots, \pi_k(n_k))$ starting at time t:
  - Walk through the breakpoints of the corresponding function
  - Reach the interval $b_1 \leq t \leq b_2$
  - Compute the cost. Given by $c + \alpha \times (t - b_1)$

Without idle times and without setup times

- Another important data structure
  - `ProcessingList`
    - Stores the processing time of a block of size l
    - In our case is defined for all blocks of size l from a given sequence π_k
    - Must be sorted in descending order of the processing time of the block

Algorithm 1 CreateProcessingList

Procedure CreateProcessingList(k, k', l)
$p_b \leftarrow 0$
for $i' = 1, \dots, l$ **do**
    $p_b \leftarrow p_b + p^{k'}_{\pi_k(i')}$
for $i \leftarrow 1, \dots, n_k - l + 1$ **do**
    `ProcessingList.pos[i]` $\leftarrow i$
    `ProcessingList.p[i]` $\leftarrow p_b$
    $p_b \leftarrow p_b - p^{k'}_{\pi_k(i)} + p^{k'}_{\pi_k(i+l)}$
`sort(ProcessingList,p)` **return** `ProcessingList`
End CreateProcessingList.

Auxiliary functions and data structures

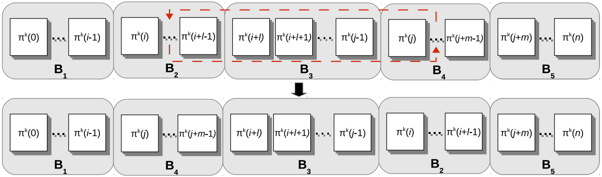
- W_j^k: represents the cost of a block $B = (\pi_k(0), \ldots, \pi_k(j))$
- $\rho_j^k(t)$: penalty function for a job j starting at time t on machine k
- $g_j^k(t)$: the cost of a block $B = (\pi_k(j), \ldots, \pi_k(n_k))$
- `ProcessingList`: ordered list containing the processing times of the blocks of size l

Move evaluation

- For a given sequence π_k, $g_j^k(t)$ can have at most n_k breakpoints
- $g_j^k(t)$ are piecewise linear functions, then the complexity of obtaining the cost of a block $(\pi_k(j), \ldots, \pi_k))$ starting at time t is $\mathcal{O}(n_k)$
- The complexity of evaluating a move **would be** $\mathcal{O}(n_k)$.
- However, it is **possible** to evaluate a move of the neighborhoods in $\mathcal{O}(1)$ time
- The costs that depend on $g_j^k(t)$ are precomputed following a given order (`ProcessingList`)

Move evaluation

- Example – (l, l')-block swap intra
  - Preprocessing: $\mathcal{O}(n^2)$
  - Neighborhood complexity: $\mathcal{O}(\max(l, l')n^2)$
  - Exchanged blocks are evaluated in $\mathcal{O}(l)$ and $\mathcal{O}(l')$ since they are small, but can be evaluated in $\mathcal{O}(1)$ by using $g_j^k(t)$ functions



$$c(\pi'^k_{ij}) = W^k_{i-1} + c(B'_4) + [g^k_{i+l}(t_1) - g^k_j(t_2)] + c(B'_2) + [W^k_{n^k} - W^k_{j+l'}]$$

$$t_1 = C^k_{i-1} + \sum_{a=j}^{j+l'-1} p^k_{\pi_k(a)} \qquad t_2 = C^k_{j+l'-1} - \sum_{a=i}^{i+l-1} p^k_{\pi_k(a)}$$

Outline

1 Introduction
- Problem
- Motivation
- Literature

2 Method
- Iterated Local Search
- Neighborhoods
- Move Evaluation

3 Computational Experiments

4 Conclusions and Future Work

Results

- Coded in C++
- Intel Core i7 with 3.40 GHz and 16 GB of RAM
- Ubuntu Linux 12.04

Results

- Problems / instances
  - Publicly available instances
  - With results reported in the literature
  - We do not report the results found for some single machine problems without sequence-dependent setup times
  - Well solved by Tanaka *et al.* (2009); Tanaka and Fujikuma (2012)

Problem	#Instances	$\|J\|$	$\|M\|$
$P\|\|\sum T_j$	2250	20 and 25	2 to 10
$P\|\|\sum w_j T_j$	1125	40 to 100	$2, 4, 10, 20$ and 30
$P\|\|\sum w'_j E_j + w_j T_j$	5000	40 to 500	$2, 4, 10, 20$ and 30
$R\|\|\sum w_j T_j$	1440	40 to 200	2 to 5
$R\|\|\sum w'_j E_j + w_j T_j$	1440	40 to 200	2 to 5

Results

- Neighborhoods parameters
  - $L_{intra} = \{1, 2\}$
  - $L'_{intra} = \{(1, 1)\}$
  - $L_{inter} = \{1, 2\}$
  - $L'_{inter} = \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)\}$
- UILS parameters
  - Number of restarts: 10
  - Number of consecutive perturbations without improvements: $4n$ and n (with IT)

Results

- $P||\sum T_j$
  - 2250 instances proposed by Tanaka and Araki (2008)
  - $n = \{20, 25\}$ jobs
  - $m = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ identical parallel machines
  - Optimal solutions available
  - The UILS algorithm was capable of finding all optimal solutions
  - In at least 9 of the 10 runs
  - Average computational time: smaller than one second

Results

- $P||\sum w_j T_j$
  - Instances proposed by Rodrigues *et al.* (2008)
  - $n = \{40, 50, 100\}$ jobs
  - $m = \{2, 4, 10\}$ machines
  - 25 instances for each (n, m)
  - Heuristics and optimal solutions

Instance group	Rodrigues *et al.* (2008)				UILS				
	Best run		Average		Best run			Average	
	Gap (%)	#BKS	Gap (%)	Time[1] (s)	Gap (%)	#Equal to BKS	#Imp	Gap (%)	Time (s)
wt40-2m	0.00	24	0.87	11.4	0.00	25	0	0.01	3.1
wt40-4m	0.00	25	3.29	38.2	0.00	25	0	0.00	4.0
wt50-2m	0.00	25	1.85	28.3	0.00	25	0	0.00	6.5
wt50-4m	0.00	25	3.08	96.9	0.00	25	0	0.00	8.2
Total	–	99	–	–	–	100	0	–	–
Avg.	0.00	–	2.27	43.7	0.00	–	–	0.00	5.5

[1] Average of $30 \times m \times n$ runs in an Intel Xeon 2.33 GHz.

Instance group	Pessoa *et al.*(2010)		UILS				
	Best run		Best run			Average	
	Gap (%)	#BKS	Gap (%)	#Equal to BKS	#Imp	Gap (%)	Time (s)
wt100-2m	0.00	21	0.00	22	2	0.00	71.1
wt100-4m	0.00	16	-0.01	15	5	0.00	94.8
Total	–	37	–	37	7	–	–
Avg.	0.00	–	0.00	–	–	0.00	83.0

- $P||\sum w'_j E_j + w_j T_j$
  - Instances proposed by Amorim (2013)
  - Heuristics and optimal solutions
  - 11 or 12 instances for each (n, m)
  - Idle times not considered

Instance Group	ILS+PR				GA+LS+PR				ILS-M				UILS				
	Best run		Average		Best run		Average		Best run		Average		Best run			Average	
	Gap (%)	#BKS	Gap (%)	Time[1] (s)	Gap (%)	#BKS	Gap (%)	Time[1] (s)	Gap (%)	#BKS	Gap (%)	Time[1] (s)	Gap (%)	#Equal to BKS	#Imp	Gap (%)	Time (s)
wet40-2m	0.00	12	0.00	15.4	0.00	12	0.01	33.6	0.00	12	0.00	13.6	-0.58	11	1	-0.58	5.6
wet50-2m	0.00	12	0.00	41.2	0.00	12	0.02	99.1	0.00	12	0.00	32.2	-0.78	11	1	-0.78	12.6
wet100-2m	0.02	5	0.02	588.0	0.00	10	0.01	3098.2	0.00	11	0.00	533.0	-0.67	10	1	-0.66	168.5
wet40-4m	0.74	11	0.74	68.4	0.74	11	0.75	201.1	0.74	11	0.74	37.0	0.00	12	0	0.00	6.3
wet50-4m	0.00	10	0.00	163.1	0.02	8	0.02	546.8	0.00	10	0.00	73.5	-0.91	10	1	-0.63	14.1
wet100-4m	0.14	2	0.16	3047.1	0.01	4	0.01	22689.9	0.01	8	0.03	1876.4	0.08	5	1	0.15	190.3
wet40-10m	0.00	5	0.00	325.6	0.00	5	0.00	4780.4	0.00	5	0.00	46.9	0.00	5	0	0.00	4.1
wet50-10m	0.03	4	0.03	852.7	0.00	5	0.01	10237.8	0.00	5	0.01	130.0	-0.60	4	1	-0.59	9.3
wet100-10m	1.43	0	1.45	16817.9	1.19	1	1.19	132735.9	1.19	0	1.23	8424.8	-0.01	0	1	0.05	140.4
Avg.	0.26	–	0.27	2435.5	0.22	–	0.22	19380.3	0.22	–	0.22	1240.8	-0.39	–	–	-0.34	61.3
Total	–	61	–	–	–	68	–	–	–	74	–	–	–	68	7	–	–

[1] Average of 3 runs in an Intel i7-3770 3.40GHz with 12 GB of RAM.

- $R||\sum w_j T_j$
  - Instances proposed by Sen and Bülbül (2015)
  - Heuristic, optimal solutions and lower bounds
  - 6 groups of instances, each of them containing 60 test-problems

Instance group	Sen and Bülbül (2015) Best #Opt	Sen and Bülbül (2015) Best Gap_{LB} (%)	PR+Benders+SiPS #Opt	PR+Benders+SiPS #Equal to BKS	PR+Benders+SiPS Gap_{LB} (%)	PR+Benders+SiPS Time (s)	UILS Best run #Opt	UILS Best run #Imp	UILS Best run #Equal to BKS	UILS Best run Gap_{LB} (%)	UILS Average Gap_{LB} (%)	UILS Average Time (s)
N40_M2	17	1.24	0	13	1.76	1.9	17	20	40	0.76	0.76	3.4
N60_M2	8	1.2	0	40	1.46	6.4	8	41	19	0.63	0.63	11.7
N60_M3	6	4.27	0	35	4.62	3.0	6	37	23	3.13	3.13	15.4
N80_M2	3	1.43	0	55	1.52	13.2	3	55	5	0.66	0.66	29.4
N80_M4	5	4.8	3	48	4.93	43.1	4	48	11	3.83	4.26	41.7
N90_M3	1	3.05	1	56	3.11	8.0	1	56	4	1.18	1.24	57.1

Results

- $R||\sum w'_j E_j + w_j T_j$
  - Same Instances proposed by Sen and Bülbül (2015)
  - Heuristic, optimal solutions and lower bounds
  - 6 groups of instances, each of them containing 60 test-problems
  - Idle times are permitted

Instance group	Sen and Bülbül (2015)						UILS					
	Best		PR+Benders+SiPSi				Best run				Average	
	#Opt	Gap_{LB} (%)	#Opt	#Equal to BKS	Gap_{LB} (%)	Time (s)	#Opt	#Imp	#Equal to BKS	Gap_{LB} (%)	Gap_{LB} (%)	Time (s)
N40_M2	22	0.16	1	1	0.95	52.9	22	13	47	0.13	0.13	21.1
N60_M2	5	0.89	0	42	0.98	109.7	5	50	10	0.43	0.45	126.8
N60_M3	4	0.82	0	20	1.58	120.3	4	37	23	0.38	0.44	105.0
N80_M2	2	0.90	0	56	0.92	134.1	2	57	3	0.36	0.38	476.2
N80_M4	0	4.54	0	58	4.57	228.3	0	60	0	1.59	1.72	312.2
N90_M3	0	2.52	0	58	2.55	153.1	0	59	1	1.27	1.37	574.0

Outline

1 Introduction
- Problem
- Motivation
- Literature

2 Method
- Iterated Local Search
- Neighborhoods
- Move Evaluation

3 Computational Experiments

4 Conclusions and Future Work

Conclusions and Future Work

- Conclusions
  - Algorithm for a class of problems
  - Takes into account the particularities of each problem (move evaluation)
  - Generalization of efficient local search methods
  - The method was capable of finding high quality solutions in competitive CPU times
  - Paper currently under second revision
  - More details in: `http://arxiv.org/abs/1509.02384`
- Future Work
  - Generation of new set of instances for the problems where it was not possible to test the proposed algorithm
  - Development of a general exact method capable of tackling the variants considered in this work
  - Lower bounds for the instances to be created

Thank you

A Local Search Based Metaheuristic for a Class of Earliness and Tardiness Scheduling Problems

Arthur Kramer[1,2]
Anand Subramanian[2]

[1]Università degli Studi di Modena e Reggio Emilia - Italy
[2] Universidade Federal da Paraíba - Brazil

March 31, 2016

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA