EL PROBLEMA DEL TALLER DE FLUJO (*FLOWSHOP SCHEDULING PROBLEM*)

Rubén Ruiz

INSTITUTO TECNOLÓGICO DE INFORMÁTICA. GRUPO DE SISTEMAS DE OPTIMIZACIÓN APLICADA.
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SOA
SISTEMAS DE OPTIMIZACIÓN APLICADA

Contents

1. Introduction

- Scheduling is a very active topic of research in operations research
- SciVal Trends. Top 50 keyphrases by relevance, based on 111,262 publications in the field of Management Science and Operations Research, from 2009 to 2018:

1. Introduction

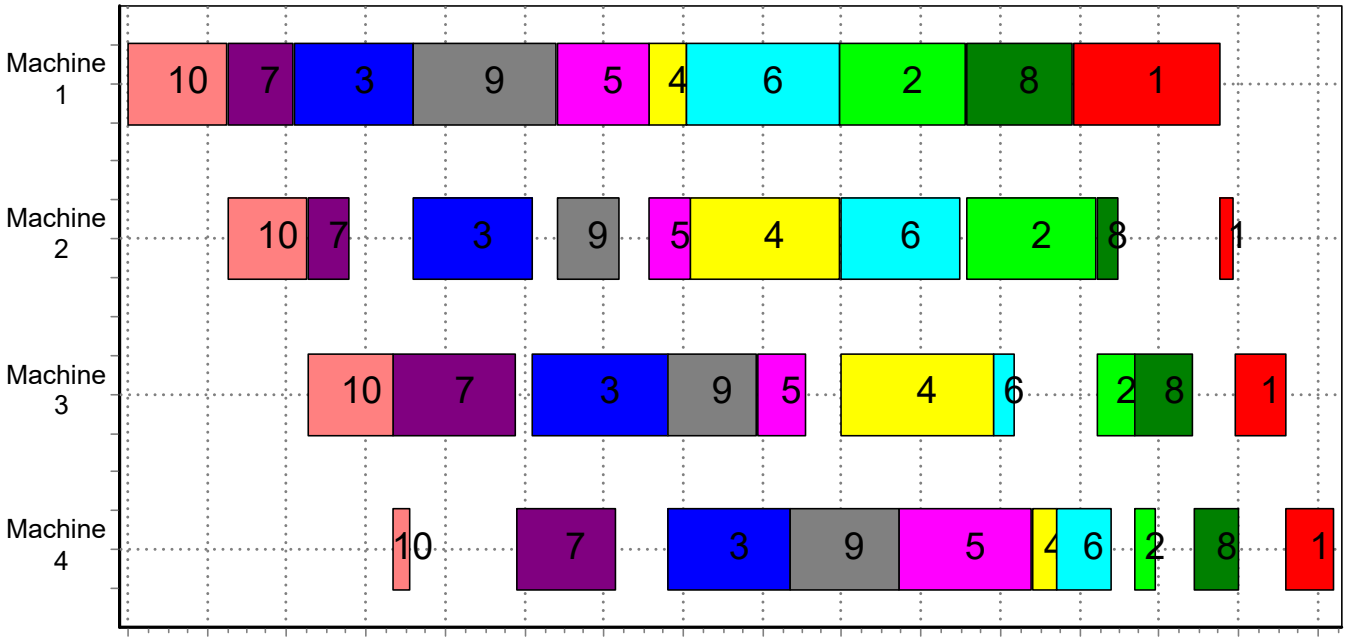
- A flowshop is a very common production layout
- In a flowshop there are n jobs that have to be processed, in the same order, in m machines
- A job is then comprised of m tasks, one per machine
- A machine cannot process more than one job at the same time and one job cannot be processed by more than one machine at the same time

1. Introduction

- Each job needs a non-negative processing time at each machine, denoted by p_{ij}
- Usually, job passing is not allowed from machine to machine: permutation flowshop problem
- The most common objective is the minimization of the maximum completion time or makespan

1. Introduction

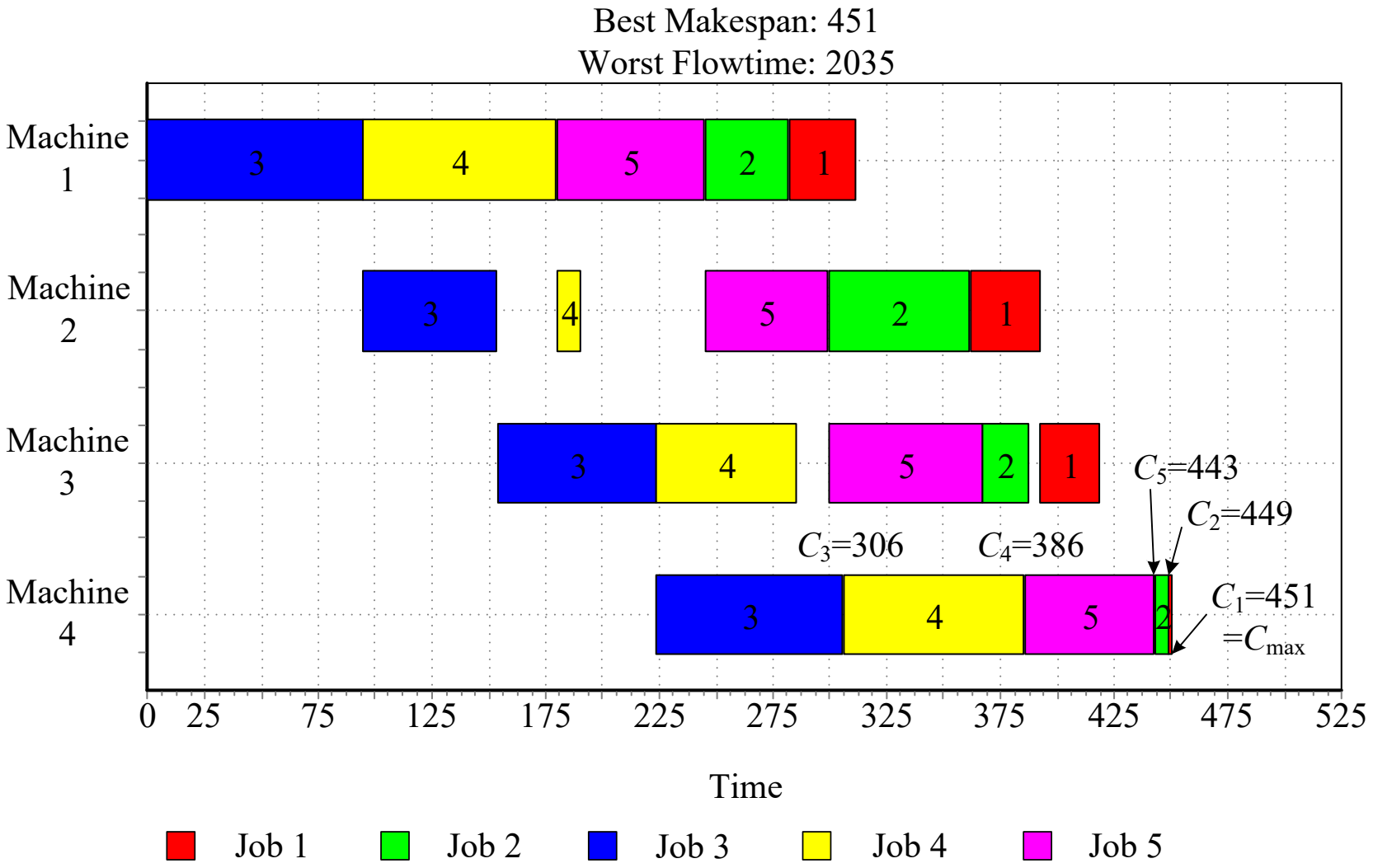
- The flowshop with makespan criterion is already an NP-Hard problem for $m \geq 3$



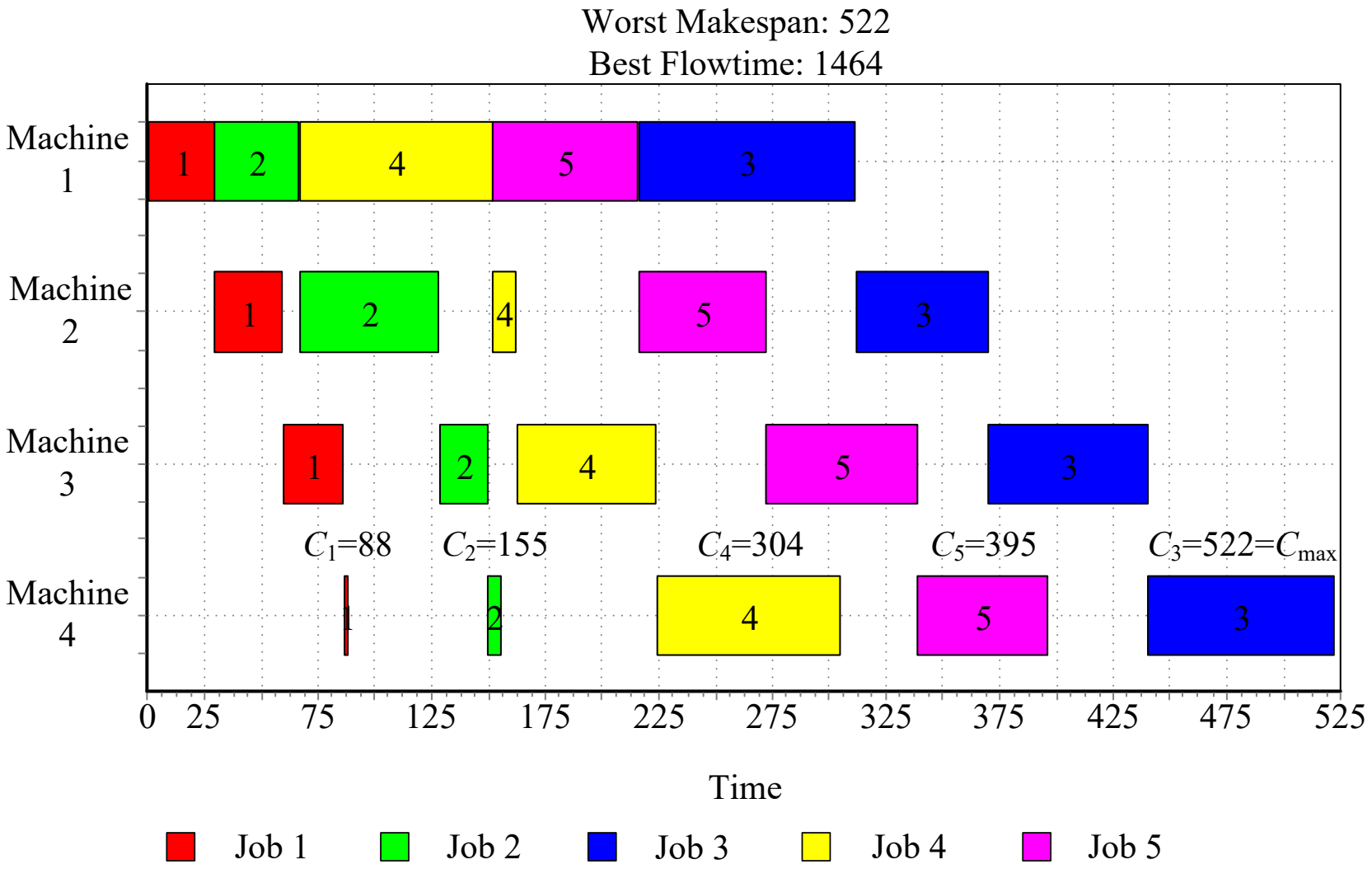
1. Introduction

- Makespan maximizes machine utilization
- Minimizing the sum of job's completion times minimizes WIP and maximizes throughput
- In practice, total flowtime is a more realistic objective

1. Introduction



1. Introduction



2. Flowtime heuristics

- More than 40 papers that specifically propose heuristics for the flowtime PFSP
- Framinan et al. (2005) propose a classification between simple and composite methods
- Only the highest performing methods are evaluated in the computational comparison

2. Flowtime heuristics

- CDS of Campbell et al. (1970)
- MINIT, MICOT and MINIMAX of Gupta (1972)
- Krone and Steiglitz (1974)
- Miyazaki et al. (1978)
- Miyazaki and Nishiyama (1980)
- Ho and Chang (1991)
- Rajendran and Chaudhuri (1991, 1992)
- Raj of Rajendran (1993)
- Ho (1995)
- LIT, SPD1 and SPD2 of Wang et al. (1997)
- RZ of Rajendran and Ziegler (1997)
- WY of Woo and Yim (1998)

2. Flowtime heuristics

- LR(x), LR(x)-FBE and LR(x)-BPE of Liu and Reeves (2001)
- NEH-Flowtime of Framinan et al. (2002)
- IH1-IH7 of Allahverdi and Aldowaisan (2002)
- B5FT of Framinan et al. (2003)
- FL and IH7-FL of Framinan and Leisten (2003)
- C1-FL and C2-FL of Framinan et al (2005)
- RZ-LW of Li and Wu (2005)
- ECH1 and ECH2 of Li and Wang (2006)
- IC1-IC3 of Li et al. (2009)
- FL-LS of Laha and Sarin (2009)

2. Flowtime heuristics

- NEH of Nawaz et al. (1983) a very capable heuristic for the FPSP, mainly for makespan criterion
- LR(x) of Liu and Reeves (2001) very powerful for flowtime criterion
- Simple idea: Why not combining them?

2. Flowtime heuristics

Procedure LR-NEH(x)

Generate a job sequence $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value)

for $l := 1$ **to** x **do** %(generate x sequences)

$\pi^l := \{\alpha_l\}$, $U := J - \{\alpha_l\}$.

for $k := 2$ **to** d **do** %(construct a partial sequence with d jobs)

Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.

endfor

% (NEH heuristic)

Generate a partial sequence $\beta = \{\beta_1, \beta_2, ..., \beta_{n-d}\}$ ($\beta_j \in U$, $j = 1,2,...,n-d$) by ascending order of total processing times.

for $k := 1$ **to** $n-d$ **do** %(construct a complete sequence)

Take job β_k from β and insert it in all the $k+d$ possible positions of π^l.

Place job β_k in π^l at the tested position resulting in the lowest total flowtime.

endfor

endfor

return the sequence $\pi \in \{\pi^1, \pi^2, ..., \pi^l\}$ with the minimum total flowtime

2. Flowtime heuristics

- As with LR(x), LR-NEH(x) has the parameter x
- It also has a parameter d, which has been set to $3n/4$
- We also propose other 4 composite heuristics. All of them start from LR-NEH(x)
- Different combinations of the local search scheme of Rajendran and Ziegler (1997). With accelerations and to optimality: iRZ and VNS of Mladenovic and Hansen (1997)

2. Flowtime heuristics

- PR1(x)
  - Each one of the x solutions in LR-NEH(x) are improved by iRZ
- PR2(x)
  - iRZ is substituted by a simple VNS based on insertion and interchange neighborhoods

2. Flowtime heuristics

- PR3(x)
  - Local search is composed by iRZ and two NEH-like local searches
- PR4(x)
  - Same as PR3(x) but iRZ is substituted by the VNS
- Essentially simple methods and local search schemes

2. Flowtime heuristics

- We test 14 simple heuristics:
  1. Raj heuristic of Rajendran (1993),
  2-4. LIT, SPD1 and SPD2 heuristics by Wang et al. (1997),
  5. RZ heuristic of Rajendran and Ziegler (1997),
  6. WY heuristic by Woo and Yim (1998),
  7-9. LR(1), LR(n/m) and LR(n) of Liu and Reeves (2001),
  10. NEH heuristic modified by Framinan et al. (2002),
  11. FL heuristic of Framinan and Leisten (2003),
  12. RZ-LW heuristic of Li and Wu (2005),
  13. FL-LS heuristic by Laha and Sarin (2009),
  14. Proposed LR-NEH(x) heuristic.

2. Flowtime heuristics

- And 13 composite heuristics:

  15-16. LR-FPE and LR-BPE of Liu and Reeves (2001),

  17. IH7 heuristic of Allahverdi and Aldowaisan (2002),

  18. IH7-FL heuristic of Framinan and Leisten (2003),

  19-20. Composite heuristics C1-FL and C2-FL of Framinan et al. (2005),

  21-23. IC1, IC2, IC3 heuristics of Li et al. (2009),

  24-27. The presented composite heuristics PR1(x), PR2(x), PR3(x) and PR4(x).

2. Flowtime heuristics

- Other methods (especially the oldest ones) were shown in previous studies to be clearly worse than other tested methods
- Accelerations and flowtime computing method of Li et al. (2006) employed to save computational time
- 120 instances of Taillard (1993)
  - n={20, 50, 100} × m={5,10,20},
  - n={200} × m={10,20} and n={500} × m={20}
- Average relative percentage improvement from best solution known as a response measure (RPI)

2. Flowtime heuristics

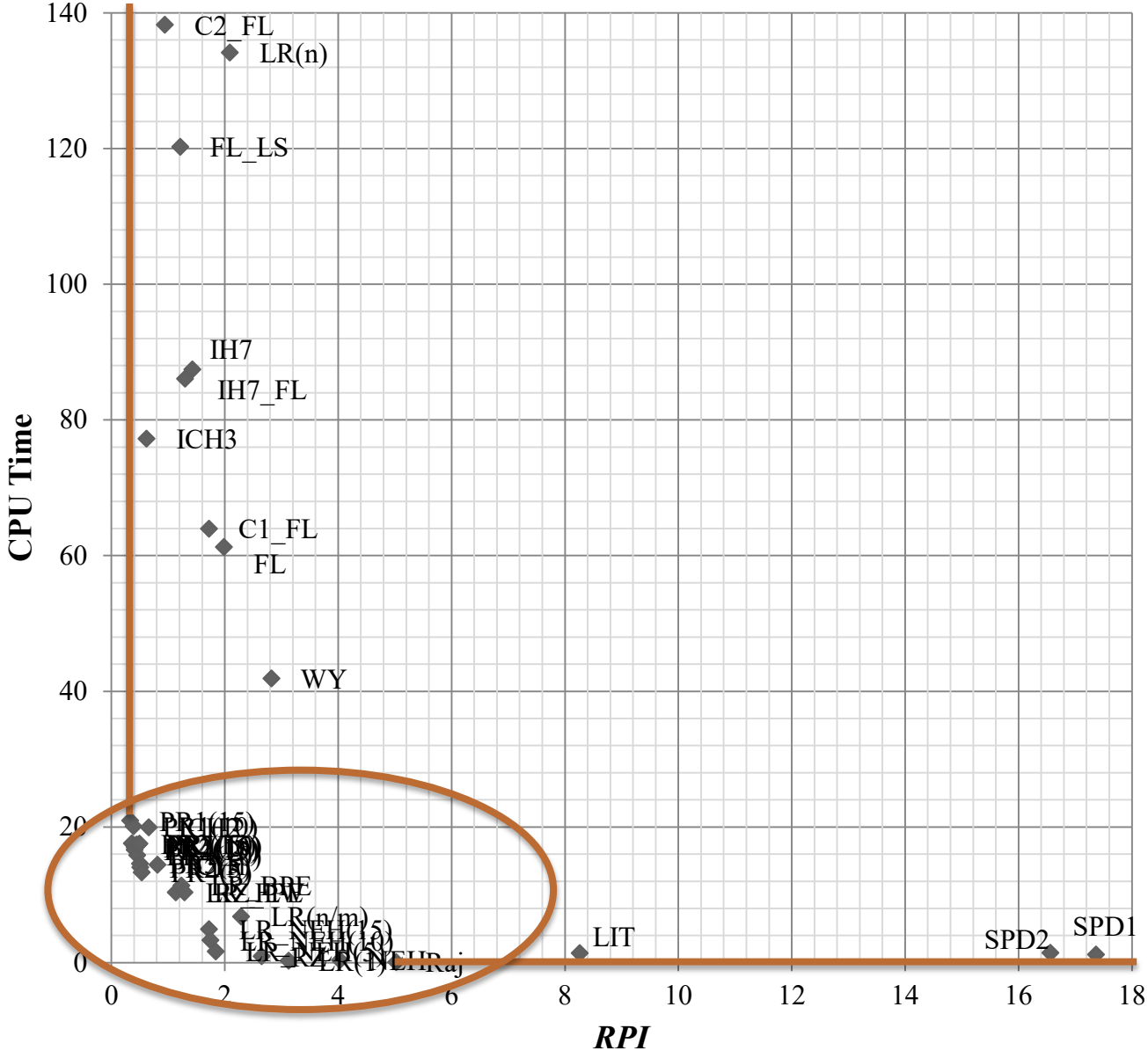
- All algorithms coded in C++
- Runs on a cluster of 30 blade servers each one with two Intel XEON E5420 processors running at 2.5 GHz and with 16 GB of RAM memory
- All methods are deterministic. However, 5 different runs are carried out in order to better estimate the CPU time
- Our tested methods LR-NEH(x), PR1(x)-PR4(x) are tested with three values of x = 5, 10 and 15
- Therefore, 37 algorithms $\times$ 5 replicates $\times$ 120 instances =22,200 results

2. Flowtime heuristics

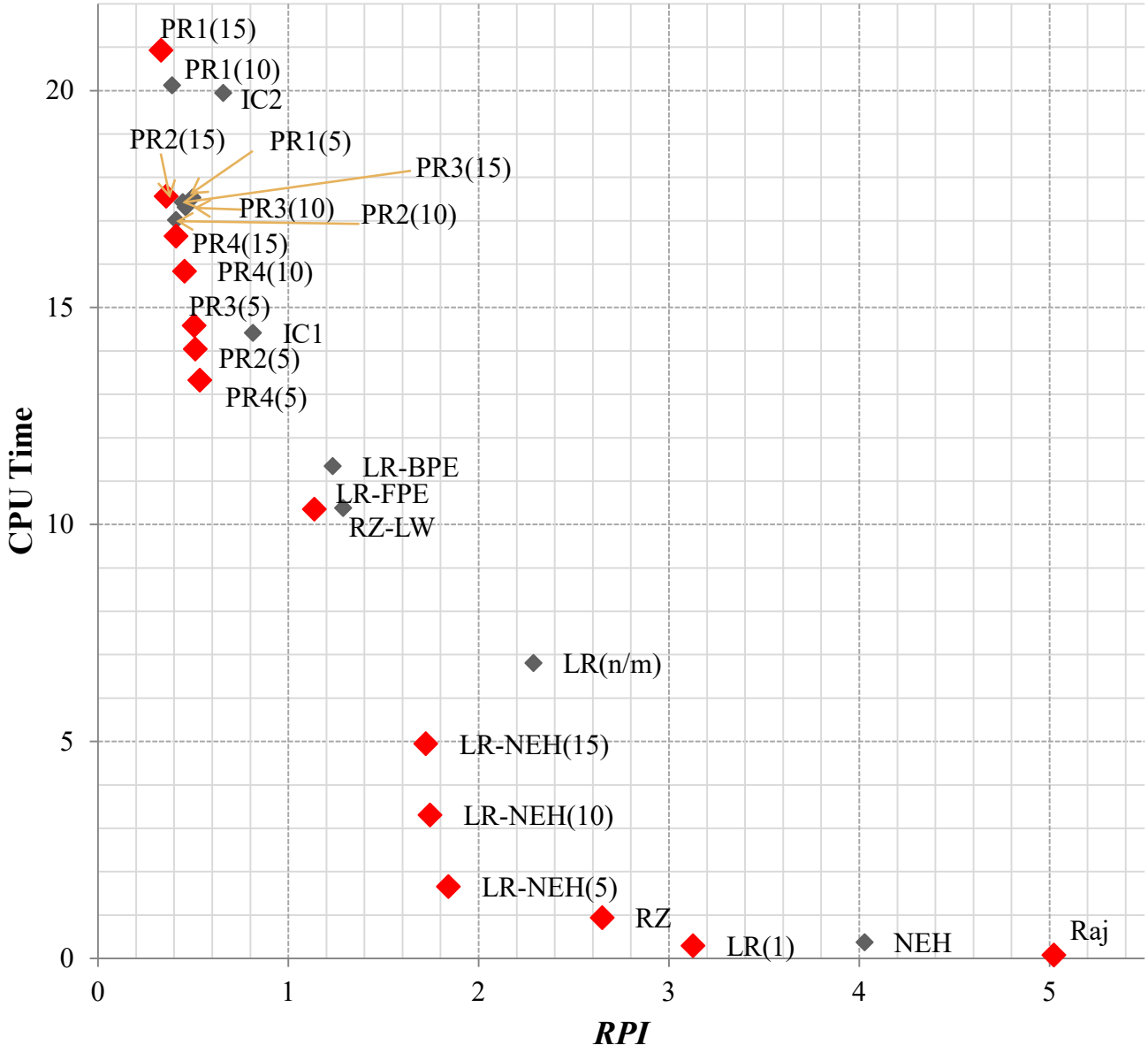
#	Algorithm	RPI	Time	Type	PARETO
1	**PR1(15)**	0.33	20.93	Composite	0
2	**PR2(15)**	0.36	17.56	Composite	0
3	PR1(10)	0.39	20.12	Composite	1
4	PR2(10)	0.41	17.02	Composite	1
5	**PR4(15)**	0.41	16.64	Composite	0
6	PR3(15)	0.45	17.43	Composite	2
7	**PR4(10)**	0.45	15.84	Composite	0
8	PR3(10)	0.46	17.31	Composite	3
9	PR1(5)	0.50	17.53	Composite	5
10	**PR3(5)**	0.51	14.58	Composite	0
11	**PR2(5)**	0.51	14.04	Composite	0
12	**PR4(5)**	0.53	13.32	Composite	0
13	IC3	0.62	77.25	Composite	12
14	IC2	0.66	19.95	Composite	10
15	IC1	0.81	14.41	Composite	2
16	C2-FL	0.95	138.22	Composite	15
17	**LR-FPE**	1.14	10.35	Composite	0
18	FL-LS	1.22	120.24	Simple	16
19	LR-BPE	1.23	11.34	Composite	1

#	Algorithm	RPI	Time	Type	PARETO
20	RZ-LW	1.29	10.38	Simple	1
21	IH7-FL	1.30	86.06	Composite	18
22	IH7	1.43	87.45	Composite	19
23	C1-FL	1.72	63.98	Composite	17
24	**LR-NEH(15)**	1.72	4.94	Simple	0
25	**LR-NEH(10)**	1.75	3.30	Simple	0
26	**LR-NEH(5)**	1.84	1.65	Simple	0
27	FL	1.99	61.24	Simple	20
28	LR(n)	2.09	134.12	Simple	26
29	LR(n/m)	2.29	6.81	Simple	3
30	**RZ**	2.65	0.94	Simple	0
31	WY	2.83	41.90	Simple	22
32	**LR(1)**	3.13	0.29	Simple	0
33	NEH	4.03	0.37	Simple	1
34	**Raj**	5.02	0.08	Simple	0
35	LIT	8.26	1.42	Simple	4
36	SPD2	16.56	1.43	Simple	5
37	SPD1	17.37	1.18	Simple	4

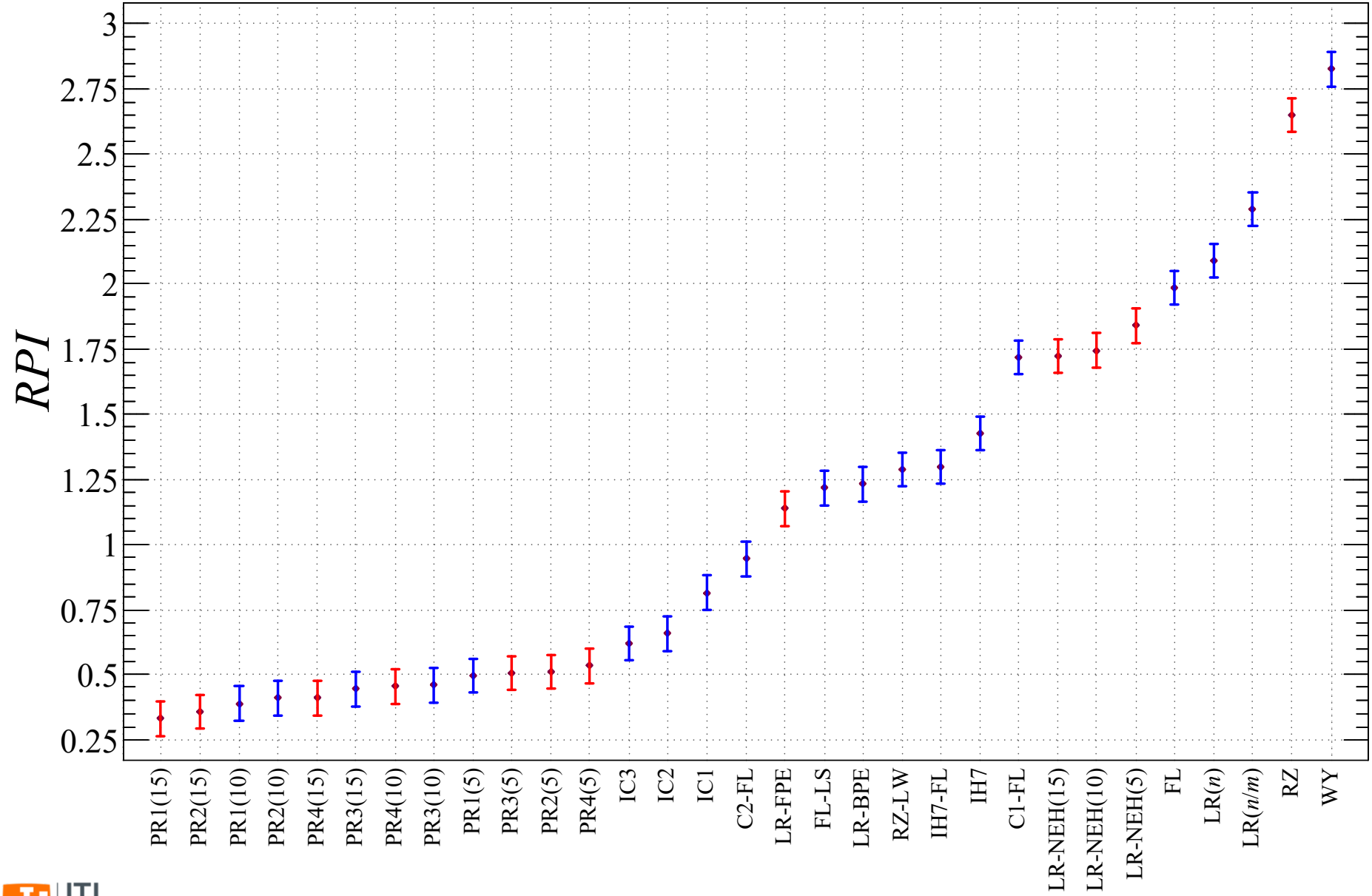
2. Flowtime heuristics



2. Flowtime heuristics



2. Flowtime heuristics



2. Flowtime heuristics

- Our presented methods range from fast and high performance simple heuristics, LR-NEH(x) to slower but with state-of-the-art performance
- PR1(5) results in 0.5% deviation from best known solutions in less than 18 seconds on average (less than 3 seconds for instances up to 100 jobs and 20 machines)

3. Flowtime local search

- Can we improve the results further?
- Most proposed methods are fairly complex
- To propose simple and easy to implement new state-of-the-art methods for the problem
- To carry out a comprehensive computational study for the problem

3. Flowtime local search

- We are interested in simple approaches
- Iterated local Search (ILS) of Lourenço et al. (2003) and Iterated Greedy (IG) of Ruiz and Stützle (2007) have shown competitive results in similar settings
- We also present population extensions of both approaches

3. Flowtime local search

procedure ILS

$\pi_0 \leftarrow GenerateInitialSolution$

$\pi \leftarrow LocalSearch(\pi_0)$ % Local search

repeat

$\pi' \leftarrow Perturbation(\pi)$ % Perturbation of the local optimum

$\pi'' \leftarrow LocalSearch(\pi')$ % Local search

$\pi \leftarrow Acceptcriterion(\pi'', \pi)$ % Decide if new solution replaces the incumbent

until termination criterion met

end

3. Flowtime local search

- INITIAL SOLUTION: We employ the very efficient heuristic LR(n/m) of Li et al. (2009) and Zhang et al (2009). This is an extension of the LR(x) heuristic of Liu and Reeves (2001)
- LOCAL SEARCH: We employ the RZ procedure of Rajendran and Ziegler (1997). With accelerations and to optimality: iRZ

3. Flowtime local search

- PERTURBATION: A given number γ of random insertion moves
- ACCEPTANCE CRITERION: Simulated annealing-like with constant temperature factor λ, as done by Stützle (1998) or Ruiz and Stützle (2007, 2008) and others

3. Flowtime local search

procedure the presented ILS algorithm

Set the parameters γ *and* λ

$\pi_0 \leftarrow LR(n/m)$ % Generate an initial solution

$\pi \leftarrow iRZ(\pi_0)$ % Local search

$\pi^* \leftarrow \pi$ % Best solution found so far

repeat

$\pi' \leftarrow Perturbation(\pi)$ % Perturbation of the local optimum

$\pi'' \leftarrow iRZ(\pi')$ % Local search

if $\sum C_j(\pi'') < \sum C_j(\pi)$ **then** % Acceptance criterion

$\pi \leftarrow \pi''$ % Accept if better than incumbent

if $\sum C_j(\pi'') < \sum C_j(\pi^*)$ **then** % check if new best solution

$\pi^* \leftarrow \pi''$

endif

elseif $rand(\,) \le \exp\{(\sum C_j(\pi) - \sum C_j(\pi''))/Temperature\}$ **then**

$\pi \leftarrow \pi''$ % Simulated annealing acceptance criterion

endif

until *termination criterion is met*

end

3. Flowtime local search

procedure IGA

$\pi_0 \leftarrow GenerateInitialSoulution$ % Generate an initial solution

$\pi \leftarrow LocalSearch(\pi_0)$ % Local search

repeat

$\pi' \leftarrow Destruction\_Construction(\pi)$ % Destruction and construction

$\pi'' \leftarrow LocalSearch(\pi')$ % Local search

$\pi \leftarrow Acceptcriterion(\pi'', \pi)$ % Decide if new solution replaces the incumbent

until termination criterion is met

end

3. Flowtime local search

- INITIAL SOLUTION, LOCAL SEARCH and ACCEPTANCE CRITERION do not change
- DESTRUCTION_CONSTRUCTION: Identical to Ruiz and Stützle (2007). d jobs are randomly selected and extracted. Then they are inserted, one by one in all positions of the sequence (as in the NEH heuristic)

3. Flowtime local search

- POPULATION VARIANTS pILS and pIGA
- Initial population of size x using LR(x)
- At each generation, one solution from the population is selected:
  - 50% of the times using binary tournament with flowtime value
  - 50% of the times using inverted binary tournament with the age of the solution

3. Flowtime local search

- GENERATIONAL SCHEME
- A new solution replaces the worst solution of the population iff:
  1. Flowtime value better than that of the worst
  2. There is no other identical solution in the population (permutation)
  3. The average diversity value of the population does not decrease (Pan and Ruiz, 2011)

3. Flowtime local search

- DIVERSITY CALCULATION

$$\left[\phi_{i,j}\right]_{n\times n} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,n} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,n} \\ \cdots & \cdots & \ddots & \cdots \\ \phi_{n,1} & \phi_{n,2} & \cdots & \phi_{n,n} \end{bmatrix}$$

$\phi_{i,j}$ = number of times job j appears at position i of the sequence

$$\left[\lambda_{j',j}\right]_{n\times n} = \begin{bmatrix} - & \lambda_{1,2} & \cdots & \lambda_{1,n} \\ \lambda_{2,1} & - & \cdots & \lambda_{2,n} \\ \cdots & \cdots & \ddots & \cdots \\ \lambda_{n,1} & \lambda_{n,2} & \cdots & - \end{bmatrix}$$

$\lambda_{j',j}$ = number of times job j appears inmediately after job j'

3. Flowtime local search

- α and β are the number of elements that are larger than zero in both matrices, respectively
- Then the diversity is calculated as:

$$div = \left(\frac{\alpha - n}{n \times \min(n, x-1)} + \frac{\beta - (n-1)}{(n-1) \times \min(n-1, x-1)} \right) \Big/ 2$$

3. Flowtime local search

- We test 16 different algorithms:
  1. Discrete Differential Evolution DDE_{RLS} of of Pan et al. (2008)
  2. Iterated Greedy IG_{RLS} of Pan et al. (2008)
  3. Estimation of Distribution EDA_J of Jarboui et al. (2009)
  4. Variable neighborhood search VNS_J of Jarboui et al. (2009)
  5. Iterated local search ILS_D of Dong et al. (2009)
  6. Hybrid genetic algorithm HGA_{T1} of Tseng and Lin (2009)
  7. Hybrid genetic algorithm HGA_Z of Zhang et al. (2009)
  8. Hybrid genetic algorithm HGA_{T2} of Tseng and Lin (2010)
  9. Genetic Local Search AGA of Xu et al. (2011)
  10. Hybrid Discrete Differential Evolution hDDE of Tasgetiren et al. (2011)
  11. Discrete Ant Bee Colony DABC of Tasgetiren et al. (2011)
  12. Iterated Greedy SLS of Dubois-Lacoste et al. (2011)
  13-16. Proposed ILS, IGA, pILS and pIGA

3. Flowtime local search

- 120 instances of Taillard (1993)
  - n={20, 50, 100}×m={5,10,20},
  - n={200}×m={10,20} and n={500}×m={20}
- We add 30 more instances with the missing n={200}×m={5} and n={500}×m={5,10}
- Average relative percentage deviation from best solution known as a response measure (AVRPD)
- All algorithms coded in C++, and run 5 times on each instance during $t = \rho mn$ milliseconds elapsed CPU time on Intel XEON E5420 processors running at 2.5 GHz and with 16 GB of RAM memory

3. Flowtime local search

SOA
SISTEMAS DE OPTIMIZACIÓN APLICADA

ρ=30

	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}
AVRPD	0.39	0.39	*7.72*	4.88	0.49	2.23	0.74	5.29

	AGA	hDDE	DABC	SLS	IGA	pIGA	ILS	pILS
AVRPD	0.87	0.64	0.83	0.41	**0.24**	0.28	0.25	0.31

ρ=60

	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}
AVRPD	0.36	0.36	*7.02*	4.39	0.49	2.13	0.63	4.50

	AGA	hDDE	DABC	SLS	IGA	pIGA	ILS	pILS
AVRPD	0.78	0.60	0.76	0.41	**0.24**	0.27	0.25	0.30

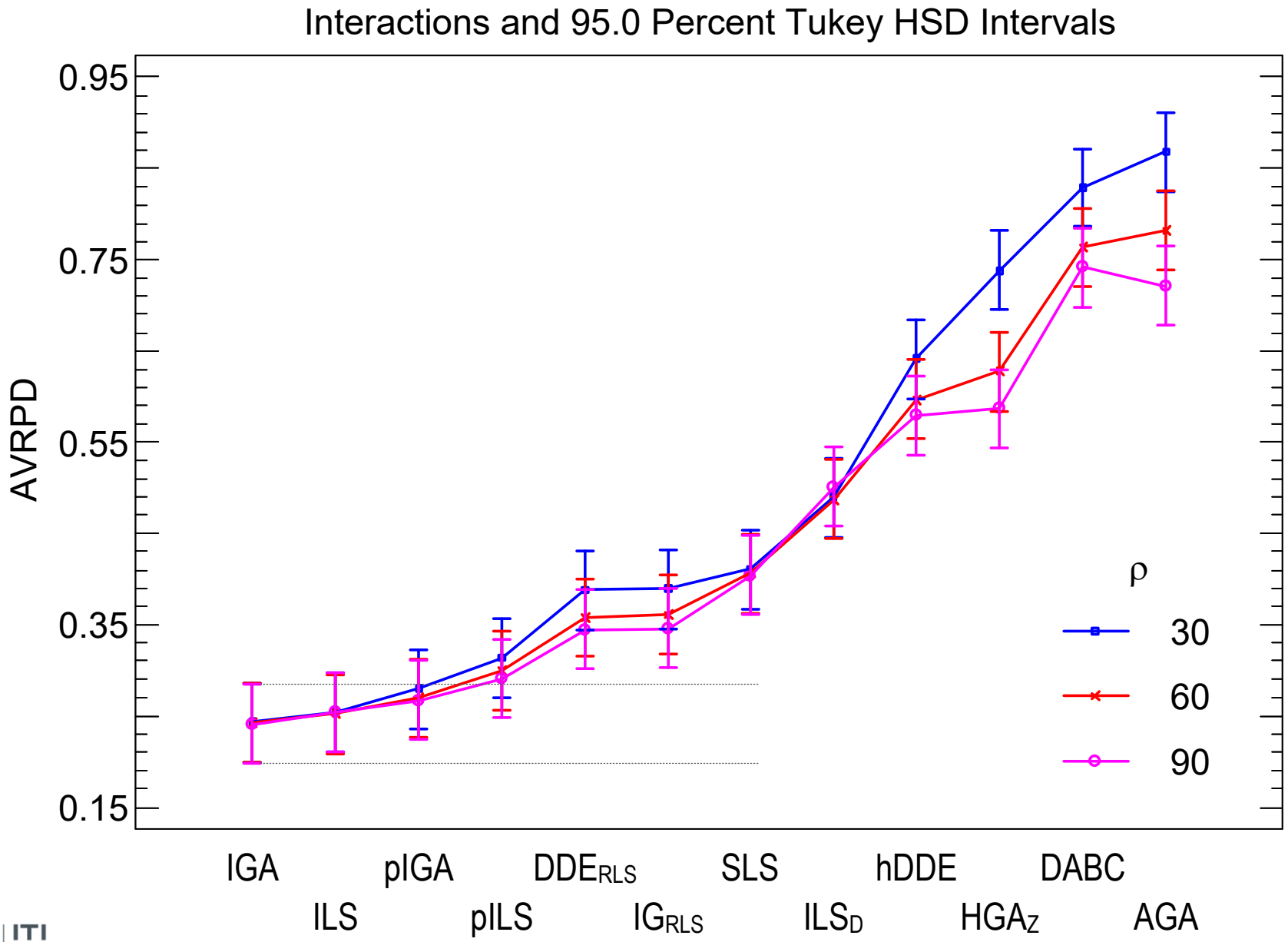
ρ=90

	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}
AVRPD	0.35	0.34	*6.64*	4.17	0.50	2.09	0.59	4.09

	AGA	hDDE	D	S	IGA	pIGA	ILS	pILS
AVRPD	0.72	0.58	0.74	0.40	**0.24**	0.27	0.25	0.29

42% Better

3. Flowtime local search



3. Flowtime local search

- We have studied the permutation flowshop scheduling problem with total flowtime criterion
- We have proposed four simple methods based on local search, Iterated Greedy and Iterated Local Search
- We have carried out a comprehensive comparison against 12 other methods
- IGA and ILS are state-of-the-art methods

3. Flowtime local search

- IGA and ILS better than other much more complex algorithms
- pIGA and pILS also good, but not better than the simpler non population variants
- Future work:
  - Sequence dependent setup times and flowtime objective seldom studied
  - Earliness-Tardiness objectives
  - Hybrid flowshops

4. Back to mathematical modeling?

- MIP models have been gradually abandoned and have given way to heuristics and, most notably, metaheuristics
- However, solvers keep improving. According to Bixby (2017, ECCO conference), since 1991 solvers are now 1.3 million times faster (accumulated) not considering hardware improvements

4. Back to mathematical modeling?

SOA
SISTEMAS DE OPTIMIZACIÓN APLICADA

- Constraint Programming (CP) is gaining traction in scheduling as of late (Bukchin and Raviv, 2018, Laborie et al., 2018)
- CP models are notorious for their ability to find integer solutions
- However, classical CP lacks bounding mechanisms and therefore does not provide optimality guarantees or gaps
- Since CPLEX 12.8, CP Optimizer guarantees bounds and optimality GAPs for any integer solution found

4. Back to mathematical modeling?

- How do modern solvers and formulations fare for different scheduling problems?
- How does CP Optimizer compare with a regular MIP solver?
- A exact approach renaissance for scheduling problems?

4. Back to mathematical modeling?

- Manne (1960) MIP:

$$
\begin{aligned}
\text{minimize} \quad & c_{\max} & & \\
\text{subject to:} \quad & c_{ji} \geq P_{ji} & \forall j \in \mathcal{J}, i \in \mathcal{I} & (1)\\
& c_{ji} \geq c_{ji-1} + P_{ji} & \forall j \in \mathcal{J}, i \in \mathcal{I} \setminus 1 & (2)\\
& c_{ji} \geq c_{j'i} + P_{ji} - M(1 - x_{jj'}) & \forall i \in \mathcal{I}, j, j' \in \mathcal{J} : j > j' & (3)\\
& c_{j'i} \geq c_{ji} + P_{j'i} - M(x_{jj'}) & \forall i \in \mathcal{I}, j, j' \in \mathcal{J} : j > j' & (4)\\
& c_{\max} \geq c_{ji} & \forall j \in \mathcal{J}, i \in \mathcal{I} & (5)\\
& c_{ji} \geq 0 & \forall j \in \mathcal{J}, i \in \mathcal{I} & (6)\\
& x_{jj'} \in \{0, 1\} & & (7)
\end{aligned}
$$

4. Back to mathematical modeling?

- CP Optimizer code

$$
\begin{array}{lll}
\text{minimize} & c_{\max} & \\
\text{subject to} & Task_{ji} = \text{cp::IntveralVar}\left(P_{ji}\right) & j \in \mathcal{J}, i \in \mathcal{I} \quad (1) \\
& \text{cp::EndBeforeStart}\left(Task_{ji}, Task_{ji-1}\right) & \forall j \in \mathcal{J}, i \in \mathcal{I} \setminus 1 \quad (2) \\
& \text{cp::NoOverlap}\left(Task_{ji} : j \in \mathcal{J}\right) & \forall i \in \mathcal{I} \quad (3) \\
& SV_i = \text{cp::SequenceVar}\left(Task_{ji} : j \in \mathcal{J}\right) & \forall i \in \mathcal{I} \quad (4) \\
& \text{cp::SameSequence}\left(SV_i, SV_{i-1}\right) & \forall i \in \mathcal{I} \setminus 1 \quad (5) \\
& c_{\max} = \max_{(j,i)} \left(\text{cp::EndOf}(Task_{ji})\right) & \quad (6)
\end{array}
$$

4. Back to mathematical modeling?

- We use Taillard's (1993) benchmark (120 instances) as well as Vallada, Ruiz and Framinan (2015) benchmark with 240 large instances (up to 800 jobs and 60 machines)
- CPLEX 12.8 + Python 3.6.6
- Cluster of 200 virtual machines with 2 virtual processors and 8 GBytes of RAM memory each. The virtual machines run Windows 10 Enterprise 64 bits. Virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each one with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GB of RAM, for a total of 576 cores and 3 TBytes of RAM

4. Back to mathematical modeling?

- CP never runs out of memory and always finds a solution:

	No Solution	Out of memory
CP	**0**	**0**
Taillard		
600	0	0
1800	0	0
3600	0	0
VRF		
600	0	0
1800	0	0
3600	0	0
MIP	**642 (29.7%)**	**187 (8.3%)**
Taillard		
600	40	0
1800	40	0
3600	34	0
VRF		
600	229	11
1800	166	73
3600	133	103

4. Back to mathematical modeling?

- CP still not competitive with metaheuristics (<0.3 RPD)

	Average GAP%	Average RPD
CP	**231.68**	**8.11**
Taillard		
600	6.86	3.62
1800	6.09	2.91
3600	5.66	2.53
VRF		
600	749.24	17.04
1800	416.81	11.79
3600	205.43	10.80
MIP	**361.42**	**12.73**
Taillard		
600	345.50	8.02
1800	315.81	6.11
3600	305.32	5.74
VRF		
600	--	--
1800	417.25	20.84
3600	423.22	22.96

4. Back to mathematical modeling?

- We have tested the following problems:
  - Non-permutation flowshop with makespan criterion
  - No-wait flowshop with makespan criterion
  - Sequence-dependent setup times flowshop with makespan criterion
  - Flowshop with Total Completion Time criterion
  - Distributed flowshop with makespan criterion
  - Hybrid flowshop with makespan criterion
- Each problem with its own benchmark

4. Back to mathematical modeling?

- Non-permutation flowshop with makespan criterion

	Average GAP%	No Solution	Out of memory
CP	**220.27**	**0**	**0**
Taillard			
600	7.40	0	0
1800	6.63	0	0
3600	6.17	0	0
VRF			
600	741.16	0	0
1800	392.91	0	0
3600	167.34	0	0
MIP	**561.02**	**122**	**422**
Taillard			
600	558.38	7	3
1800	534.28	0	10
3600	520.05	0	10
VRF			
600	592.87	111	59
1800	595.45	4	168
3600	565.10	0	172

4. Back to mathematical modeling?

SOA
SISTEMAS DE OPTIMIZACIÓN APLICADA

- No-wait flowshop with makespan criterion

	Average GAP%	No Solution	Out of memory
CP	**877.06**	**0**	**0**
Taillard			
600	270.31	0	0
1800	56.10	0	0
3600	55.43	0	0
VRF			
600	2572.11	0	0
1800	1411.31	0	0
3600	897.13	0	0
MIP	**1674.67**	**312**	**30**
Taillard			
600	1189.40	0	0
1800	1093.97	0	0
3600	1040.93	0	0
VRF			
600	2423.46	107	10
1800	2222.93	107	10
3600	2077.32	98	10

4. Back to mathematical modeling?

- Sequence-dependent setup times flowshop with makespan criterion (480 instances)

	Average GAP%	No Solution	Out of memory
CP	**50.32**	**0**	**0**
600	51.57	0	0
1800	50.09	0	0
3600	49.28	0	0
MIP	**1244.42**	**23**	**28**
600	1314.61	11	0
1800	1291.40	7	0
3600	1127.25	5	28

4. Back to mathematical modeling?

- Flowshop with Total Completion Time criterion (360 instances)

	Average GAP%	No Solution	Out of memory
CP	**557.12**	**0**	**10**
600	702.26	0	1
3600	411.98	0	9
MIP	**295.93**	**404**	**138**
600	308.80	266	14
3600	283.06	138	124

4. Back to mathematical modeling?

- Distributed flowshop with makespan criterion (600 instances)

	Average GAP%	No Solution	Out of memory
CP	**194.24**	**0**	**0**
600	197.43	0	0
1800	191.62	0	0
3600	193.67	0	0
MIP	**187.81**	**516**	**119**
600	199.77	234	20
1800	166.17	131	50
3600	197.49	151	49

4. Back to mathematical modeling?

- Hybrid flowshop with makespan criterion (1440 instances)

	Average GAP%	No Solution	Out of memory
CP	**281.92**	**0**	**0**
600	284.91	0	0
1800	279.54	0	0
3600	281.30	0	0
MIP	**1144.33**	**1263**	**0**
600	1268.79	521	0
1800	1007.86	314	0
3600	1156.33	428	0

4. Back to mathematical modeling?

- CP is vastly superior to the best combination of MIP model + state-of-the-art solver
- CP gives an integer solution in all cases
- CP does not show out of memory problems
- Greatly reduced relative percentage deviations and GAP values in most scenarios
- CP still not competitive with state-of-the-art metaheuristics but getting somewhat close in some cases

5. Hybrid flowshops

- Companies still not using finite capacity schedulers for production programming in most cases
- Trend to address more realistic problems

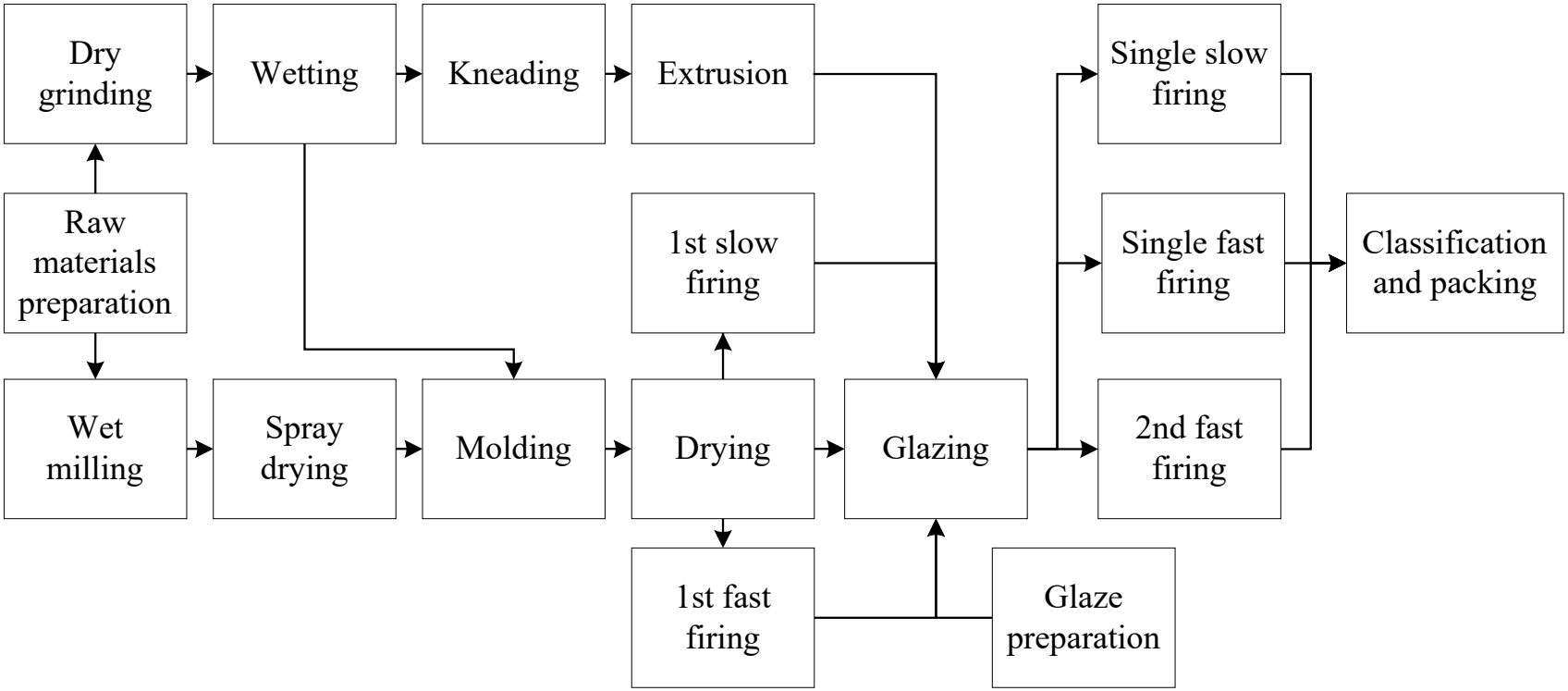
5. Hybrid flowshops

5. Hybrid flowshops

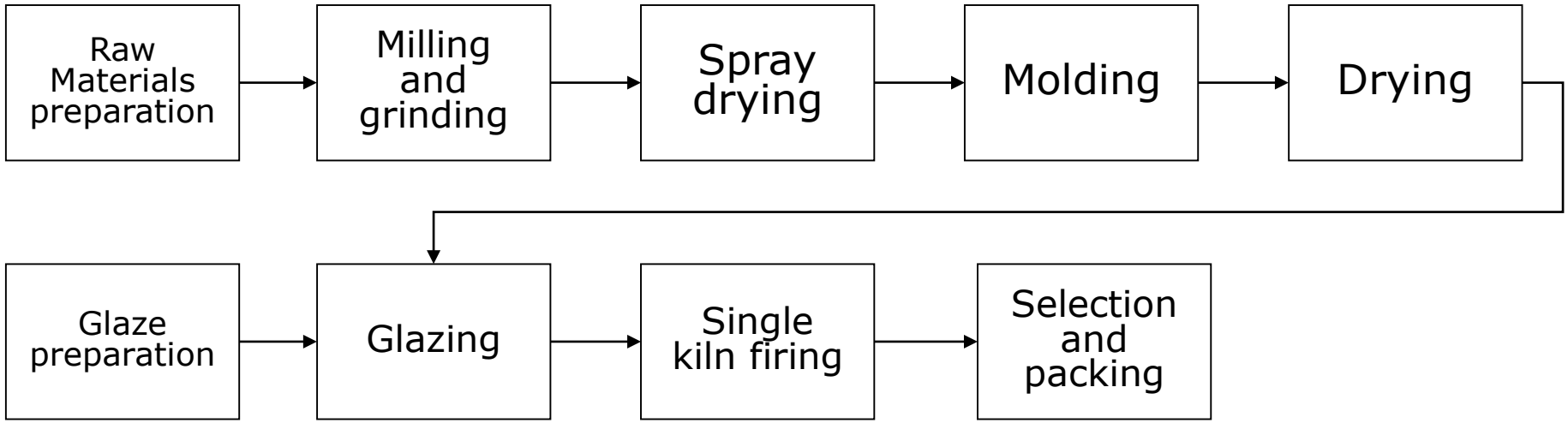
63. / Ma

5. Hybrid flowshops

5. Hybrid flowshops



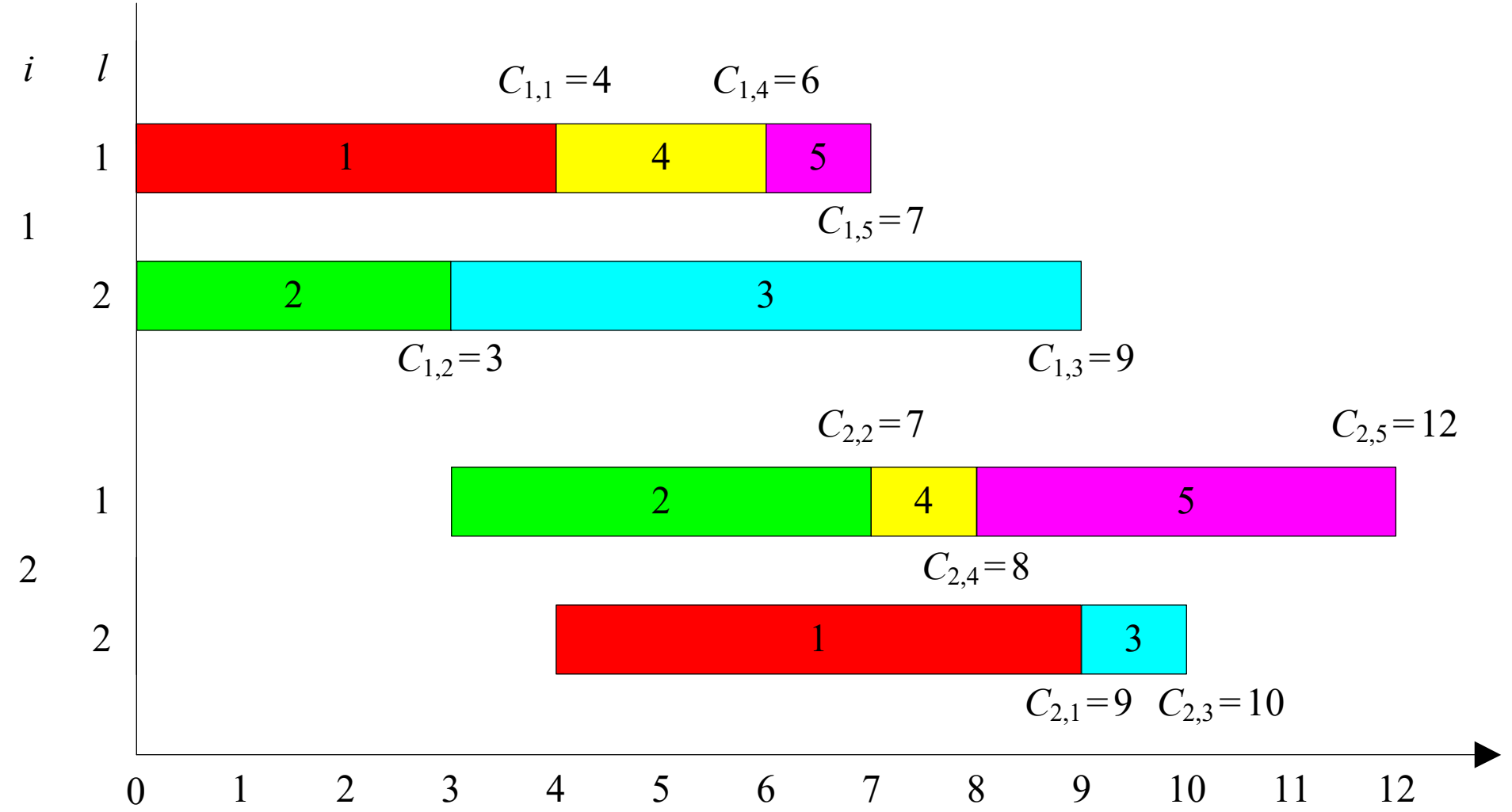
5. Hybrid flowshops



5. Hybrid flowshops

- We deal with hybrid flexible flowshop problems:
  - A set N of jobs $N=\{1,...,n\}$ that have to be processed on a set M of stages $M=\{1,...,m\}$
  - At each stage, we have a set $M_i=\{1,...,m_i\}$ of m_i unrelated parallel machines
  - All jobs are processed sequentially on the stages
  - p_{ilj} is the processing time of job j at machine l inside stage i

5. Hybrid flowshops



5. Hybrid flowshops

- We consider many realistic constraints:
  - F_j denotes the set of stages that job j visits
  - E_{ij} is the set of machines that can process job j at stage i
  - rm_{il} represents the release date of machine l inside stage i. No job can start processing on this machine before this release date
  - P_j is the set of jobs that should precede job j. The first operation of job j should wait until all last operations of jobs in P_j have finished

5. Hybrid flowshops

- S_{iljk} is the machine based setup times on machine l at stage i when processing job k after having processed job j.
- Setups might be either anticipatory or non-anticipatory.
  - A_{iljk}=0 -> the setup is non-anticipatory, and one must wait for job j to arrive at machine l before carrying out the setup.
  - A_{iljk}=1 -> the setup can be done as soon as machine l is free

5. Hybrid flowshops

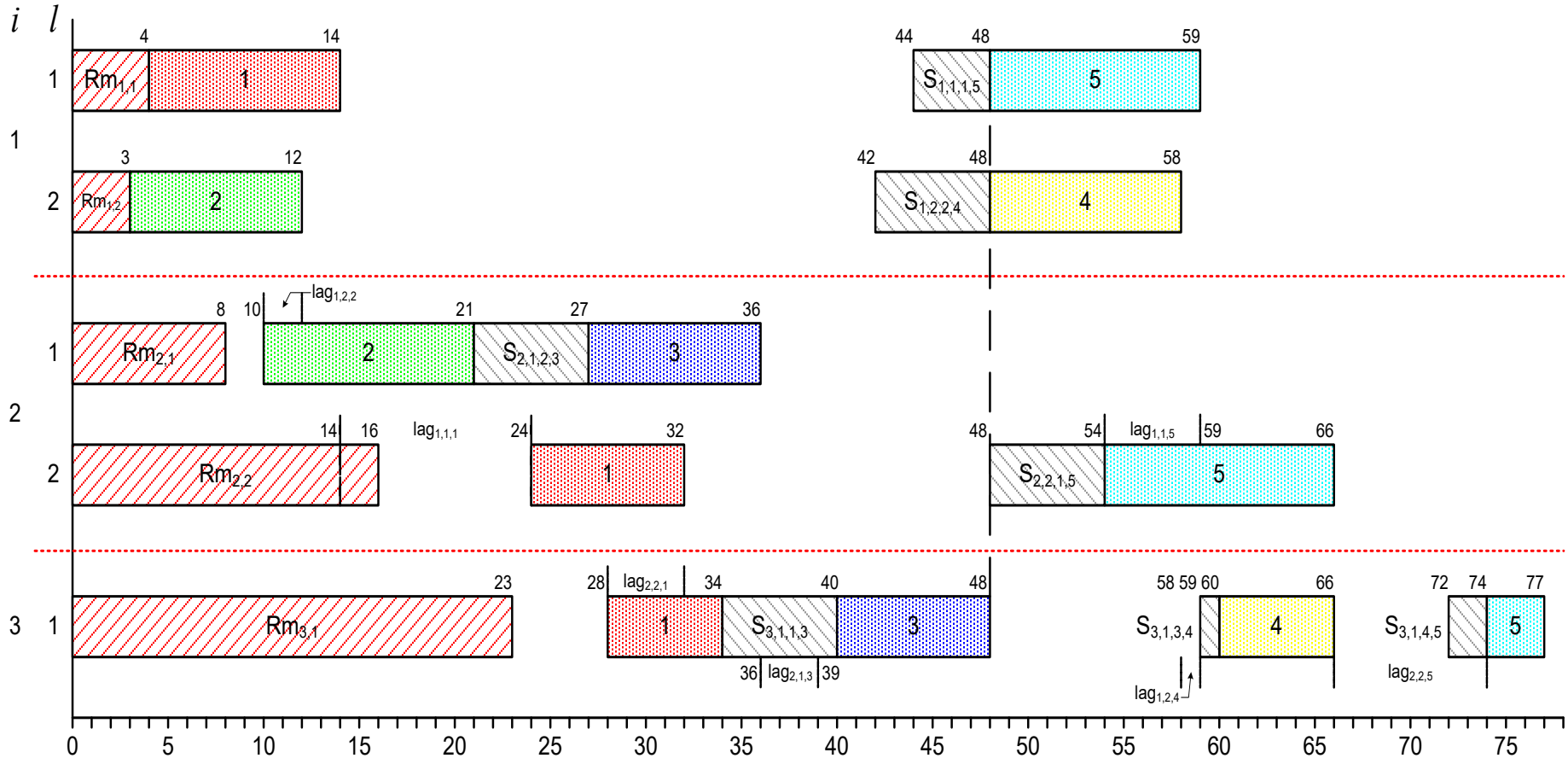
- lag_{ilj} is the time lag between the end of job j at machine l inside stage i and the beginning of the next stage in which job j is processed.
  - If $lag_{ilj}<0$ then overlapping occurs (not all parts in job j need to be finished at stage i before processing at the next stage starts)
  - If $lag_{ilj}>0$ then we have a waiting time between stages

5. Hybrid flowshops

- We will consider the common makespan criterion initially
- The problem is NP-Hard since many simplifications of it are NP-Hard
- Following the common three field notation and Vignier et al. (1999), the problem can be denoted as:

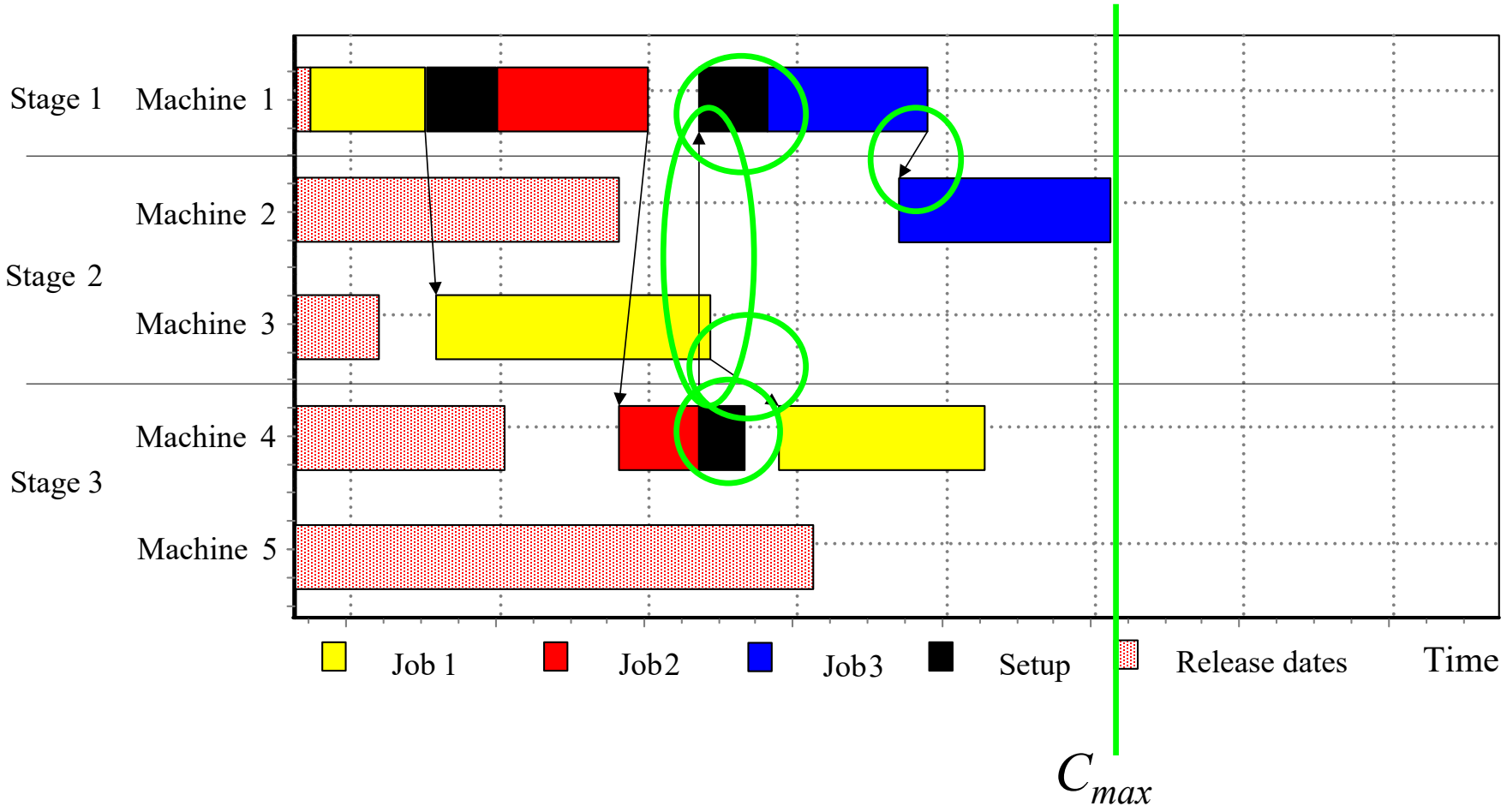
$$HFFLm,((RM^{(i)})_{i=1}^{(m)})/rm,lag,S_{ijk},M_j,prec/C_{\max}$$

5. Hybrid flowshops

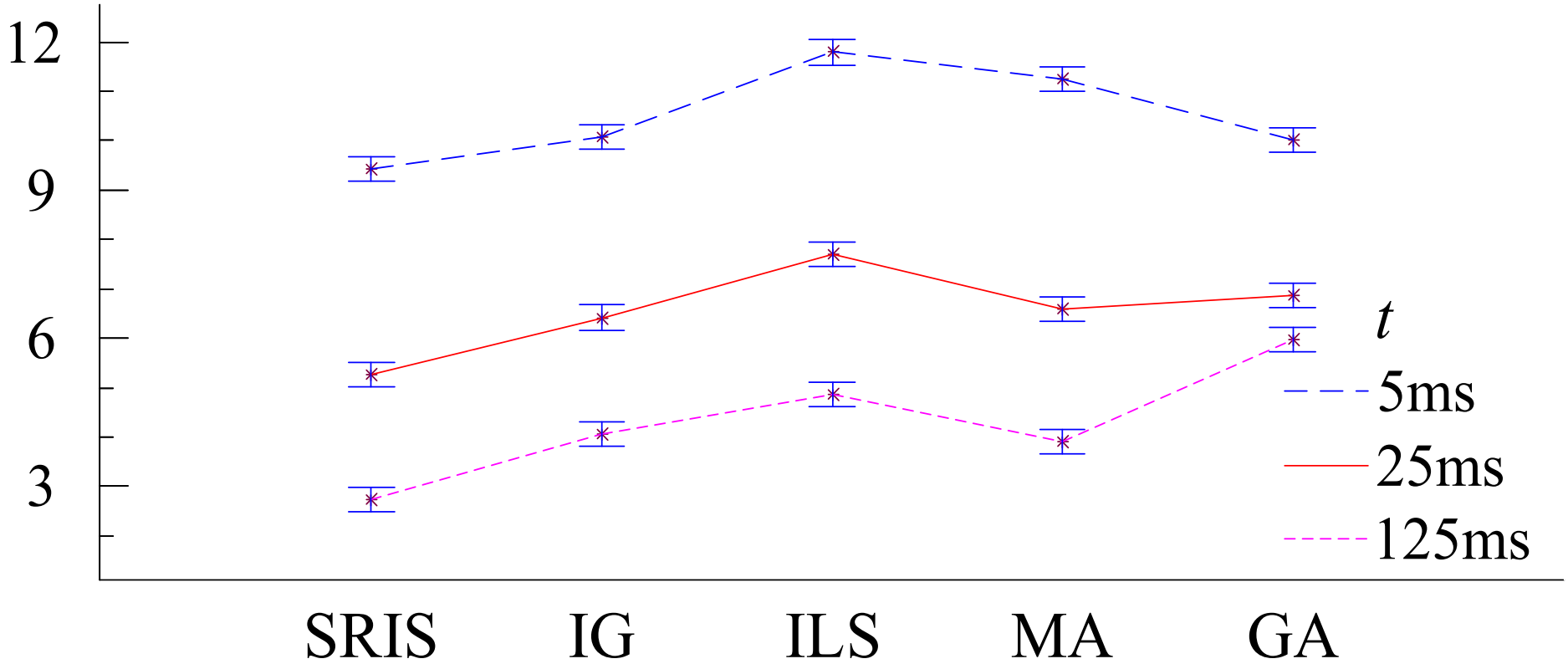


$$C_{\max} = 77$$

5. Hybrid flowshops



5. Hybrid flowshops

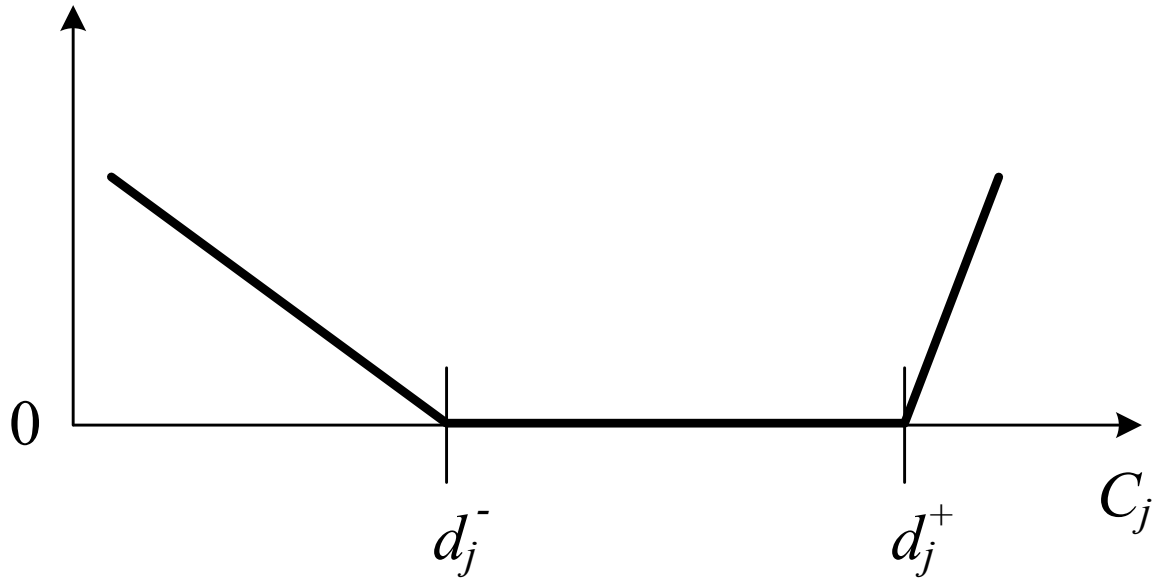


5. Hybrid flowshops

- We extend the problem with the consideration of due dates
- C_j is the completion time of job j at the shop
- d_j is the due date of job j
- w_j its importance or priority
- $T_j=\max\{0, C_j - d_j\}$ is the tardiness of job j
- $E_j=\max\{0, d_j - C_j\}$ is the earliness of job j

5. Hybrid flowshops

- Extension of the due date concept to due date windows



- Considered objective function:

$$T_j^{dw} = \max\{C_j - d_j^+, 0\}$$

$$E_j^{dw} = \max\{d_j^- - C_j, 0\}$$

$$TWET^{dw} = \sum_{j=1}^{n} \left(w'_j E_j^{dw} + w_j T_j^{dw}\right)$$

5. Hybrid flowshops

- Literature on hybrid flowshop very rich
- This objective has not been studied yet
- Objectives of the research:
  - To propose simple methods
  - To compare them against the best adapted algorithms from the literature
  - State-of-the-art results?

5. Hybrid flowshops

- We propose an Iterated Local search procedure
- Very simple:

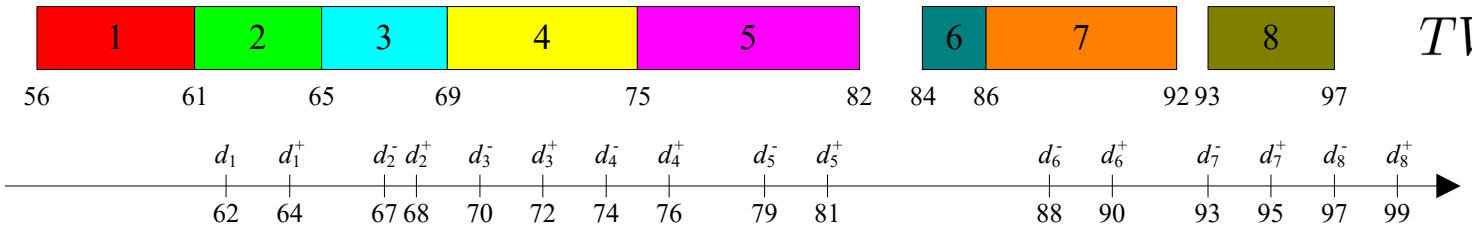
procedure ILS
  $\pi_0 :=$ GenerateInitialSolution
  $\pi :=$ LocalSearch(π_0)
  while (termination criterion not satisfied) **do**
    $\pi' :=$ Perturbation(π)
    $\pi'' :=$ LocalSearch(π')
    $\pi :=$ AcceptanceCriterion(π'', π)
  endwhile
end

5. Hybrid flowshops

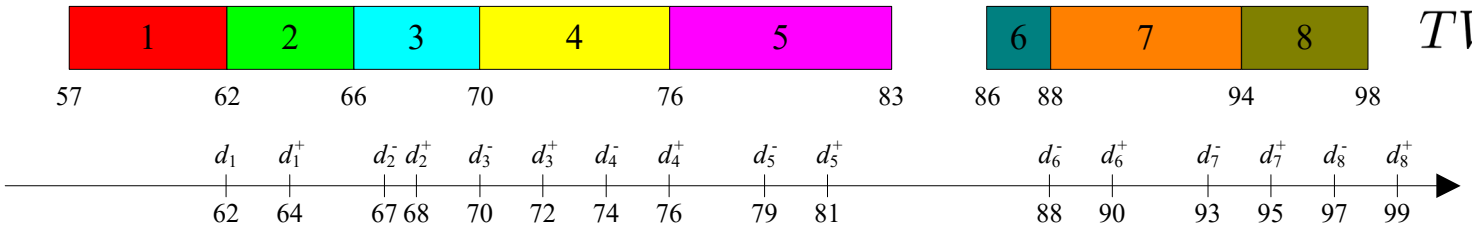
- Solution representation is a complex issue in HFS
- A simple permutation not enough
- Complex representations too time consuming
- Simple permutation with the FAM rule
- For stage 2,…m, jobs are sorted according to their completion times in the previous stage

5. Hybrid flowshops

- Active schedules not enough to optimize earliness
- We apply an idle time insertion method to all jobs in the last machine



$TWET^{dw} = 13$



$TWET^{dw} = 6$

5. Hybrid flowshops

- We apply a two stage local search
- The first one works with the permutation (insertion and interchange inside VND)
- A second local search is carried out over the exact representation
- But only a few promising neighbors are explored

5. Hybrid flowshops

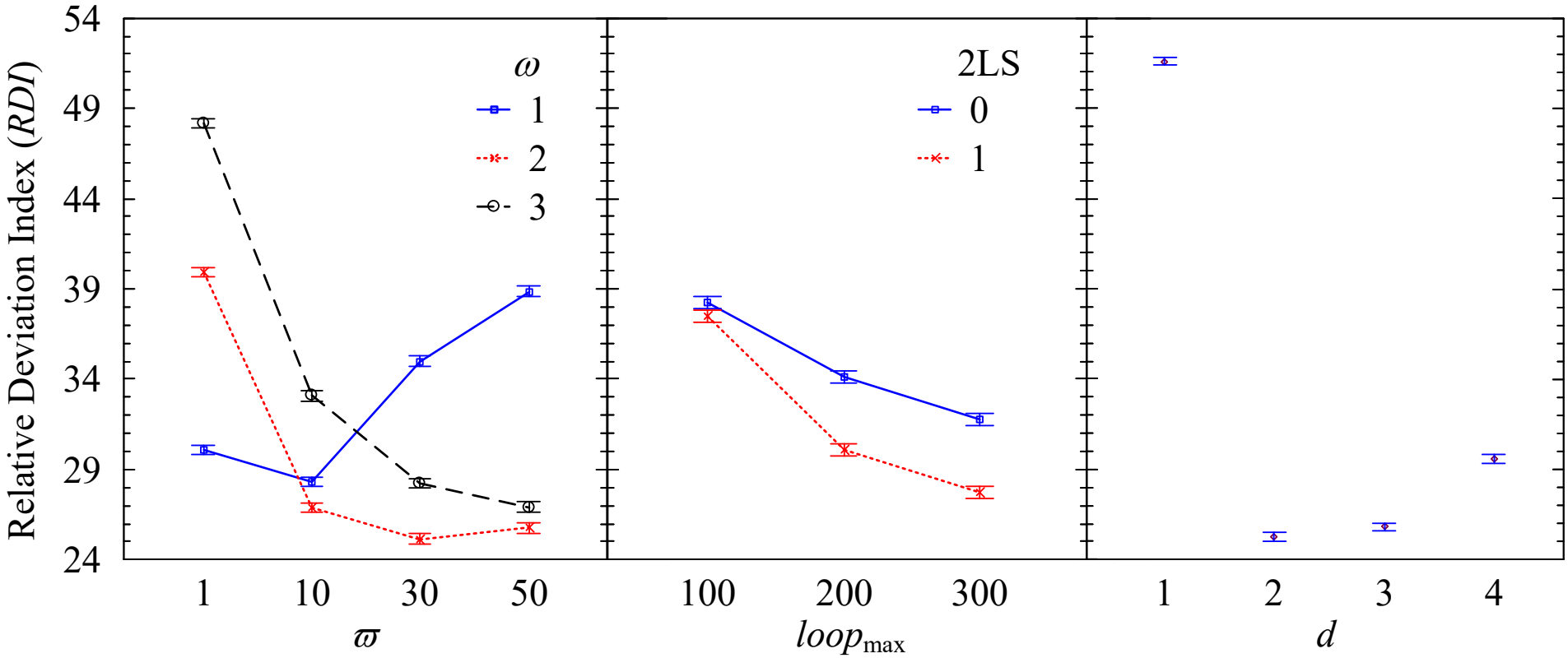
- Simple perturbation by carrying out a number of insertion and interchanges
- Directed perturbation. A number of perturbed solutions is generated and the best is obtained
- Tournament acceptance criterion
  - List with the best found solutions
  - If a better solution is found the list is cleared
  - If solution not better, tournament select a solution from the list

5. Hybrid flowshops

- We also present an Iterated Greedy (IG) procedure based on the work of Ruiz and Stützle (2007) for the regular flowshop
- Straight adaptation where we apply the same local search in two stages, idle time insertion, tournament acceptance criterion

5. Hybrid flowshops

- Calibration by means of DOE+ANOVA



5. Hybrid flowshops

- We test the proposed ILS and IG against the 9 best existing methods from the literature:

1. The Artificial Immune System of Engin and Döyen (2004) (AIS),
2. The GA of Ruiz and Maroto (2006) (GA_R),
3. The Ant Colony Optimization of Alaykýran et al. (2007) (ACO),
4. The GA of Kahraman et al. (2008) (GA_K),
5. The improved SA of Naderi et al. (2009) (HSA),
6. The ACO of Khalouli et al. (2010) (ACO_K),
7. The ILS of Naderi et al. (2010) (ILS_N),
8. The Discrete Colonial Competitive Algorithm of Behnamian and Zandieh (2011) (DDCA),
9. The Artificial Bee Colony of Pan et al. (2013) (ABC_P).

5. Hybrid flowshops

- All algorithms coded in C++
- Competing algorithms carefully reimplemented
- All methods calibrated with ANOVA
- All methods include the proposed idle time insertion procedure
- Runs on a cluster of 30 blade servers each one with two Intel XEON E5420 processors running at 2.5 GHz and with 16 GB of RAM memory
- 342 days of computer time for calibration
- Calibration instances different from test instances

5. Hybrid flowshops

- Two sets of instances:
  - 1620 small instances with up to 20 jobs, 3 stages and 4 machines per stage
  - 1400 large instances with up to 200 jobs, 10 stages and 10 machines per stage
- Average relative deviation index from best solution known (RDI)
- 3 different stopping times: $t = \rho \cdot n \cdot m \cdot$ milliseconds where ρ is tested at 3 levels: 30, 60 and 90

5. Hybrid flowshops

- We test ILS and IG with and without Tournament acceptance criterion (ILS, ILST, IG, IGT)
- 5 replicates per algorithm and instance
- Therefore, 13 algorithms × 3 stopping times × 5 replicates × (1620+1440) instances =596,700 results
- 193 days of computer time for tests

5. Hybrid flowshops

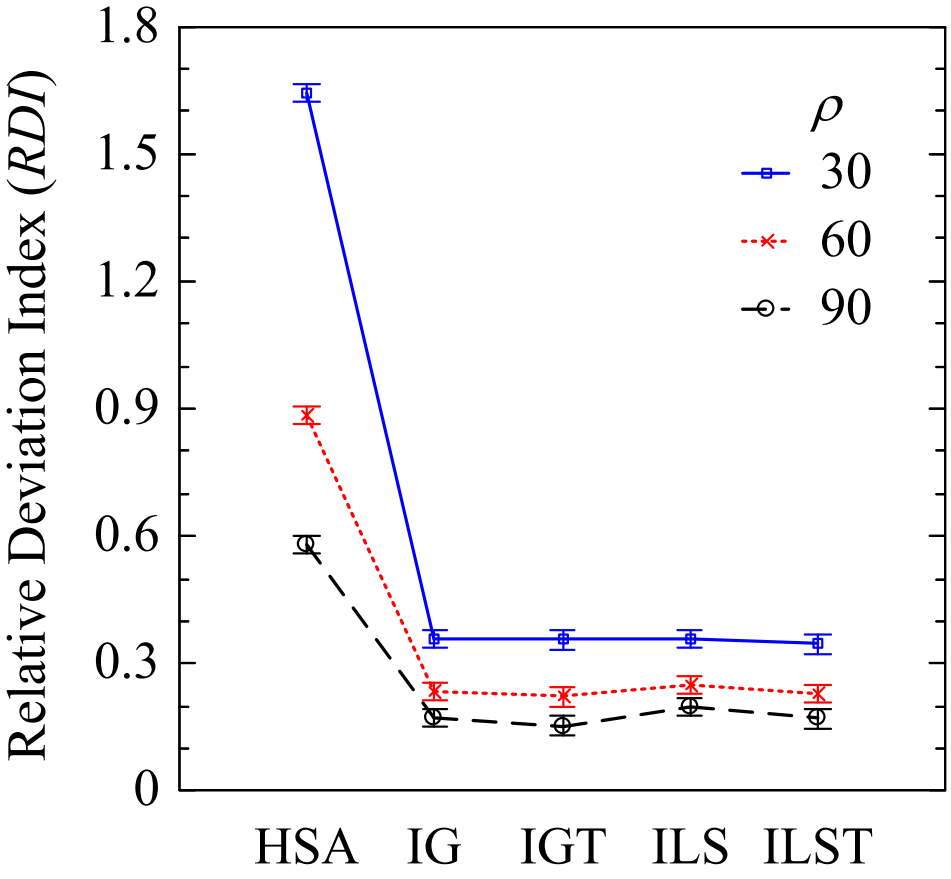
ρ	ABC_P	ACO	ACO_K	AIS	DCCA	GA_K	GA_R	HSA	ILS_N	IG	IGT	ILS	ILST
30	1.71	69.98	66.26	4.2	5.47	2.96	2.35	1.09	1.02	**0.45**	0.46	**0.45**	0.44
60	1.47	69.3	60.99	3.7	4.91	2.72	2.04	0.96	0.94	**0.33**	0.35	0.34	0.34
90	1.37	68.8	58.17	3.4	4.81	2.59	1.88	0.91	0.9	**0.28**	0.30	0.29	0.29
Average	1.52	69.4	61.81	3.8	5.07	2.76	2.09	0.99	~~0.96~~	**0.36**	0.37	**0.36**	**0.36**

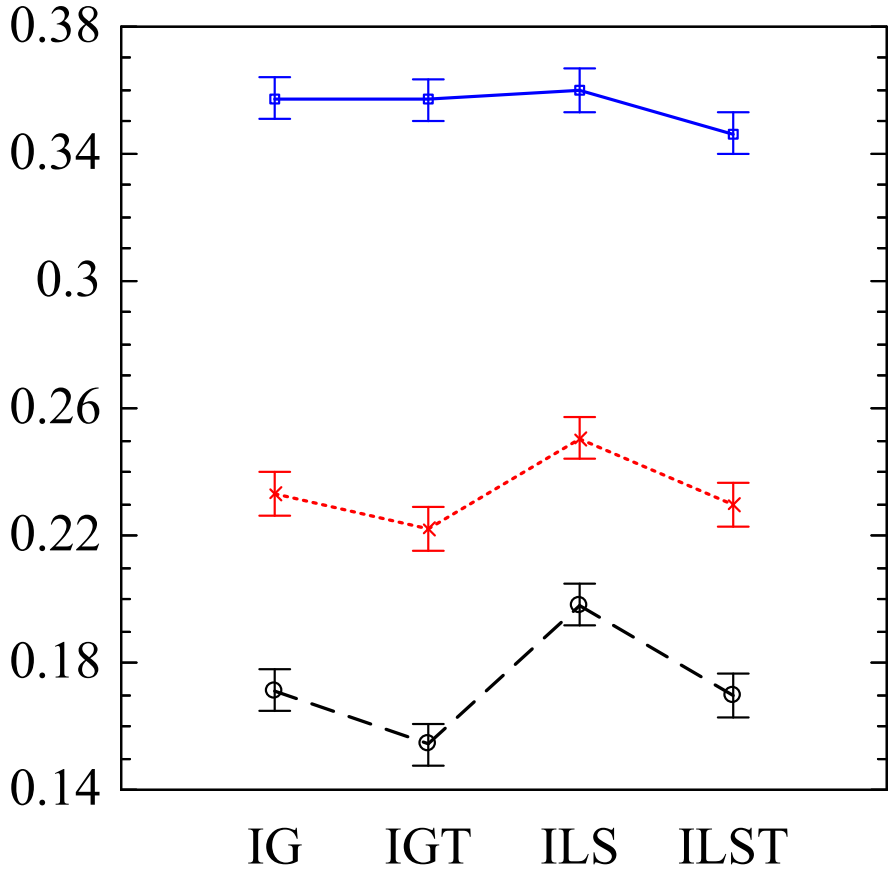
275% lower

ρ	ABC_P	ACO	ACO_K	AIS	DCCA	GA_K	GA_R	HSA		IG	IGT	ILS	ILST
30	1.70	8.18	97.27	6.42	2.58	2.47	2.07	1.64	1.77	0.36	**0.35**	0.36	0.35
60	1.28	8.16	95.82	5.97	2.49	2.08	1.48	0.89	0.99	0.23	**0.22**	0.25	0.23
90	1.07	8.15	94.99	5.72	2.40	1.92	1.21	0.58		7	**0.15**	0.20	0.17
Average	1.35	8.17	96.03	6.04	2.49	2.16	1.58	1.04	~~1.14~~	~~0.25~~	**0.24**	0.27	0.25

433% lower

5. Hybrid flowshops





5. Hybrid flowshops

- We have studied the hybrid flowshop problem with a generalized objective function which minimizes the earliness and tardiness from a due date window
- We have presented simple local search methods
- Two stage local search with a limited local search in the exact representation key to results

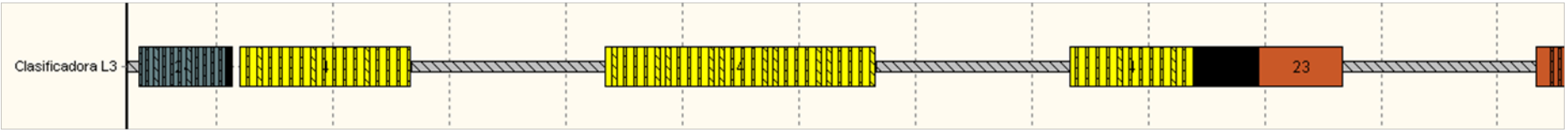
5. Hybrid flowshops

- Comprehensive comparison against 9 adapted and calibrated methods
- Results are clearly state-of-the-art in all tested scenarios under small and large instances
- Future work:
  - More realistic considerations like non-identical machines per stage or sequence-dependent setup times

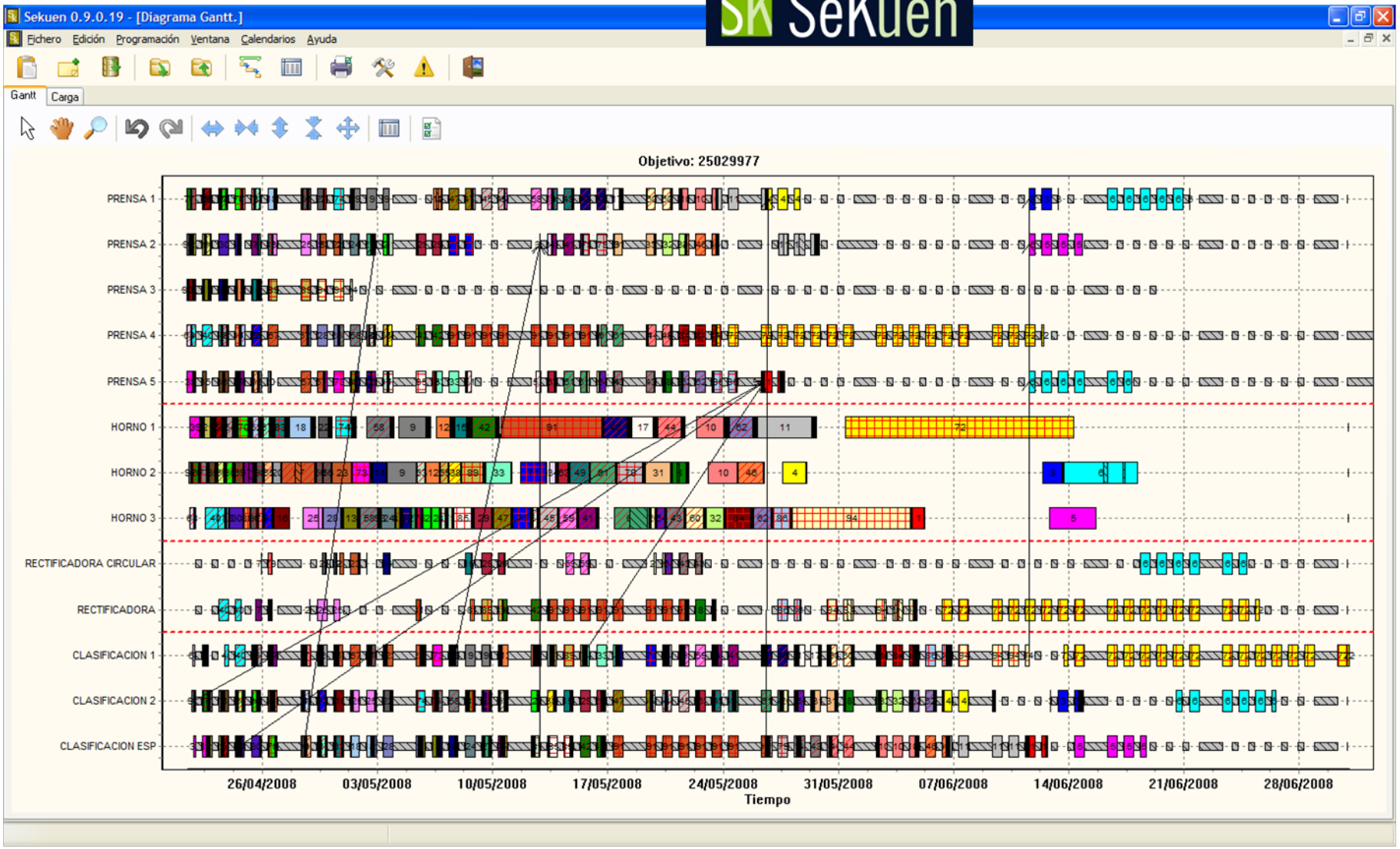
5. Hybrid flowshops

- Although we have considered a highly realistic problem, there are still a number of assumptions:
  - No errors in the processing of jobs
  - No breakdowns and continuous machine availability
  - Unlimited buffer capacity in-between stages
  - Not considered: Recirculation, preemption, no wait …
  - Heuristics / metaheuristics need to be developed

5. Hybrid flowshops



5. Hybrid flowshops



6. Conclusions

- Flowshops problems have many applications in practice
- There are many variants in objectives, constraints and settings
- Simple methods usually show good performance
- Lot of work to do!

EL PROBLEMA DEL TALLER DE FLUJO (*FLOWSHOP SCHEDULING PROBLEM*)

Rubén Ruiz

INSTITUTO TECNOLÓGICO DE INFORMÁTICA. GRUPO DE SISTEMAS DE OPTIMIZACIÓN APLICADA.
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SOA
SISTEMAS DE OPTIMIZACIÓN APLICADA