Algoritmos Iterados Golosos: fundamentos, aplicaciones y resultados

Rubén Ruiz

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SISTEMAS DE OPTIMIZACIÓN APLICADA

Outline

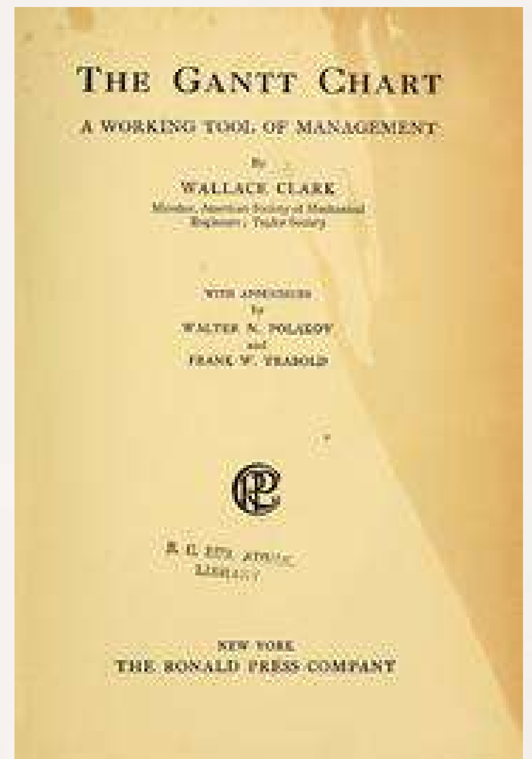
1. Introduction
2. The flowshop problem
3. Basic IG algorithm
4. Results for other flowshop problems
5. Parallel machines
6. Complex hybrid problems
7. Multiobjective
8. Distributed scheduling
9. Conclusions

1. Introduction

”Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives”

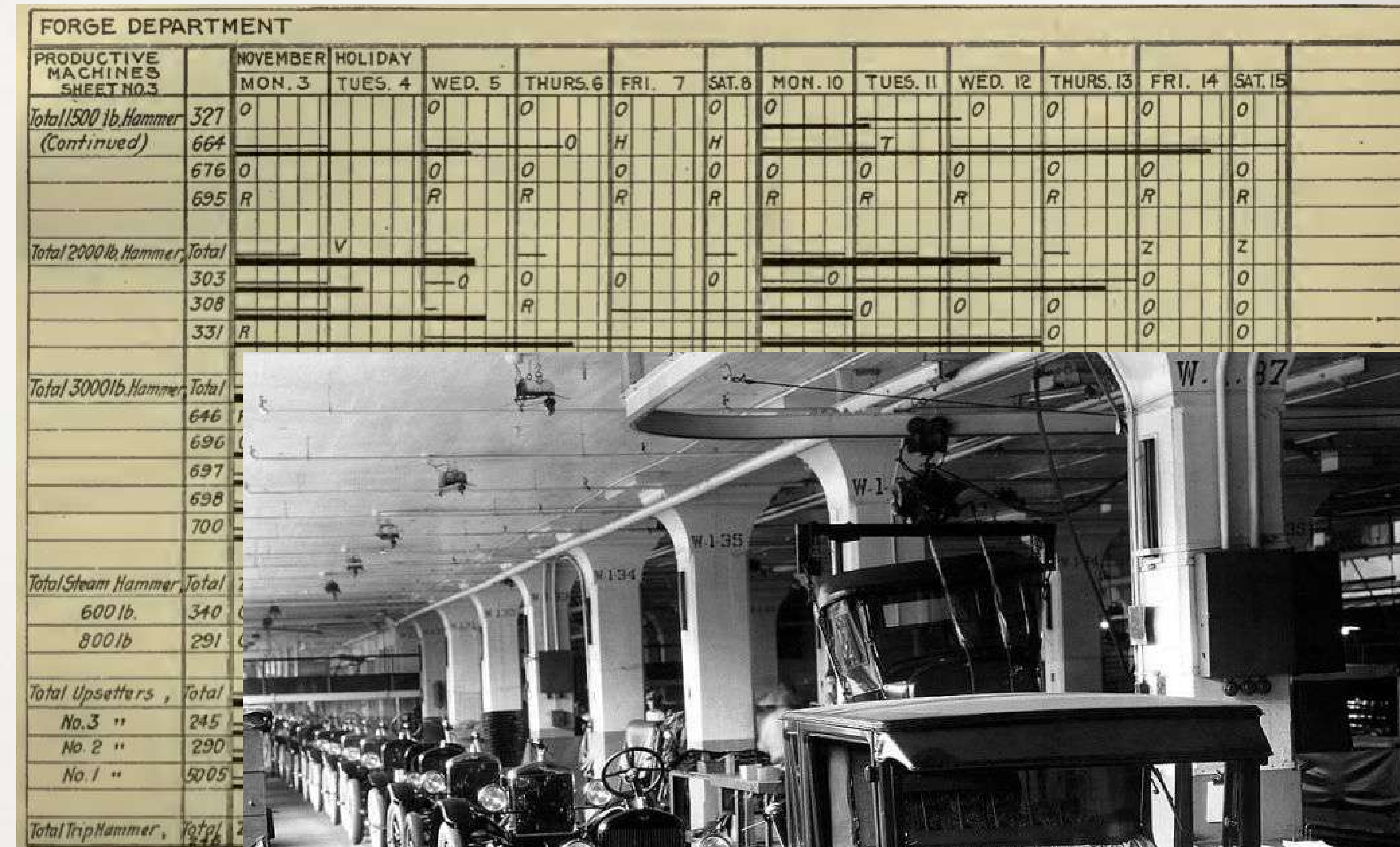
Scheduling. Theory, Algorithms and Systems. Michael Pinedo. Springer (2016). Fifth Edition

Introduction

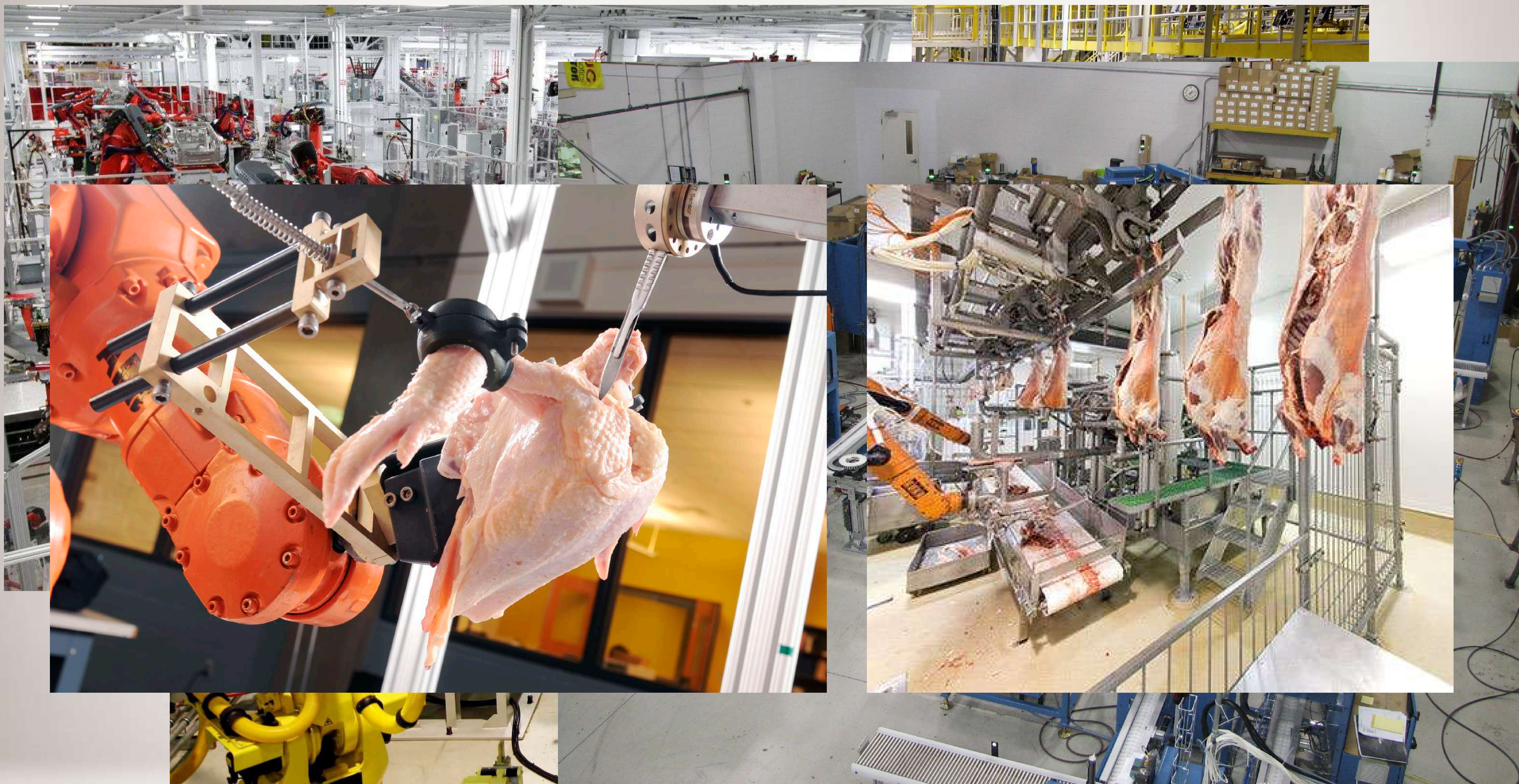


Henry Lawrence Gantt

(1861-1919)



Introduction



Introduction

Scheduling today is notoriously difficult and complicated

Production processes vary a lot from industry to industry:

- Not the same producing an LCD panel
- Than a ceramic tile
- Ad-hoc specific algorithms for each process/product is not a viable approach, as we would need thousands of different algorithms with huge maintenance costs

Introduction

We need general optimization methods

Context independent

Flexible

But at the same time powerful

Optimality is a panacea for real complex problems

We have to resort to heuristics

Introduction

Metaheuristics

”…higher level procedure or heuristic designed to find, generate, or select a lower-level procedure or heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem…”

(wikipedia)

Introduction

Metaheuristics is a very prolific field

1. Genetic algorithms (Holland, 1975)
2. Simulated Annealing (Kirkpatrick et al., 1983)
3. Tabu Search (Glover, 1986)
4. GRASP (Feo and Resende, 1989)
5. Ant Colony Optimization (Dorigo, 1992)
6. Iterated Local Search (Stützle, 1998)
7. Particle Swarm Optimization (Kennedy, 1995)
8. VNS (Hansen and Mladenović, 1999)

Introduction

Maybe a bit too prolific

9. Artificial Immune Systems (Forrest et al., 1994)
10. Self-Propelled Particles (Vicsek et al., 1995)
11. Differential Evolution (Storm and Price, 1997)
12. Harmony Search (Zong, 2001)
13. Bee Colony Optimization (Karaboga, 2005)
14. Firefly Optimization (Krishnanand and Ghose, 2005)
15. Intelligent Water Drops (Shah-Hosseini, 2009)

…

Introduction

And today we have really lost our minds

Kangaroo algorithms (Fleury, 1995)

Squeaky Wheel Optimization (Joslin and Clements, 1999)

Imperialist Competitive Algorithm (Atashpaz-Gargari and Lucas, 2007)

Cuckoo Optimization (Rajabioun, 2011)

Water cycle algorithms (Eskandar, 2012)

Mine Blast optimization (Sadollah, 2013)

Gases Brownian Motion Optimization (Abdechiri, 2013)

Leapfrog optimization, bats, flies, galaxies, roots, … whatever

Introduction

Some wild ideas received as editor/referee (extract!)

Lion algorithm

Flower pollination

Lizards

Grey Wolf optimization

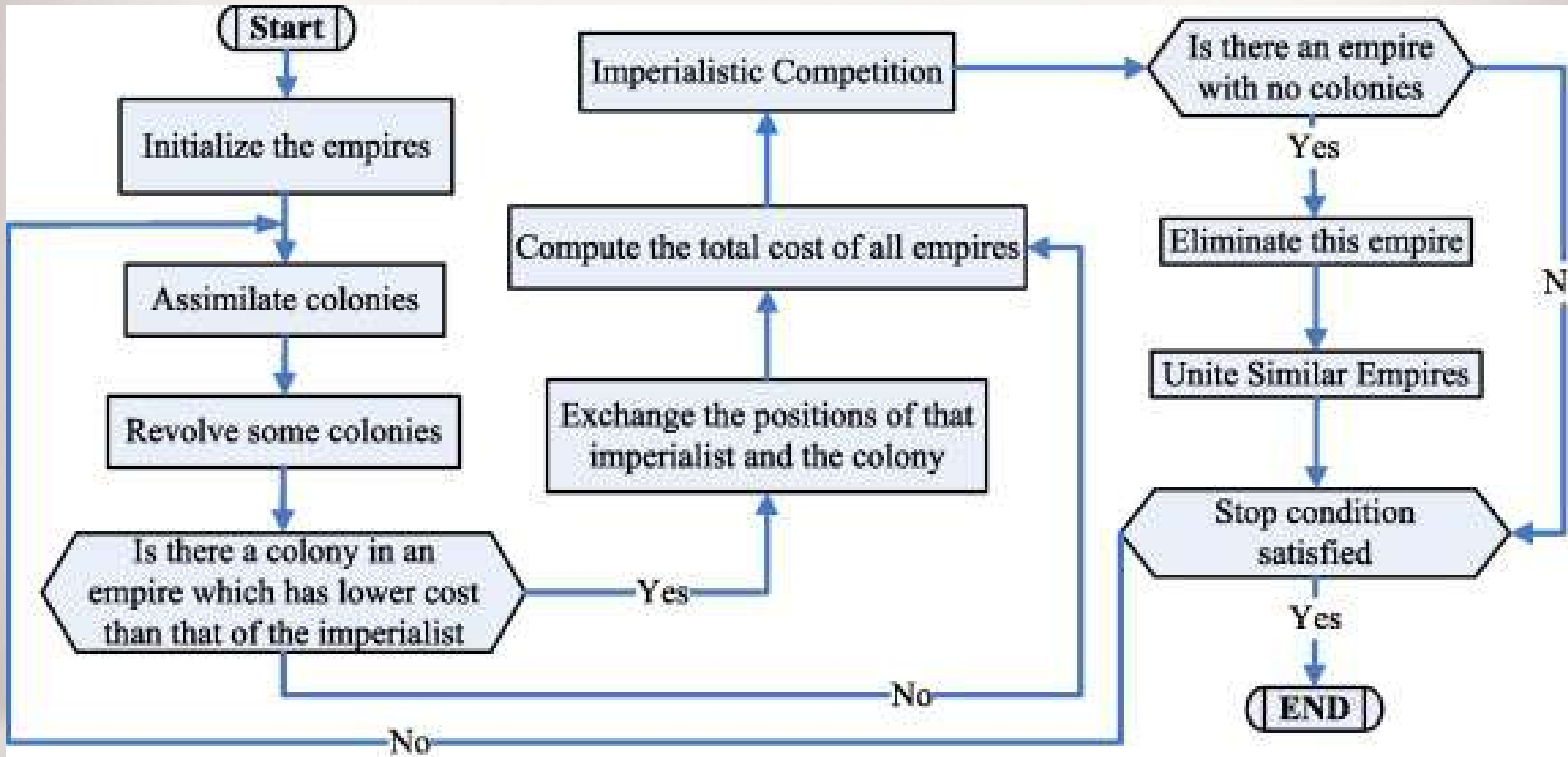
Electromagnetic Field Optimization

Cuckoo - GA Cannibalism Based Hybrid Evolutionary Algorithm

…and all kinds of very, very weird hybrids…

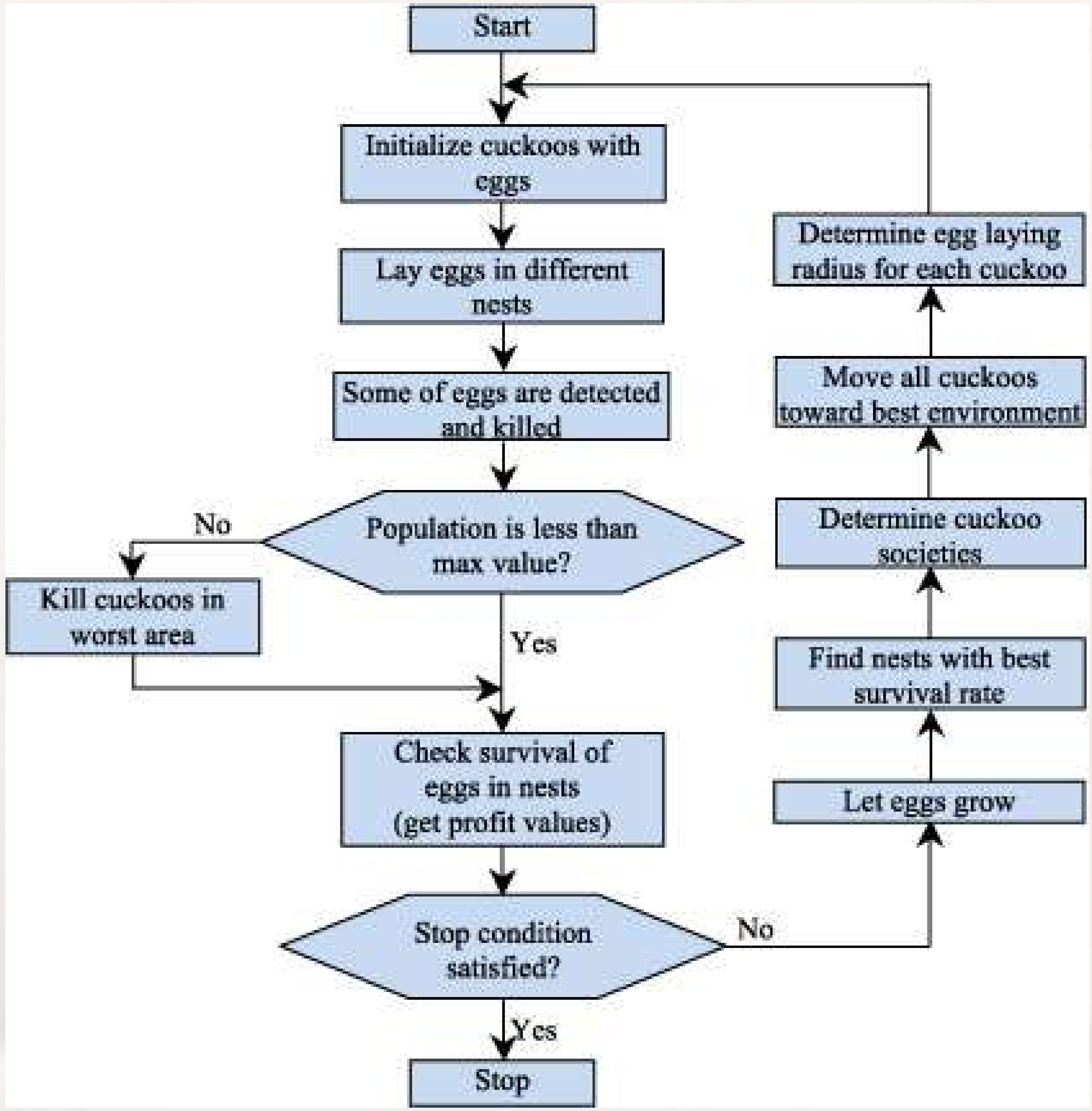
Introduction

An example: The Imperialist Competitive Algorithm



Introduction

Another example: Cuckoo Optimization



Introduction

Submission to IFORS 2014 Barcelona

The Performance of the Flying Elephants Approach for Solving Traditional Non-Differentiable Problems

Contributed abstract in session Unsorted, stream 13. Metaheuristics and Matheuristics (contributed).

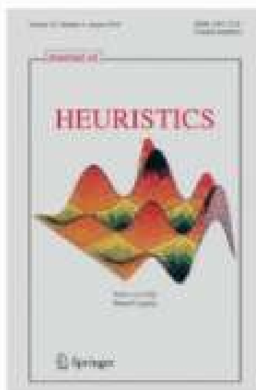
Area: 13. Metaheuristics and Matheuristics

Abstract

Flying Elephants is a generalization and a new interpretation of the Hyperbolic Smoothing approach. The name is definitely not associated to any analogy with the biology area. It is only a metaphor. The Flying feature is directly derived from its differentiability property, **which permits intergalactic trips of the Elephant into spaces with large number of dimensions**, differently of the short local searches associated to traditional heuristics. Computational results for solving distance geometry, covering, clustering, Fermat-Weber and hub location problems show the performance of the approach.

Introduction

Springer Link   Search



Journal of Heuristics
August 2016, Volume 22, Issue 4, pp 649–664

Flying elephants: a general method for solving non-differentiable problems

Authors | Authors and affiliations

Adilson Elias Xavier, Vinicius Layter Xavier

Article
First Online: 09 November 2014

1 Citations | 358 Downloads

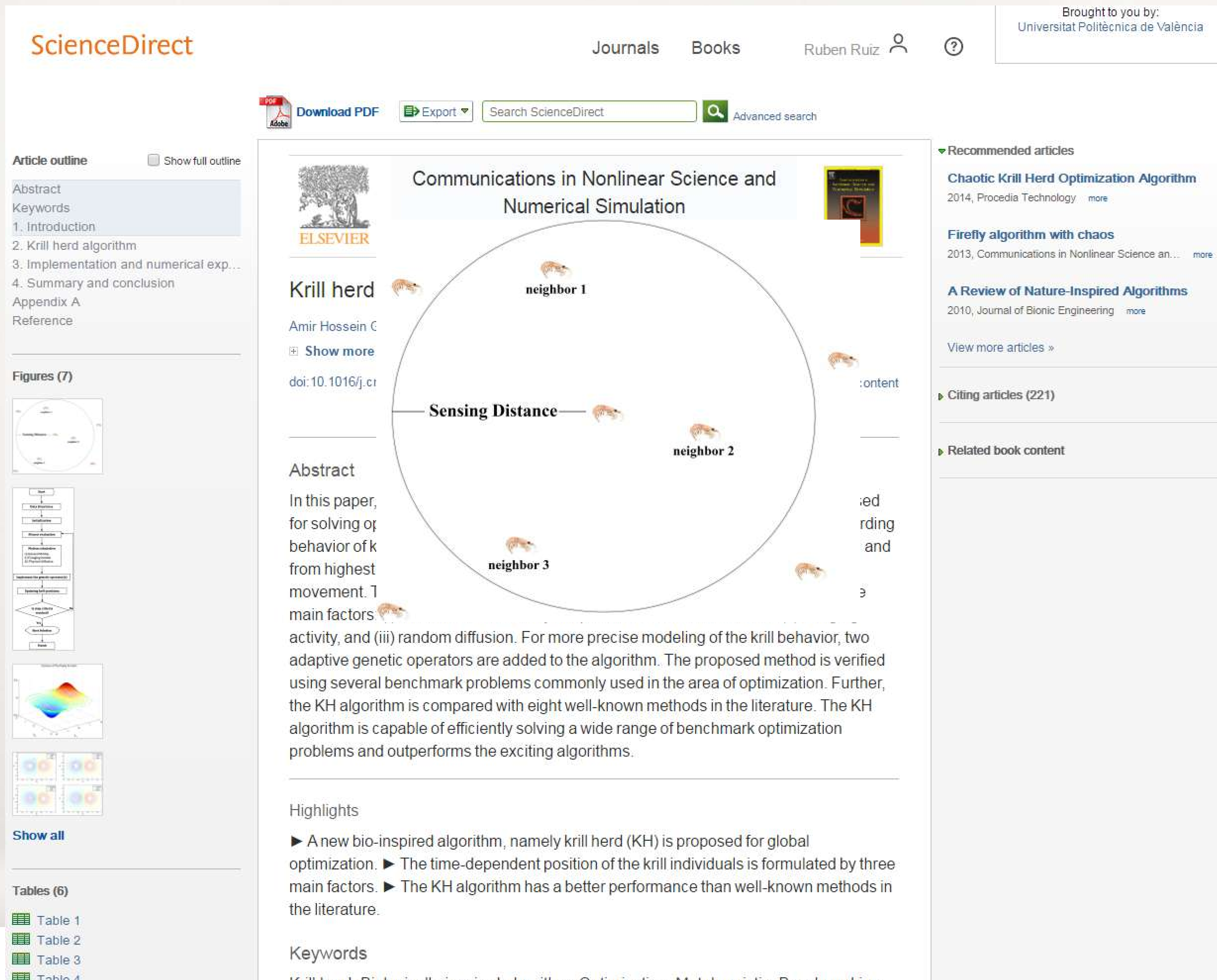
Abstract

Flying Elephants (FE) is a generalization and a new interpretation of the Hyperbolic Smoothing approach. The article introduces the fundamental smoothing procedures. It contains a general overview of successful applications of the approach for solving a select set of five important problems, namely: distance geometry, covering, clustering, Fermat–Weber and hub location. For each problem the original non-smooth formulation and the succedaneous completely differentiable one are presented. Computational experiments for all related problems obtained results that exhibited a high level of performance according to all criteria: consistency,

Even the editor had to apologize!

Introduction

Not the only one by far:



Introduction

We have put together a bestiary

Claus Aranha @ Tsukuba University

Home Profile Research Education Students Blog

EC Bestiary

Updated May 3rd, 2017

> "Till now, madness has been thought a small island in an ocean of sanity. I am beginning to suspect that it is not an island at all but a continent." -- Machado de Assis, *The Psychiatrist*.

Introduction

The field of meta-heuristic search algorithms has a long history of finding inspiration in natural systems. Starting from classics such as Genetic Algorithms and Ant Colony Optimization, the last two decades have witnessed a fireworks-style explosion (pun intended) of natural (and sometimes supernatural) heuristics - from Birds and Bees to Zombies and Reincarnation.

The goal of the Evolutionary Computation Bestiary is to catalog the, ermm... exuberance of the meta-heuristic "eco-system". We try to keep a list of the many different animals, plants, microbes, natural phenomena and supernatural activities that can be spotted in the wild lands of the metaphor-based computation literature.

While we personally believe that the literature could do with more mathematics and less marsupials, and that we,

Gotta catch 'em all!

BioHeuristics GO

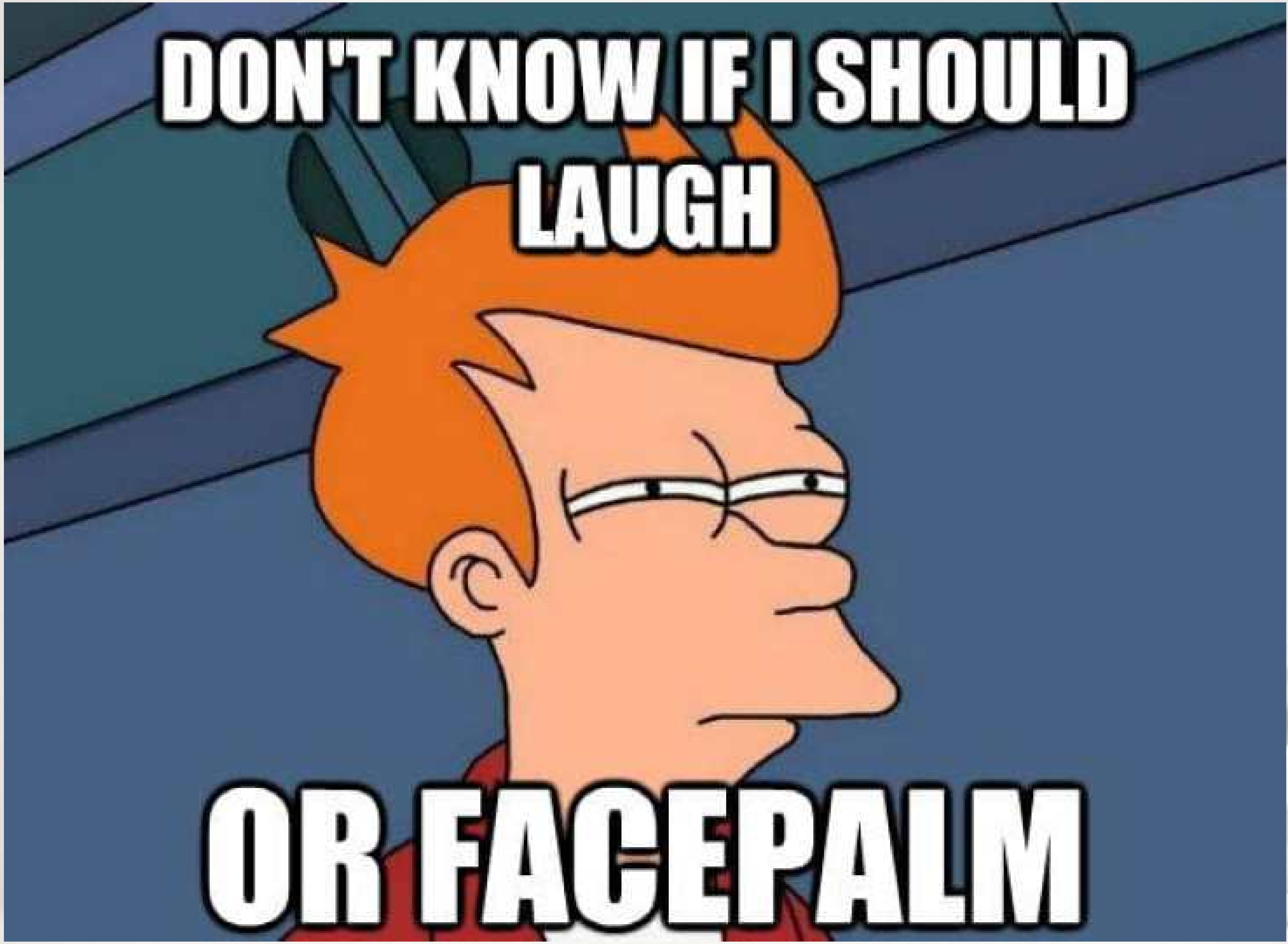
http://conclave.cs.tsukuba.ac.jp/research/bestiary/

Introduction

Many wild ideas (>130 “unique” entries):

- Lion algorithm
- Flower pollination
- Lizards
- Grey Wolf optimization
- Electromagnetic Field Optimization
- Reincarnation
- Sperm Motility Optimization
- Even Zombies!!!!

Introduction



Introduction

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 22 (2015) 3–18
DOI: 10.1111/itor.12001

Metaheuristics—the metaphor exposed

Kenneth Sörensen

University of Antwerp, Operations Research Group ANT/OR, Prinsstraat 13 – B5xx, 2000 Antwerp, Belgium
E-mail: kenneth.sorensen@ua.ac.be [Sörensen]

Received 2 November 2012; received in revised form 12 November 2012; accepted 13 November 2012

Abstract

In recent years, the field of combinatorial optimization has witnessed a true tsunami of "novel" metaheuristic methods, most of them based on a metaphor of some natural or man-made process. The behavior of virtually any species of insects, the flow of water, musicians playing together – it seems that no idea is too far-fetched to serve as inspiration to launch yet another metaheuristic. In this paper, we will argue that this line of research is threatening to lead the area of metaheuristics away from scientific rigor. We will examine the historical context that gave rise to the increasing use of metaphors as inspiration and justification for the development of new methods, discuss the reasons for the vulnerability of the metaheuristics field to this line of research, and point out its fallacies. At the same time, truly innovative research of high quality is being performed as well. We conclude the paper by discussing some of the properties of this research and by pointing out some of the most promising research avenues for the field of metaheuristics.

Introduction

This hasn't gone unnoticed

Sörensen, K. (2015). Metaheuristics—the metaphor exposed, *International Transactions in Operational Research* 22(1): 3-18.

Abstract:

"In recent years, the field of combinatorial optimization has witnessed a true tsunami of "novel" metaheuristic methods, most of them based on a metaphor of some natural or man-made process. The behavior of virtually any species of insects, the flow of water, musicians playing together – it seems that no idea is too far-fetched to serve as inspiration to launch yet another metaheuristic. In this paper, we will argue that this line of research is threatening to lead the area of metaheuristics away from scientific rigor. We will examine the historical context that gave rise to the increasing use of metaphors as inspiration and justification for the development of new methods, discuss the reasons for the vulnerability of the metaheuristics field to this line of research, and point out its fallacies"

Introduction

Many of these bizarre methods get cited a lot

1. Being the first to apply the method X to the problem Y
2. Easy to improve the basic method X by adding operators or hybridizing
3. In a little while the method X gets many citations and then everybody thinks that it is good because of that
4. Easy to publish: There are more than 50,000 scientific journals in the world and more than 50 million papers published throughout history

Introduction

Zong Woo Geem, Joong Hoon Kim, and G. V. Loganathan. "A new heuristic optimization algorithm: harmony search." Simulation 76(2):60-68, 2001. 4555 citations in Google Scholar at 16:23, 12th of September, 2019

Introduction

Peas-to-Melons comparisons

Focusing only on solution quality, not considering (to some extent) CPU time

Metaheuristics use resources (CPU time, memory) to give a solution

Not carefully controlling CPU time in the comparisons leads to fallacies that are misleading (part) of the scientific community

Introduction

Comparisons often against published tables with results obtained years ago:

- Different processors (older)
- Memory speed, bus speed (older)
- Different compilers (older)
- Different programming languages
- Different operating systems
- Different coding skills
- Different stopping criteria

These factors add-up!

Introduction

Corrections based on raw CPU frequency are utterly wrong

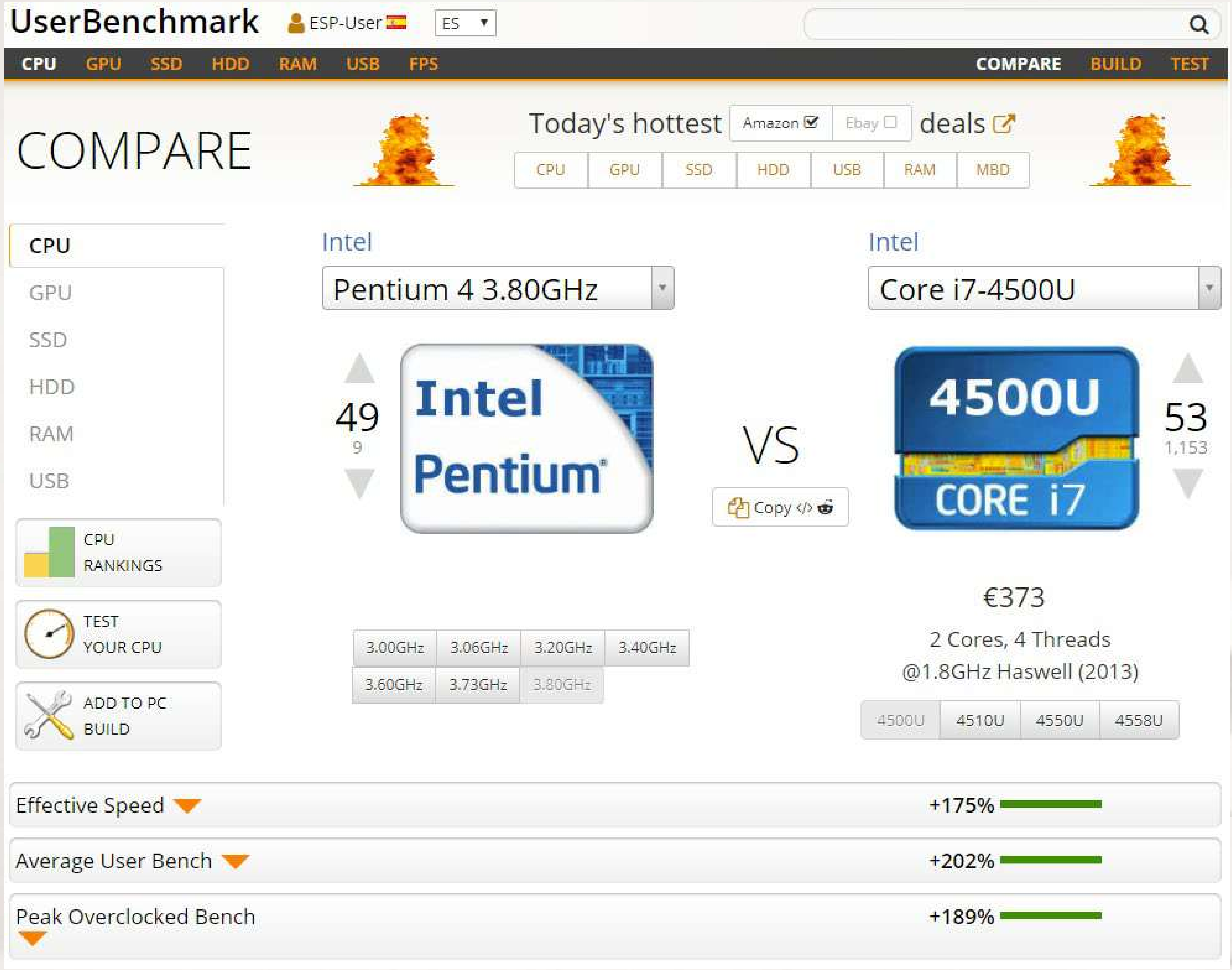
Intel Pentium 4 570 3.8 GHz (circa 2004)

Intel Core i7 4500U 1.8 GHz (circa 2013)

Older model more than twice the clock speed

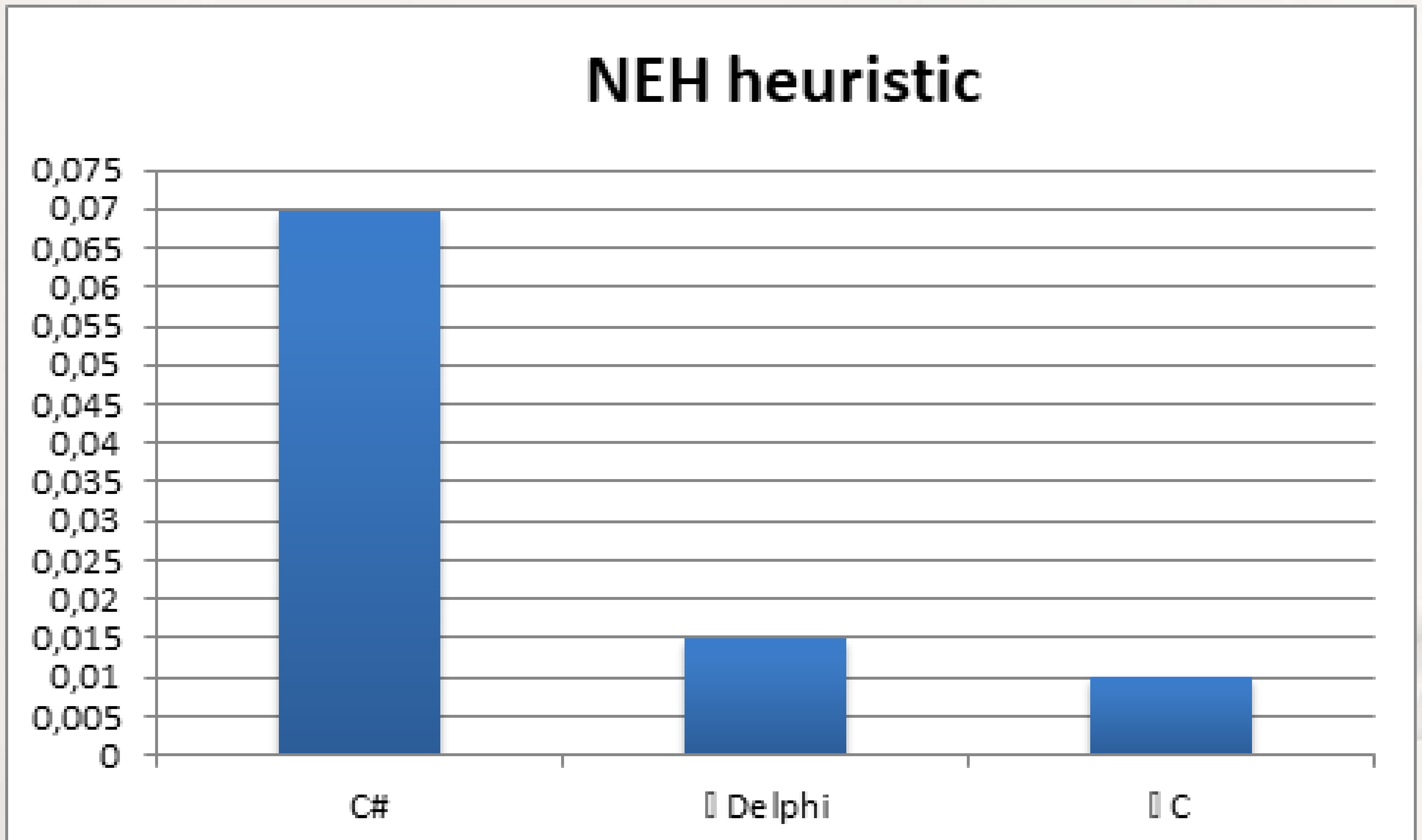
According to cpu.userbenchmark.com the new model is TWICE as fast with HALF the clock speed

Introduction



Introduction

Is the compiler/language so important?



7x speed up from C# to C

Introduction

From Visual Studio 2013 to Visual Studio 2015 you get a 20% improvement in C# binary speed due to new compiler technology "Roslyn"

Inlining/optimizing a frequently called function can improve code speed by two % digits

How can we trust a 7% improvement in solution quality in a "new" method in a Peas-To-Melons comparison?

Introduction

- Apples-to-apples comparisons:
  - REIMPLEMENT published algorithms
    - In the same language
    - Sharing most functions
    - Same coding skills
  - TEST in the same computer platform
    - Same processor, speed, architecture
    - Same compiler
    - Same OS
  - Run with used thread CPU-time as stopping criterion
  - Carry out statistical testing for significance

Introduction

What authors are doing as a result of the Peas-to-Melons comparisons:

"New" ideas easily best published methods in "comparable" running times

The better results of the "new" ideas are basically the compounded effect of a faster CPU, newer compilers, etc.

"Hybridized" versions of existing methods are seen as "better" just because they run on newer hardware not because they are actually better

Introduction

Do we need such complexities?

Simple methods have many advantages:

1. Easy to understand
2. Easy to code
3. Easy to transfer to industry
4. Easy to extend and adapt, less parameters, etc.

Introduction

In this talk I will defend the choice of very simple algorithms

That at the same time produce state-of-the-art results

…Without frowning metaphors

2. The flowshop problem

n jobs to schedule in m machines

Each job visits the machines in the same order

The order of the jobs is the same for all machines

$n \cdot m$ tasks to schedule

p_{ij} is the processing time of job j at machine i

Jobs are independent and available for processing at time 0.

Machines are continuously available

The flowshop problem

Objective: Find a permutation π of jobs so that a given criterion is optimized: sequence.

$n!$ possible solutions

Makespan minimization ($C_{\max}$) is the most common objective

NP-Complete for $m \geq 3$ (Garey et al., 1976)

Denoted as $F/prmu/C_{\max}$

The flowshop problem

If we have a permutation π of jobs and $\pi_{(j)}$ denotes the job occupying the j –th position, then:

$$O_{i,\pi_{(j)}}s = \max\left\{O_{i,\pi_{(j-1)}}f; O_{i-1,\pi_{(j)}}f\right\}$$
$$O_{i,\pi_{(j)}}f = O_{i,\pi_{(j)}}s + p_{i,\pi_{(j)}}, \quad i = (1,\ldots,m), j = (1,\ldots,n)$$

We have that $O_{0j}f = 0, j = (1,\ldots,n)$ and $O_{i0}f = 0, i = (1,\ldots,m)$

From the start and finish times:

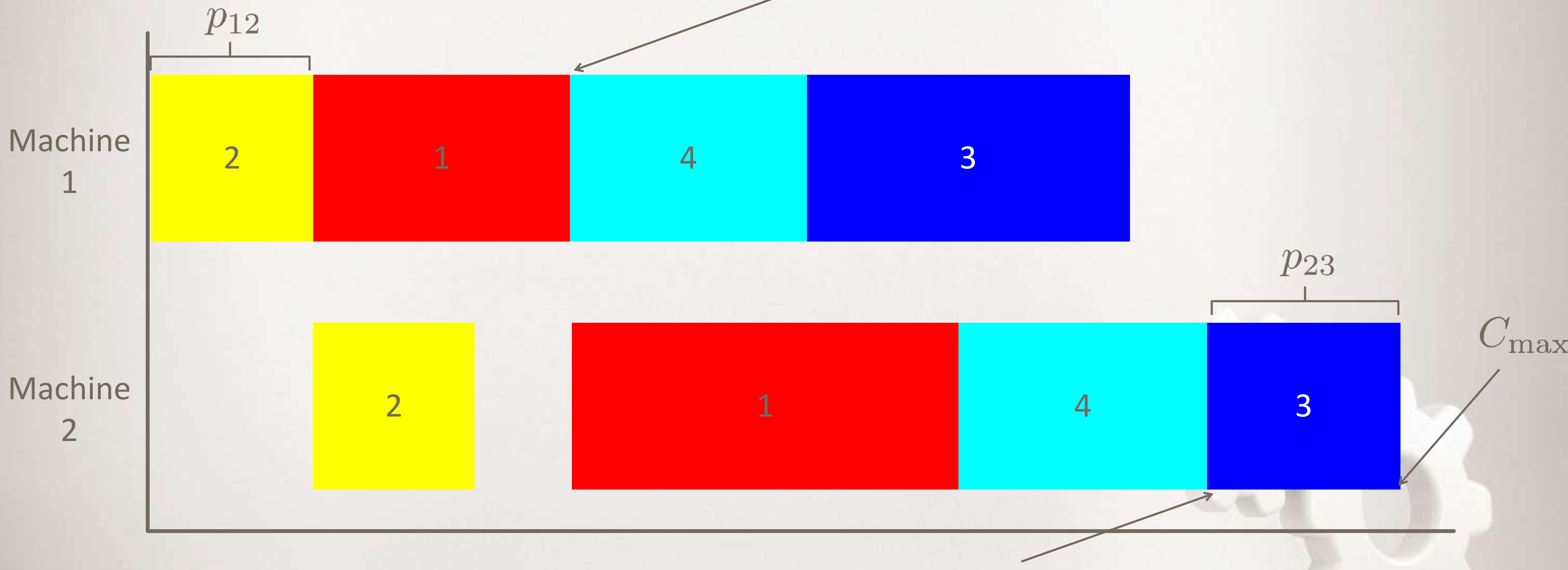
$$C_{\pi_{(j)}} = O_{m,\pi_{(j)}}f, \quad j = (1,\ldots,n)$$

And therefore $C_{\max} = \max\left\{C_{\pi_{(1)}}, C_{\pi_{(2)}}, \ldots, C_{\pi_{(n)}}\right\}$

The flowshop problem

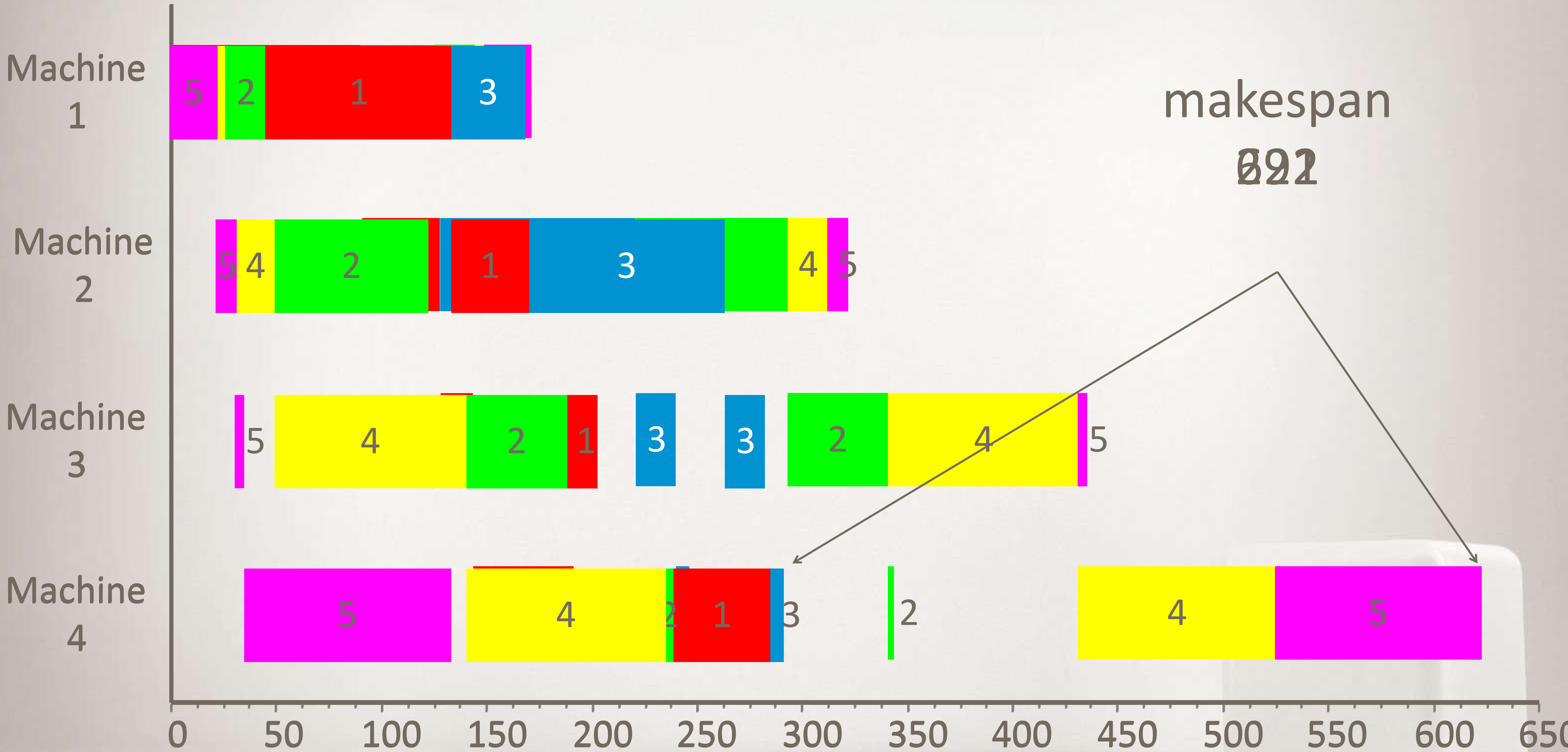
$\pi = \{2, 1, 4, 3\} \qquad \pi_{(1)} = 2, \qquad \pi_{(3)} = 4$

$O_{1,\pi_{(3)}}s = \max\{O_{1,\pi_{(2)}}f; O_{0,\pi_{(3)}}f\}$



$O_{2,\pi_{(4)}}s = \max\{O_{2,\pi_{(3)}}f; O_{1,\pi_{(4)}}f\}$

The flowshop problem



The flowshop problem

Complex and hard to reproduce state-of-the-art

- TSAB of Nowicki and Smutnicki (1996)
- RY of Reeves and Yamada (1998)
- TSGW of Grabowski and Wodecki (2004)
- PACO and M-MMAS of Rajendran and Ziegler (2004)

Algorithms full of operators, accelerations and problem – specific knowledge = bad reproducibility and inability to extend to other problems

3. Basic IG algorithm

Initialization

Local search (optional)

While stopping criterion not satisfied

- Random partial destruction
- Greedy reconstruction
- Local search (optional)
- Acceptance criterion

Solution representation

The most natural is a permutation of size n

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	12	2	5	13	9	3	1	17	19	10	6	15	20	18	11	16	4	14	8

Easy to code by an array or a list

Initialization

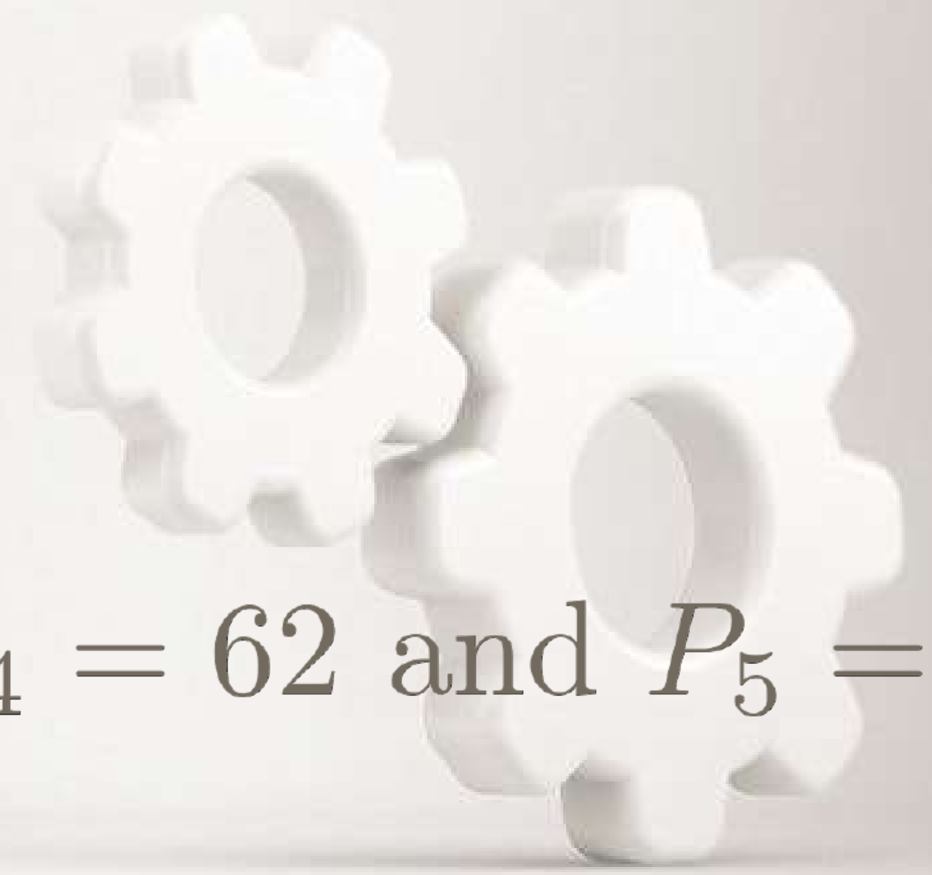
It is very common to use effective heuristics to obtain good initial solutions

In the flowshop problem with makespan minimization the most cited and high performing heuristic is the NEH of Nawaz, Enscore and Ham (1983) (Ruiz and Maroto, 2005)

Initialization

Let us see a NEH example with 5 jobs and 4 machines (5×4):

	Jobs				
Machines	**1**	**2**	**3**	**4**	**5**
1	31	19	23	13	33
2	41	55	42	22	5
3	25	3	27	14	57
4	30	34	6	13	19



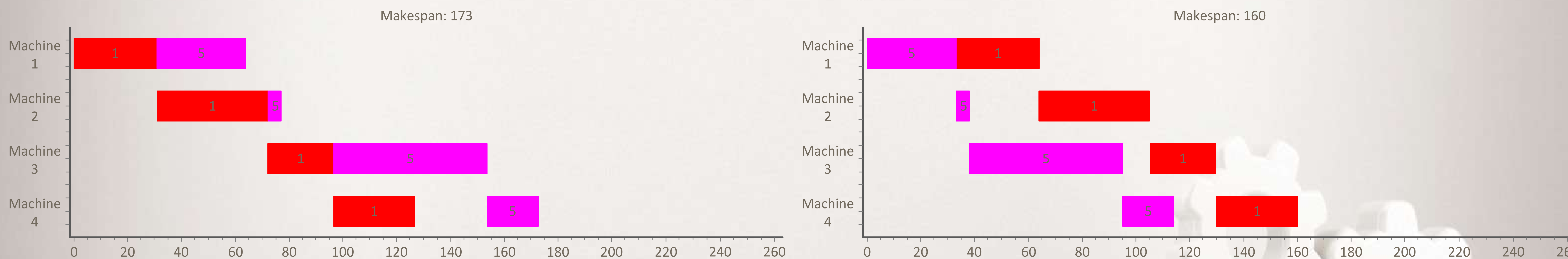
Calculate the P_j :

$$P_1 = 31 + 41 + 25 + 30 = 127,\ P_2 = 111,\ P_3 = 98,\ P_4 = 62 \text{ and } P_5 = 114$$

Initialization

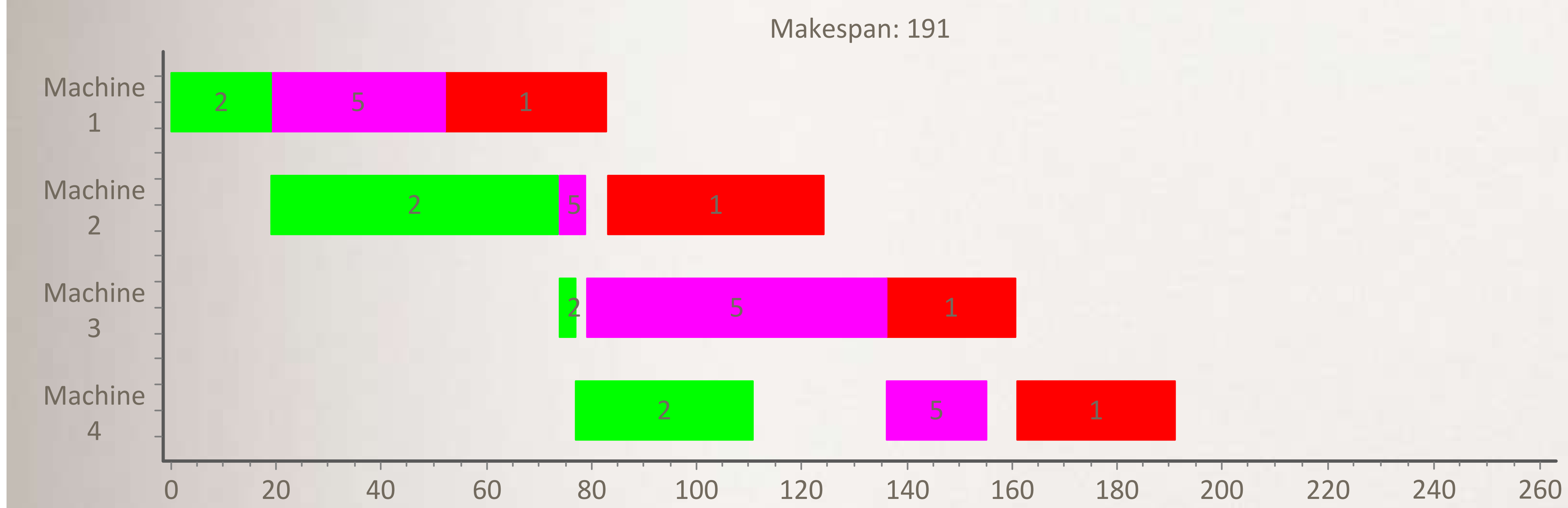
The jobs, ordered by P_j are: $\ell = \{1, 5, 2, 3, 4\}$

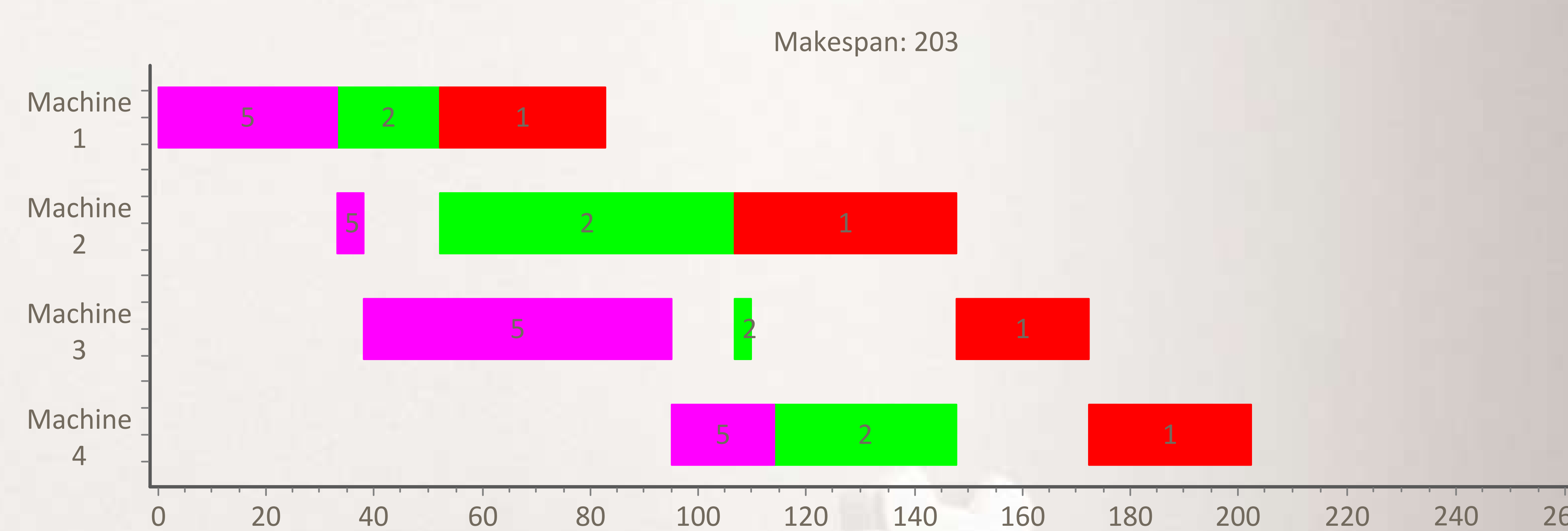
Take the first two jobs and test the sequences {1,5} and {5,1}:

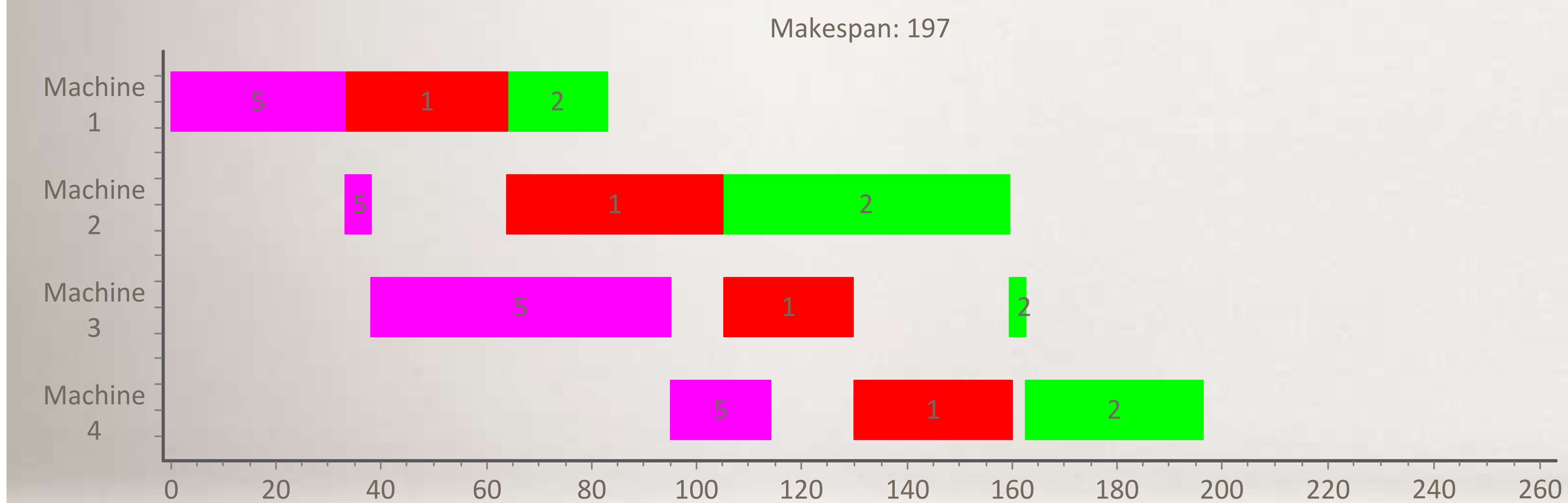


Initialization

The next job in the list is 2, therefore we have to test the sequences {2,5,1}, {5,2,1} and {5,1,2} :

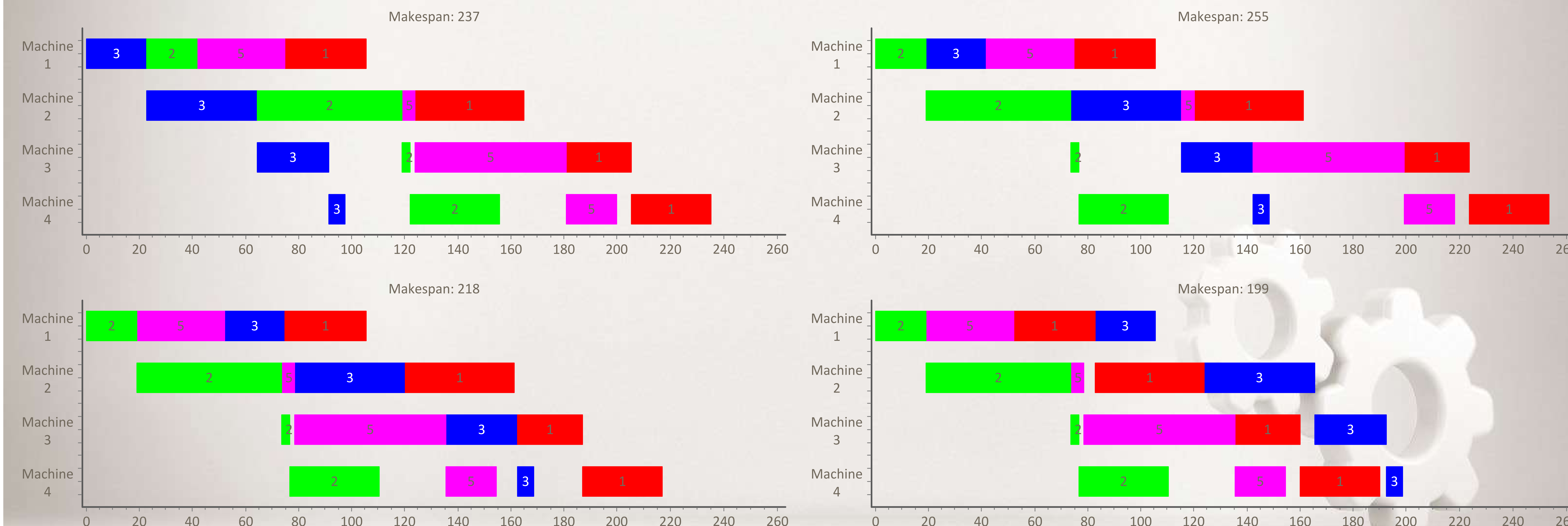






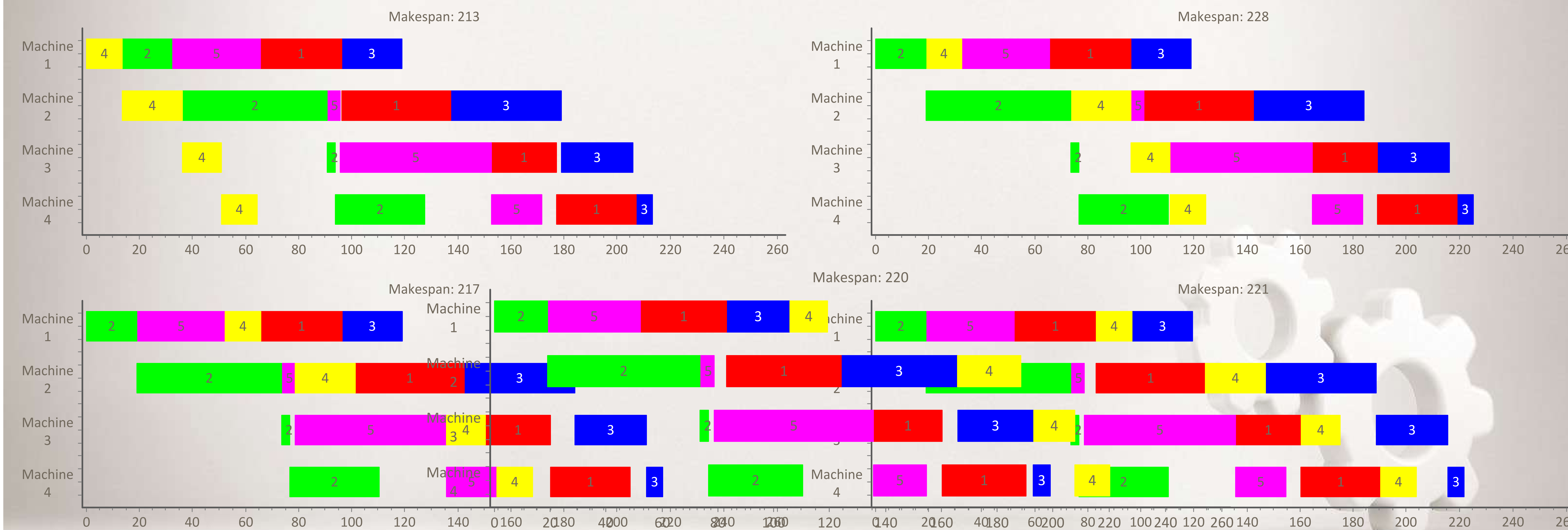
Initialization

Now we insert job 3, so the sequences to test are {3,2,5,1}, {2,3,5,1}, {2,5,3,1} and {2,5,1,3}:



Initialization

Now we insert the last job. There are 5 possible sequences {4,2,5,1,3}, {2,4,5,1,3}, {2,5,4,1,3}, {2,5,1,4,3} and {2,5,1,3,4}:



Initialization

NEH evaluates a total of $[n(n+1)/2]-1$ sequences, where n of these are complete schedules

Computational complexity of $\mathcal{O}(n^3m)$

With Taillard (1993) implementation, complexity goes down to $\mathcal{O}(n^2m)$

In practice, large problems of 500×20 are solved with fast code in less than 30 milliseconds

Destruction

We start from a complete permutation π of n jobs

A random number of jobs are selected ($destruct$)

And are removed from the sequence in the selected order

Two sub-sequences are obtained: The original without the removed jobs : π_D , of size $n - destruct$ and the one with the removed jobs: π_R , of size $destruct$

Reconstruction

NEH's last step is used

We start from subsequence π_D

And carry out $destruct$ iterations

At each iteration the first job of π_R is reinserted in all the positions of π_D ($n - destruct + i$)

The job is placed in the position resulting in the smallest $C_{\max}$

Finished when π_D is complete ($\pi_R = \varnothing$)

Example

Instance Car8 of Carlier. 8×8

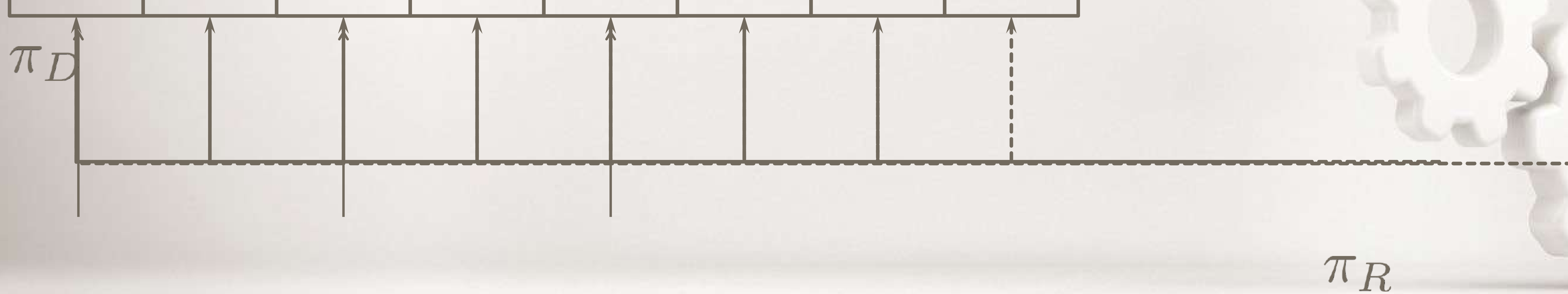
Solution after NEH: {7,3,4,1,8,2,5,6} $C_{\max} = 8564$

Destruction phase

Reconstruction phase

7	3	4	1	8	2	5	6

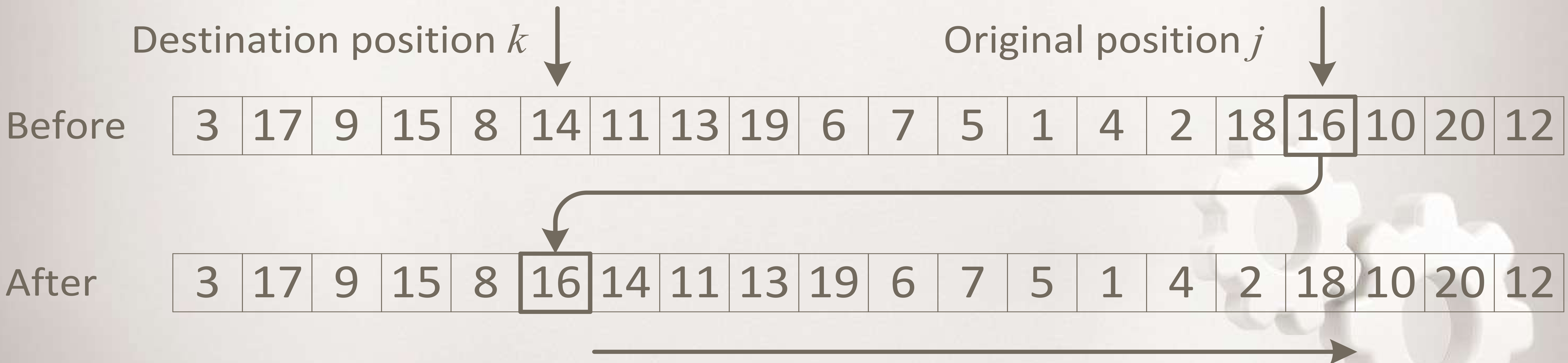
$C_{\max} = 8306$



Local search

Many potential neighborhoods

For the flowshop problem the most effective is insert



Local search

Local search based on the insertion neighborhood

All jobs extracted and reinserted into all possible positions, until local optimality

Insertion neighborhood: Given two positions $j, k \in N, j \neq k$

$$\pi' = \left\{\pi_{(1)}, \ldots, \pi_{(j-1)}, \pi_{(j+1)}, \ldots, \pi_{(k)}, \pi_{(j)}, \pi_{(k+1)}, \ldots, \pi_{(n)}\right\}, (j < k)$$

$$\pi' = \left\{\pi_{(1)}, \ldots, \pi_{(k-1)}, \pi_{(j)}, \pi_{(k+1)}, \ldots, \pi_{(j-1)}, \pi_{(j+1)}, \ldots, \pi_{(n)}\right\}, (j > k)$$

$$I = \{(j,k) : j \neq k,\ 1 \leq j,k \leq n \wedge j \neq k-1,\ 1 \leq j \leq n, 2 \leq k \leq n\}$$

Local search

procedure *Insertion_LocalSearch* (π)

  $improve$:= true;

  while ($improve$ = true) **do**

    $improve$ = false;

    for $i := 1$ **to** n **do**

      extract job k randomly from π (no repetition)

      $\pi' :=$ best permutation after inserting k into all possible positions of π;

      if $C_{\max}(\pi') < C_{\max}(\pi)$ **then**

        $\pi := \pi'$;

        $improve$:= true;

      endif

    endfor

  endwhile

  return π

end

Acceptance criterion

After destruction, reconstruction and optional local search we check if the new solution is accepted

Accepting only better solutions results in premature convergence

We apply a fixed temperature simulated annealing criterion

$$\text{Temperature} = T \cdot \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$$

Iterated Greedy algorithm

procedure *Iterated_Greedy*
  π := NEH_Heuristic;
  π := Insertion_LocalSearch(π);
  $\pi_b := \pi$;
  while (termination criterion not satisfied) **do**
    $\pi' := \pi$;
    for $i := 1$ **to** *destruct* **do** % Destruction phase
      π' := randomly extract a job of π' and insert it into π'_R;
    for $i := 1$ **to** *destruct* **do** % Reconstruction phase
      π' := best permutation after inserting $\pi_R(i)$ into all possible positions of π';
    π'' := Insertion_LocalSearch(π'); % Local Search
    if $C_{\max}(\pi'') < C_{\max}(\pi)$ **then** $\pi := \pi''$; % Acceptance criterion
      if $C_{\max}(\pi) < C_{\max}(\pi_b)$ **then** $\pi_b := \pi$;
    elseif $\left(random \leq e^{-\frac{(C_{\max}(\pi'') - C_{\max}(\pi))}{\text{Temperature}}}\right)$ **then** $\pi := \pi''$;
    endif
  endwhile
  return π_b
end

Calibration

Only two parameters to calibrate $destruct$ and T

DOE, complete factorial design with 42 combinations:

- $destruct$: 7 levels: 2, 3,…,8
- T : 6 levels: 0.0, 0.1,…,0.5

Set of instances where: $n = \{20, 50, 80, \ldots, 440, 470, 500\}$

and $m = \{5, 10, 15, 20\}$ where $p_{ij} \in U[1, 99]$

68 combinations × 5 replicates = 340 calibration instances

Calibration

Each algorithm configuration solves the 340 instances

AMD Athlon XP 1600+ (1.4 GHz) and 512 MBytes of RAM

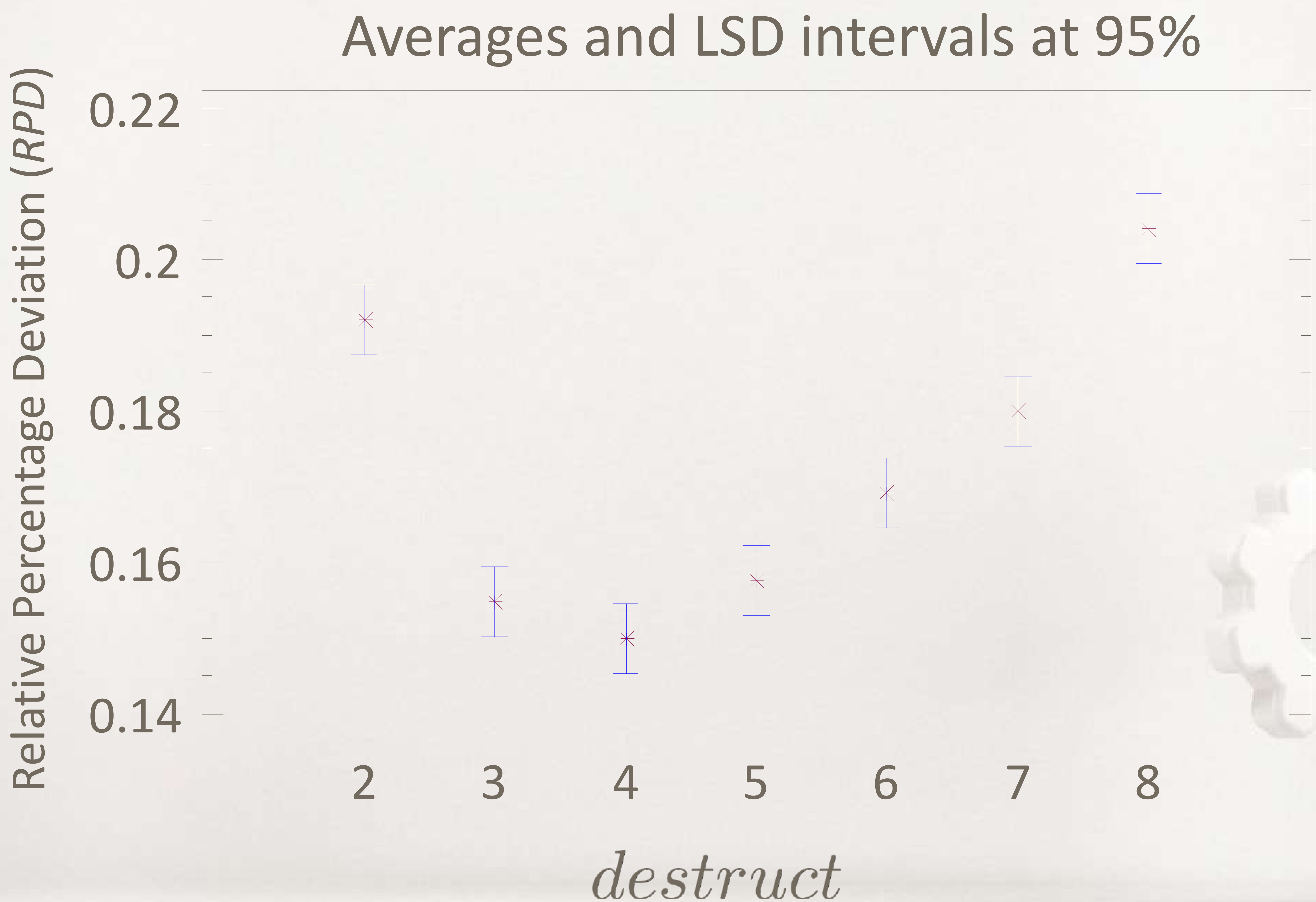
Stopping criterion set to $n \cdot (m/2) \cdot 20$ elapsed milliseconds

Response variable:

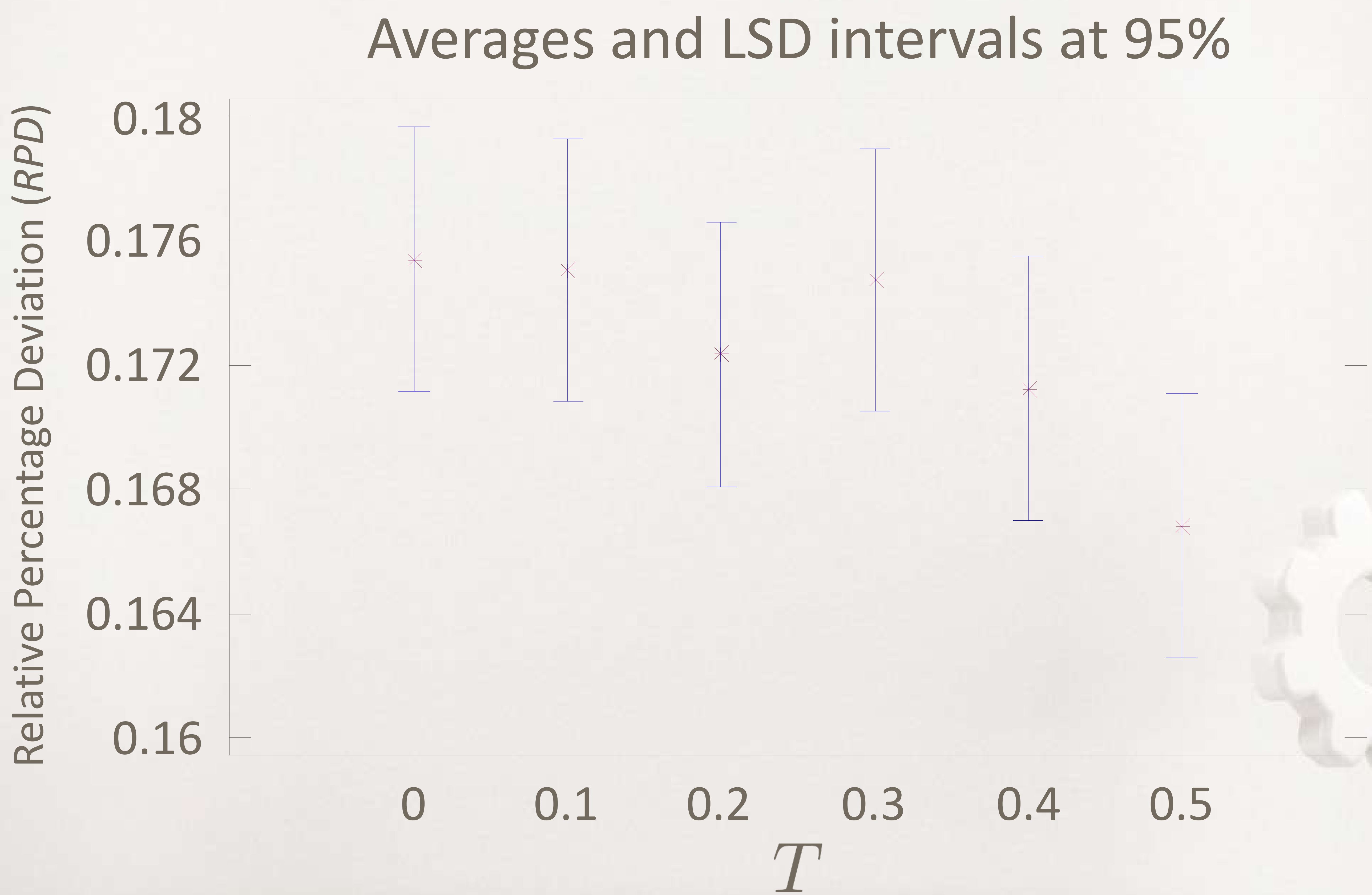
$$\text{Relative Percentage Deviation } (RPD) = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$$

Calibration

Multifactor ANOVA



Calibration



Comparison

We compare two IG versions, with and without local search: IG_RS e IG_RS$_{LS}$

120 Taillard (1993) instances

12 reimplemented methods from the literature

Stopping criterion $n \cdot (m/2) \cdot 60$ ellaped milliseconds

Response variable:

$$\text{Av. Rel. Pertentage Deviation } (\overline{RPD}) = \sum_{i=1}^{R} \left(\frac{Heu_{sol_i} - Best_{sol}}{Best_{sol}} \cdot 100 \right) / R$$

Comparison

NEH of Nawaz et al. (1983) with Taillard (1990) accelerations: **NEHT**

Simulated annealing of Osman and Potts (1989): **SA_OP**

Tabu Search of Widmer and Hertz (1989): **SPIRIT**

GA of Reeves (1995): **GA_REEV**, of Chen et al. (1995): **GA_CHEN**, of Murata et al. (1996): **GA_MIT**, of Aldowaisan Allahverdi (2003): **GA_AA** and Ruiz et al. (2006): **GA_RMA** and **HGA_RMA**

Iterated Local Search of Stützle (1998): **ILS**

Ant Colony Optimization of Rajendran and Ziegler (2004): **M-MMAS** and **PACO**

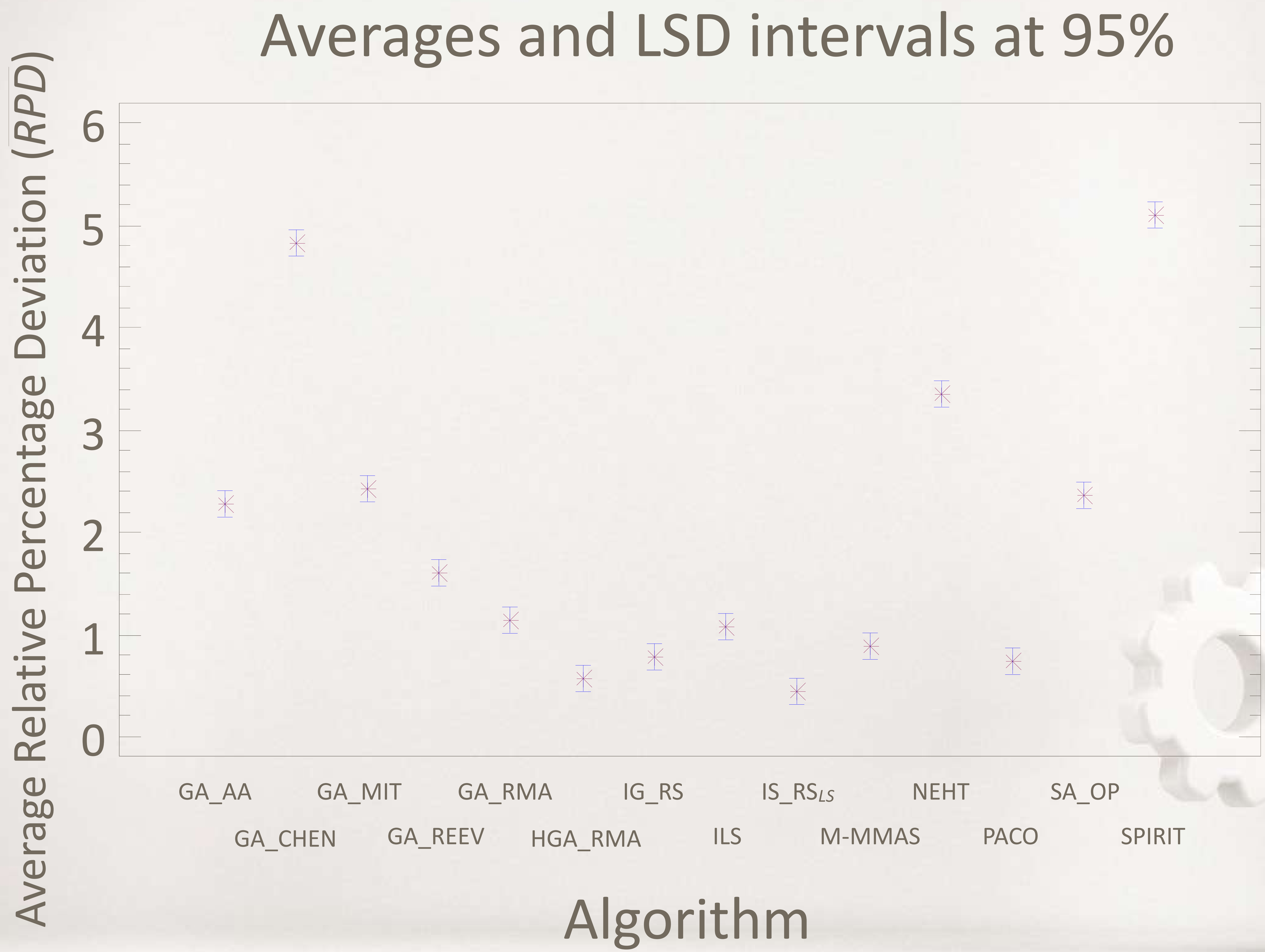
Comparison

Average results:

Method	NEHT	GA_RMA	HGA_RMA	SA_OP	SPIRIT	GA_CHEN	GA_REEV
AVRPD	3.35	1.13	0.57	2.37	5.09	4.83	1.61

Method	GA_MIT	ILS	GA_AA	M_MMAS	PACO	IG_RS	IG_RS_{LS}
AVRPD	2.42	1.06	2.28	0.88	0.75	0.78	0.44

Comparison



Averages and LSD intervals at 95%
Average Relative Percentage Deviation (RPD)
6
5
4
3
2
1
0
GA_AA
GA_CHEN
GA_MIT
GA_REEV
GA_RMA
HGA_RMA
IG_RS
ILS
IS_RSLS
M-MMAS
NEHT
PACO
SA_OP
SPIRIT
Algorithm

Comparison

- Comparison not possible with other methods
  - Tabu Search of Nowicki and Smutnicki (1996): **TSAB**
  - GA with Path Relinking of Reeves and Yamada (1998): **RY**
  - Tabu Search of Grabowski and Wodecki (2004): **TSGW**
- Reimplementation impossible: very difficult to code
- Source codes/binaries not provided in any of the three cases

Comparison

Comparing published results:

Size	IG_RS_{LS}	TSGW	RY	TSAB
50×20	0.02	0.15	0.18	0.24
100×20	0.23	0.26	0.56	0.39
200×20	0.38	0.17	0.55	0.23

Comparison

Later IG results of Vallada and Ruiz (2009) using more modern computers and parallel computing:

	Number of processors			
	2	4	6	8
AVRPD	0.31	0.27	0.25	0.23

Iterated Greedy algorithm

Very simple method

State-of-the-art results

No problem specific knowledge

Easy to extend to other problems and objectives

Very important to carry out detailed and statistically sound testing

Iterated Greedy algorithm

Reference

R. Ruiz and T. Stützle (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *European Journal of Operational Research*, 177(3): 2033-2049.

4. Results for other flowshop problems

Makespan is not the most realistic criterion in practice

Total flowtime: $$TFT = \sum_{j=1}^{n} C_j$$

Weighted tardiness. Given a due date d_j for each job and a priority or importance (weight) w_j :

$$TFT = \sum_{j=1}^{n} w_j T_j$$

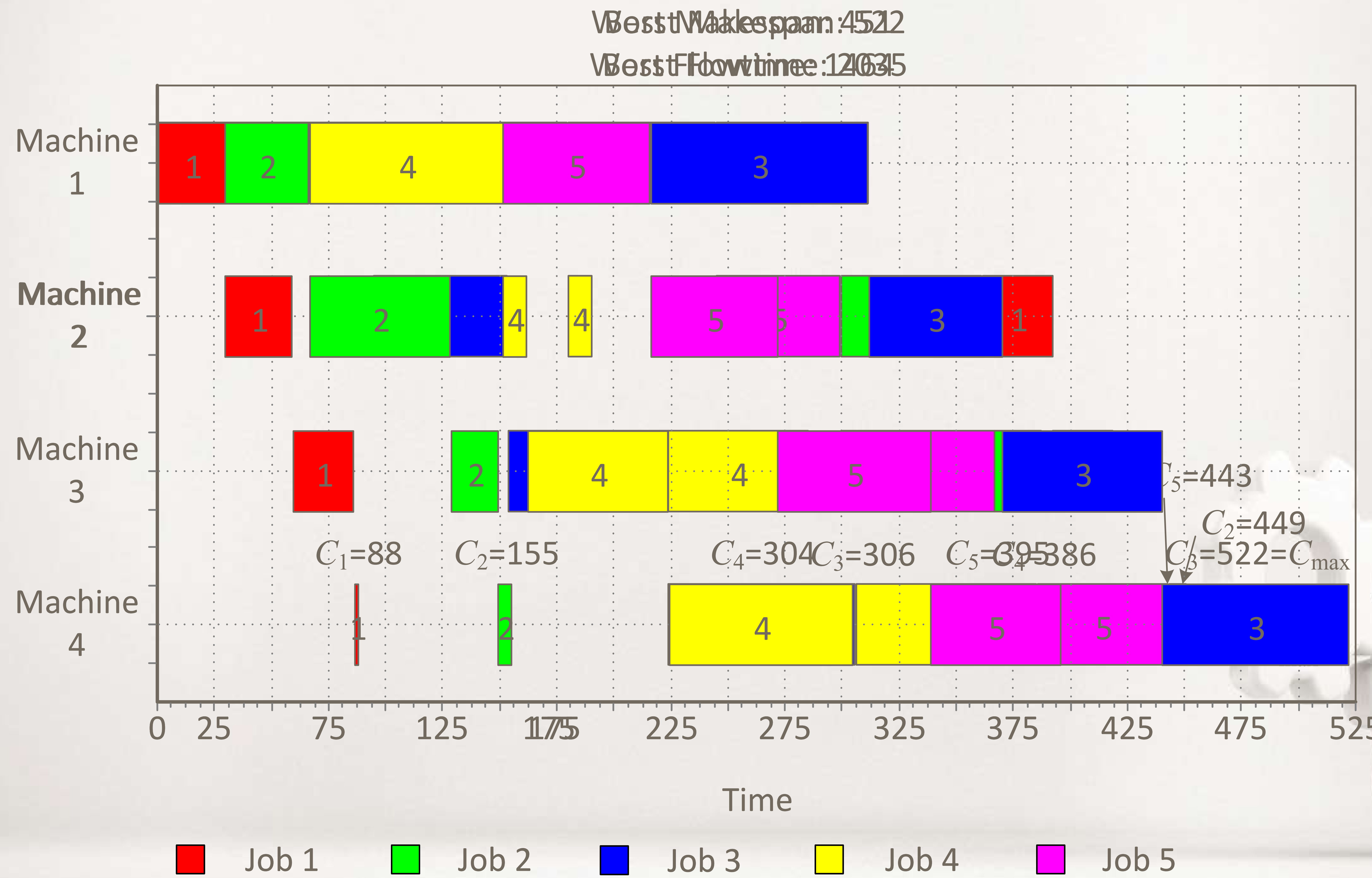
Where T_j is the tardiness of a job j, calculated as:

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

Do we need to change the IG to obtain good results?

Total flowtime

Total flowtime is not correlated with makespan



Total flowtime. *Iterated Greedy* algorithm

Same algorithm

We only change the initialization from NEH to LR(n/m) of Li et al. (2009)

Local search is a variant of the insertion of Rajendran and Ziegler (1997)

Everything else is the same

Total flowtime. *Iterated Greedy* algorithm

Let us compare results with 12 other methods:

1. Discrete Differential Evolution DDE_{RLS} 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)

Total flowtime. Evaluation

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

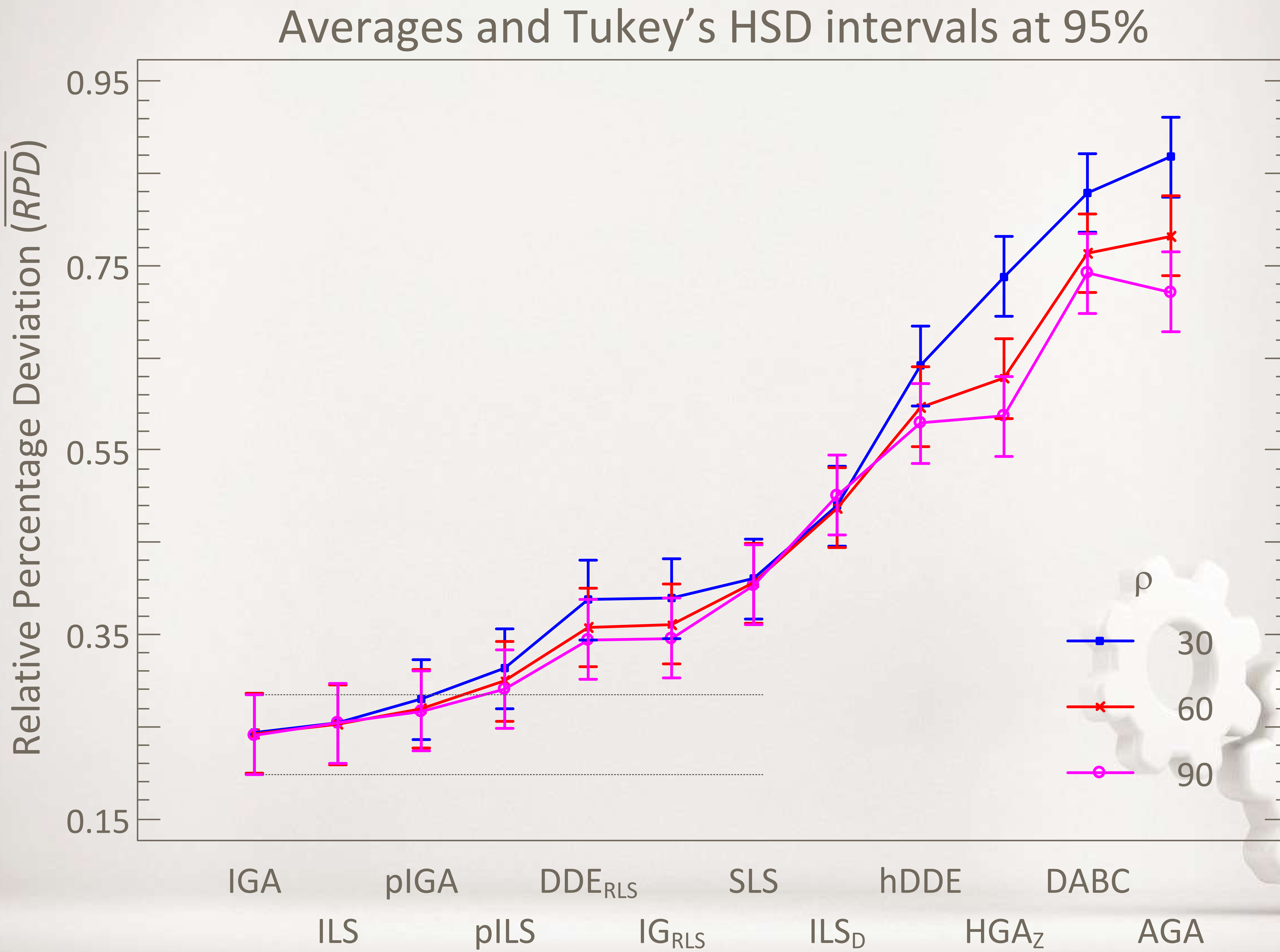
	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}
AVRPD	0.36	0.36	*7.02*	4.39	0.49	2.13	0.63	4.50
	AGA	hDDE	DABC	SLS	IGA	pIGA	ILS	pILS
AVRPD	0.78	0.60	0.76	0.41	**0.24**	0.27	0.25	0.30

ρ=90

	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}
AVRPD	0.35	0.4	*6.64*	4.17	0.50	2.09	0.59	4.09
	AGA	hDDE	DA	LS	IGA	pIGA	ILS	pILS
AVRPD	0.72	0.58	0.	.40	**0.24**	0.27	0.25	0.29

42% better

Total flowtime. Evaluation



Total flowtime. Evaluation

- ILS is a bit more complex than IG. Works a bit worse
- pIG is a population variant of IG
  - It has a population of solutions
  - A selection operator is needed
  - A generational scheme and a convergence control operator is needed
- pIG is worse and takes more time to converge

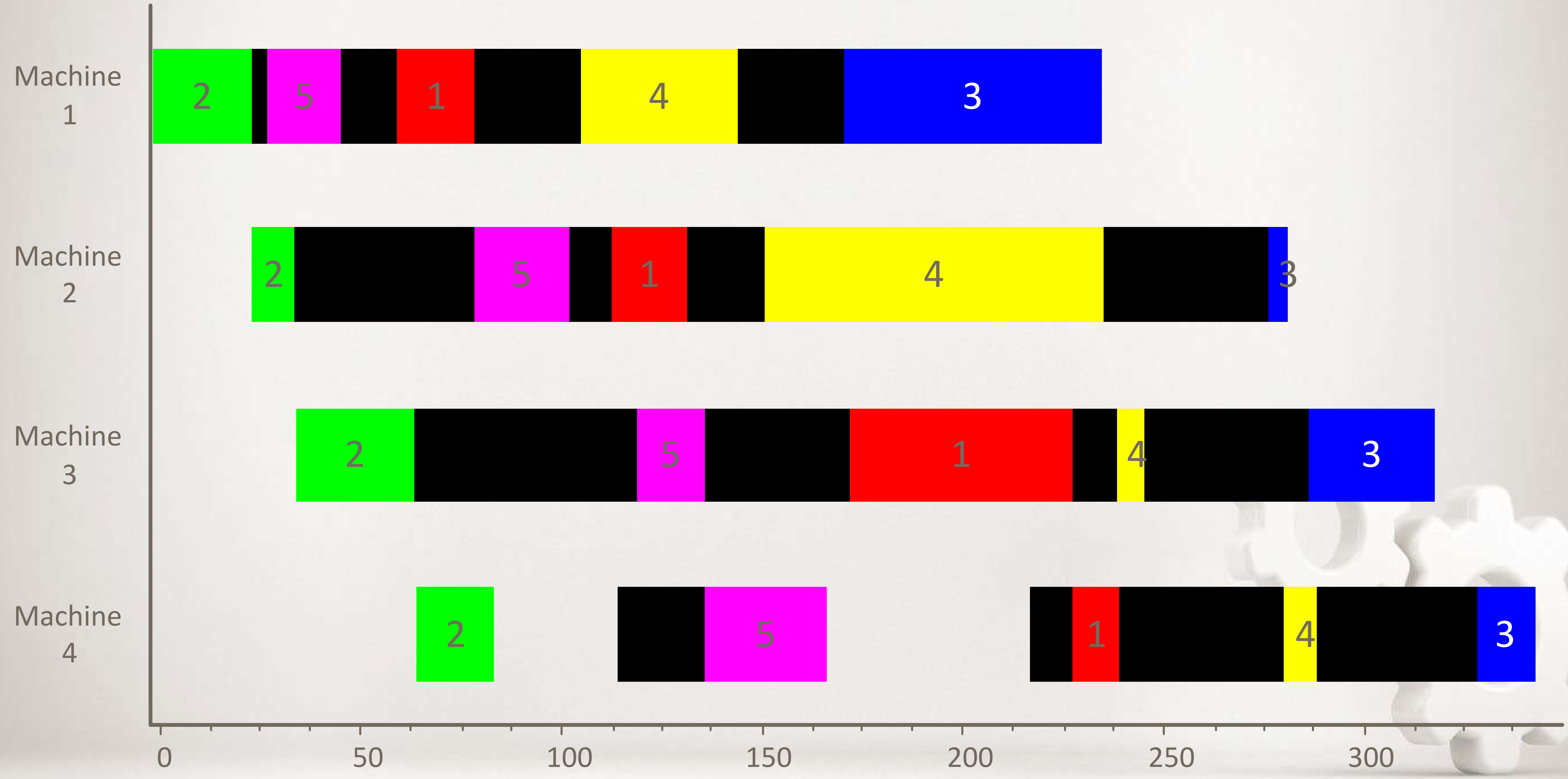
Total flowtime

Reference

Q.-K. Pan and R. Ruiz (2012). Local search methods for the flowshop scheduling problem with flowtime minimization, *European Journal of Operational Research*, 222(1): 31-43.

Sequence dependent setup times

Cleaning, fixing, reconfiguring, etc.



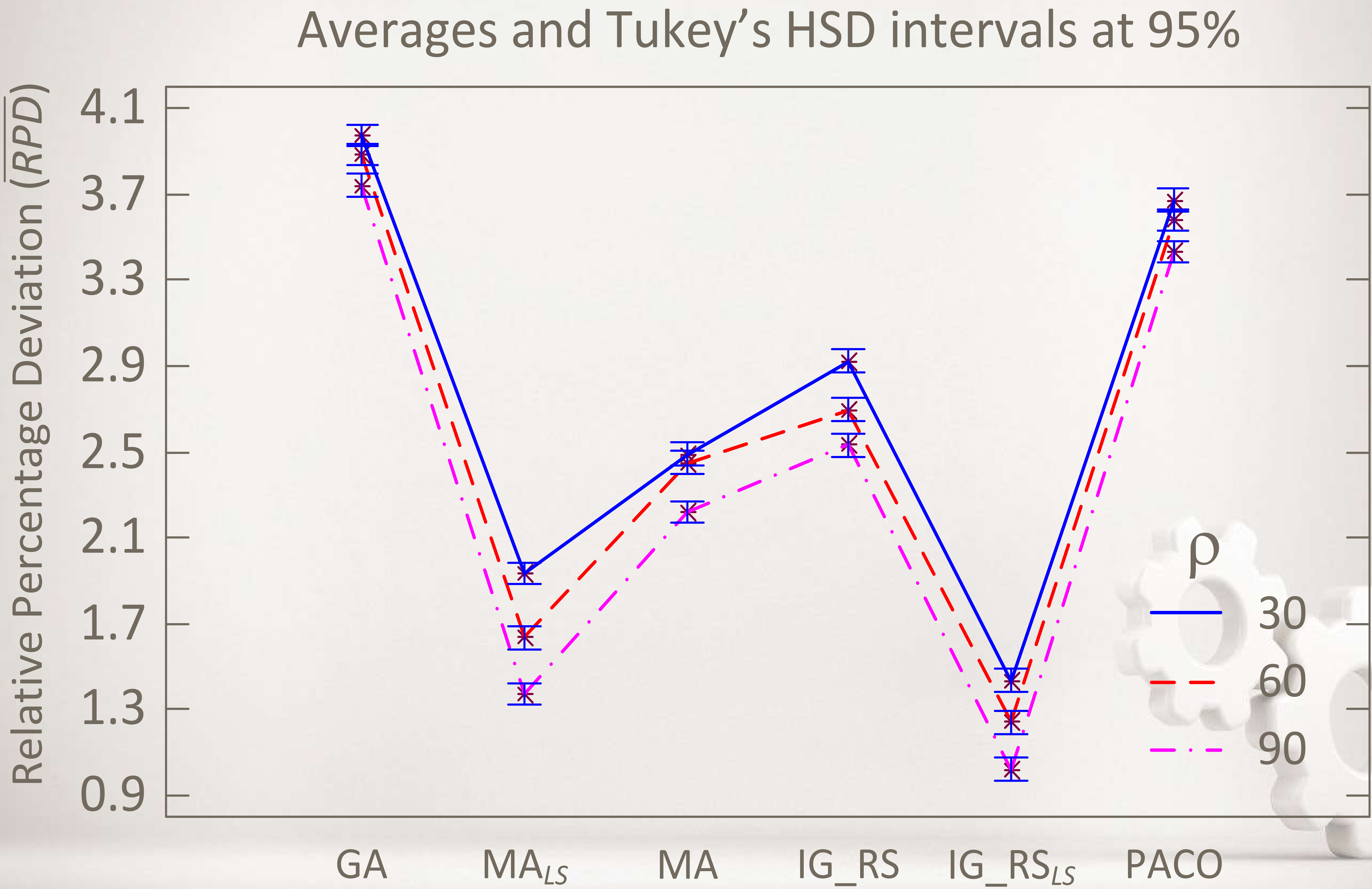
SDST. *Iterated Greedy* algorithm

Same algorithm

We only adapt the NEH initialization to NEH with setups of Ríos-Mercado and Bard (1998). Trivial change

Everything else the same. We only change the objective function evaluation, which considers setup times

SDST. Evaluation



SDST. Evaluation

A big change in the problem does not affect the general IG algorithm

IG is more simple than the other approaches

The more we complicate the algorithm, the worse results

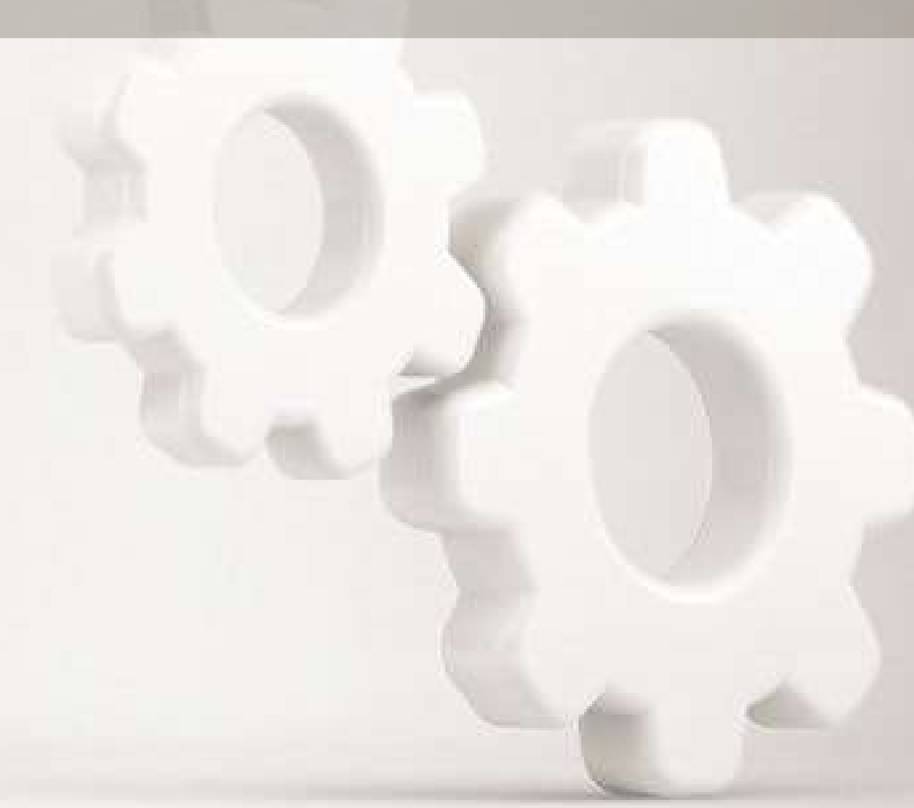
Better results are obtained

Results maintained for weighted tardiness as well

Sequence dependent setup times

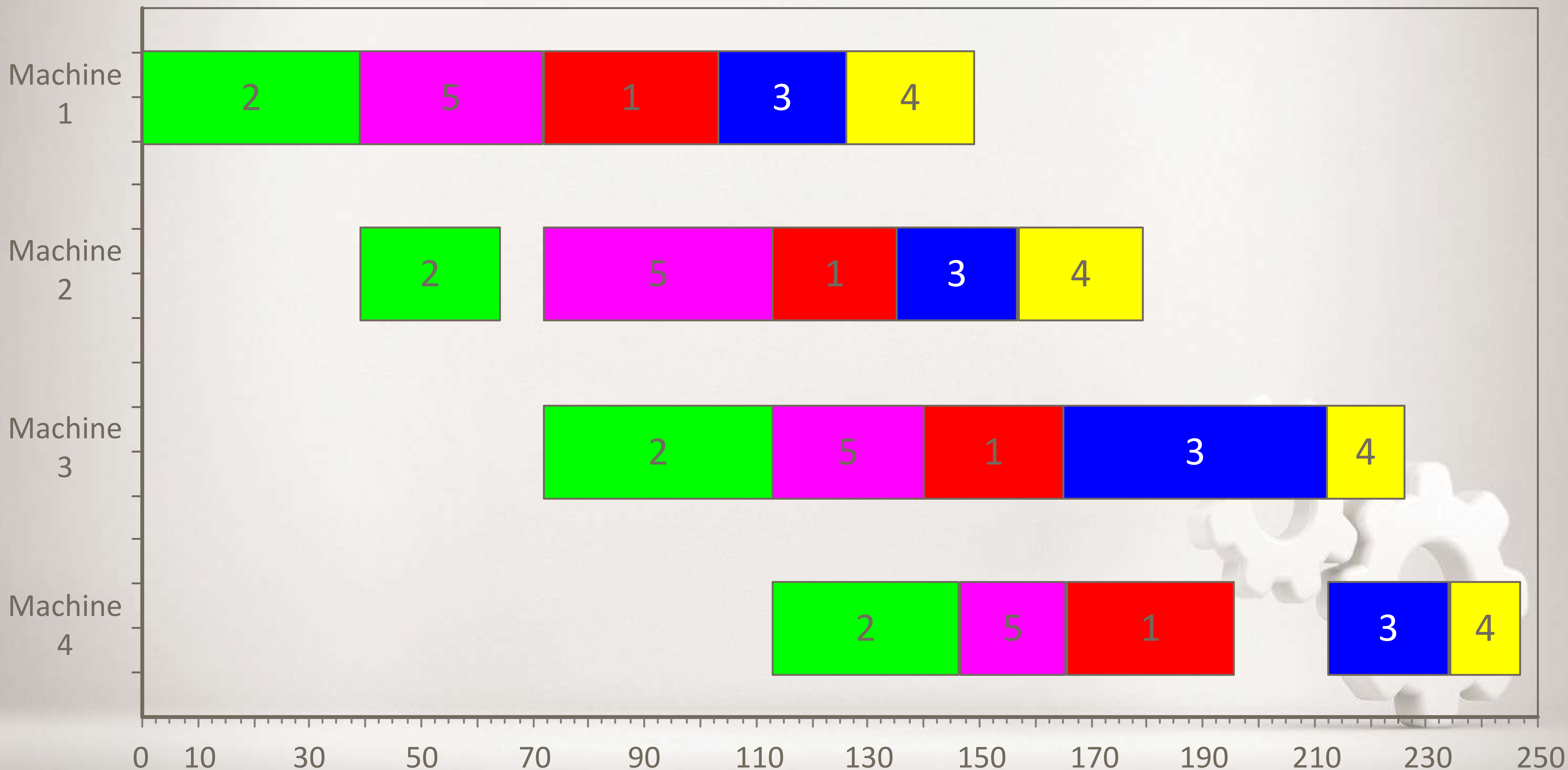
Reference

R. Ruiz and T. Stützle (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, *European Journal of Operational Research*, 187(3): 1143-1159.



Machines that cannot stop (no-idle)

Mixed no-idle flowshop:



No-idle machines. *Iterated Greedy* algorithm

Same algorithm

We change again the initialization to a more efficient NEH version of Rad et al. (2009)

A slightly modified local search and a reconstruction operator with more insertions

Everything else the same

No-idle machines. Evaluation

Comparison with other 7 methods:

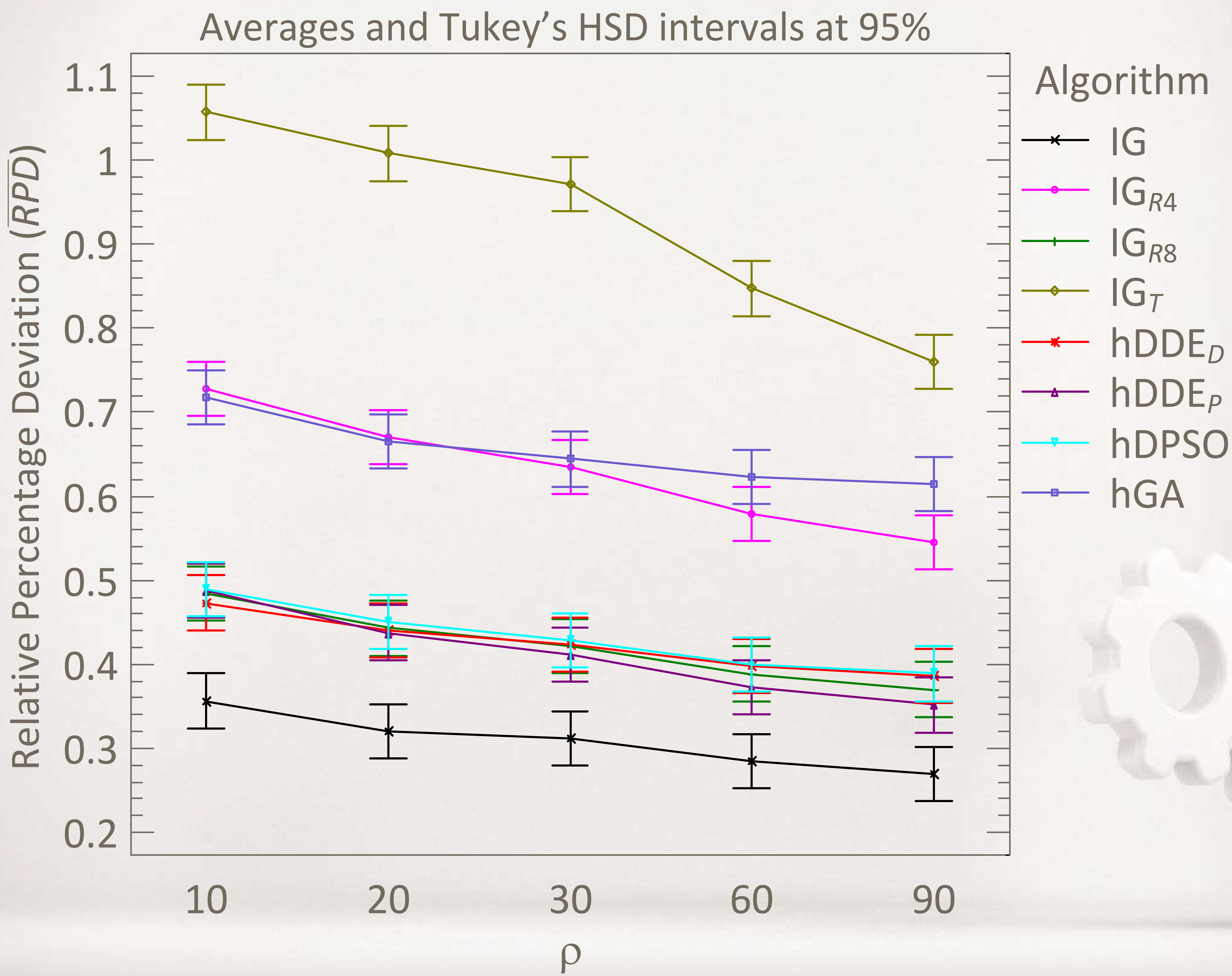
1. Hybrid GA of Ruiz et al. (2006) (hGA)
2. Hybrid Ant Colony Optimization method of Pan and Wang (2008a) (hDPSO),
3. Hybrid Discrete Differential Evolution of Pan and Wang (2008b) ($hDDE_P$),
4-5. IG of Ruiz et al. (2009) with d=4 and d=8 (IGR_4) and (IGR_8),
6. Hybrid Discrete Differential Evolution of Deng and Gu (2012) ($hDDE_D$),
7. An IG recently hybridized with differential evolution of Fatih Tasgetiren et al. (2013) (IG_T).

No-idle machines. Evaluation

Instance group	$hDDE_D$	$hDDE_P$	hDPSO	hGA	IG	IG_{R4}	IG_{R8}	IG_T
1	0.42	0.41	0.42	0.65	**0.33**	0.61	0.42	0.95
2	0.42	0.41	0.44	0.66	**0.35**	0.63	0.42	0.96
3	0.42	0.42	0.43	0.66	**0.31**	0.65	0.43	0.95
4	0.47	0.45	0.47	0.74	**0.37**	0.64	0.46	1.14
5	0.44	0.42	0.45	0.68	**0.31**	0.66	0.44	0.98
6	0.42	0.40	0.41	0.62	**0.26**	0.62	0.40	0.81
7	0.39	0.37	0.40	0.56	**0.23**	0.61	0.37	0.71
Average	0.42	0.41	0.43	0.65	**0.31**	0.63	0.42	0.93

32% better

No-idle machines. Evaluation



No-idle machines. Evaluation

Once again a change in the problem does not affect in a significant way the IG

With minimal adaptations IG is clearly superior to other more complex methods

It is confirmed that the more weight we add to the IG, the worse it performs

It is not needed to make it more complex. The simpler the better

No-idle machines

Reference

Q.-K. Pan and R. Ruiz (2014). An effective iterated greedy algorithm for the mixed no-idle flowshop scheduling problem, *OMEGA, The International Journal of Management Science*, 44(1), 41-50.

Also best results for the pure no-idle problem

R. Ruiz, E. Vallada and C. Fernández-Martínez (2009). Scheduling in flowshops with no-idle machines. Book chapter in Computational Intelligence in Flow Shop and Job Shop Scheduling (Editor Uday Chakraborty). Springer, New York

5. Parallel machines

Parallel machines only have a single stage

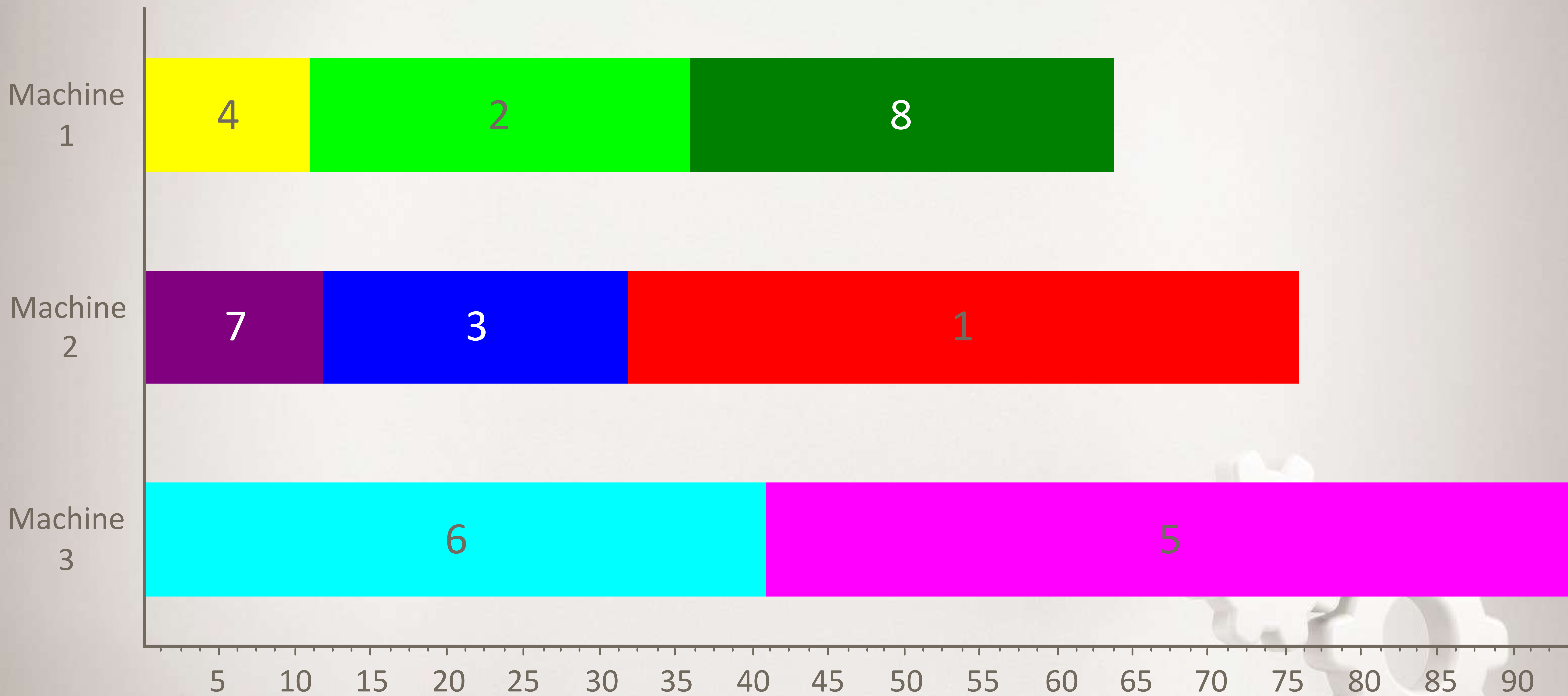
Each job has to be assigned to exactly one out of the m available machines

Most complex version: Unrelated Parallel Machines

Studied a lot in the scheduling literature

Do we need to significantly change the IG?

Parallel machines



Parallel machines. *Iterated Greedy* algorithms

The representation changes significantly. Instead of a permutation we have a set of lists:

	1	2	3	4	5	6	7
Machine 1	7	13	20	14	1	3	6

	1	2	3	4	5	6
Machine 2	12	11	5	19	10	8

	1	2	3	4	5	6	7
Machine 3	2	9	18	17	15	16	4

Parallel machines. *Iterated Greedy* algorithms

In reality if we want to minimize the makespan, the order of the jobs at each machine is irrelevant. It is simply an assignment problem:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	3	1	3	2	1	1	2	3	2	2	2	1	1	3	3	3	3	2	1

Parallel machines. *Iterated Greedy* algorithms

Initial solution very simple: assign each job to the fastest machine

Local search in two neighborhoods:

- Insertion: Extract each job and insert it into all other machines
- Interchange: Interchange all jobs assigned to different machines
- We apply both neighborhoods inside a VND scheme until local optimality

Parallel machines. *Iterated Greedy* algorithms

Random directed destruction (DIG):

The machine generating the makespan has more probability of losing jobs in the destruction operator

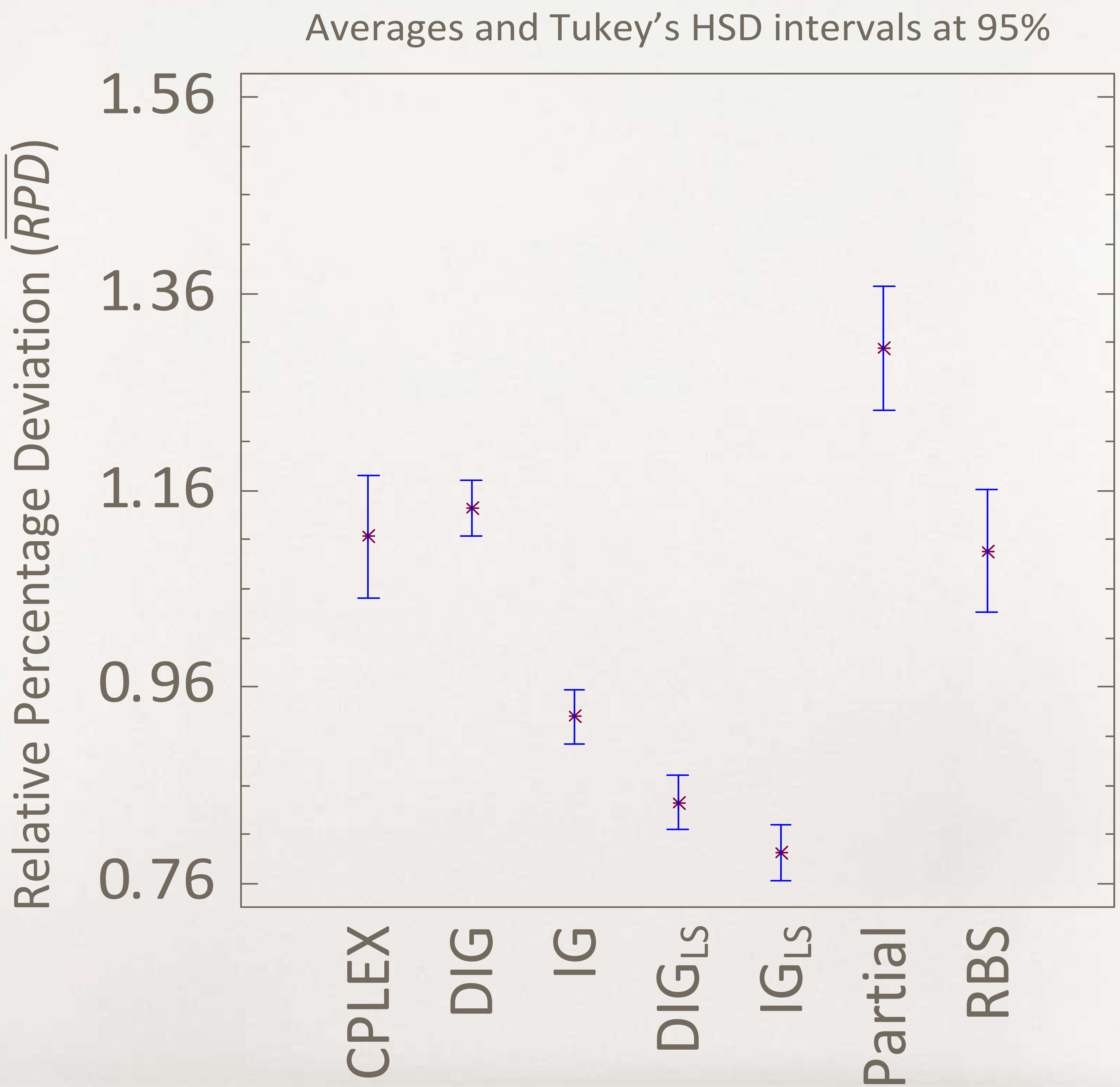
This is the only sensible change in the IG

Parallel machines. Evaluation

Let us compare with three other methods:

1. CPLEX, stopping at the same time than IG
2. PARTIAL of Mokotoff and Jimeno (2002). Takes much longer as it is based on exact techniques
3. Recovering Beam Search (RBS) of Ghirardi and Potts (2005)

Parallel machines. Evaluation



Parallel machines. Evaluation

IG works better than other methods that use CPLEX (RBS and Partial)

Directed destruction "gets in the way"

Local search phase improves results significantly

It is confirmed that IG works with other solution representations

Parallel machines

Reference

L. Fanjul-Peyro and R. Ruiz (2010). Iterated greedy local search methods for unrelated parallel machine scheduling, *European Journal of Operational Research*, 207(1): 55-69.

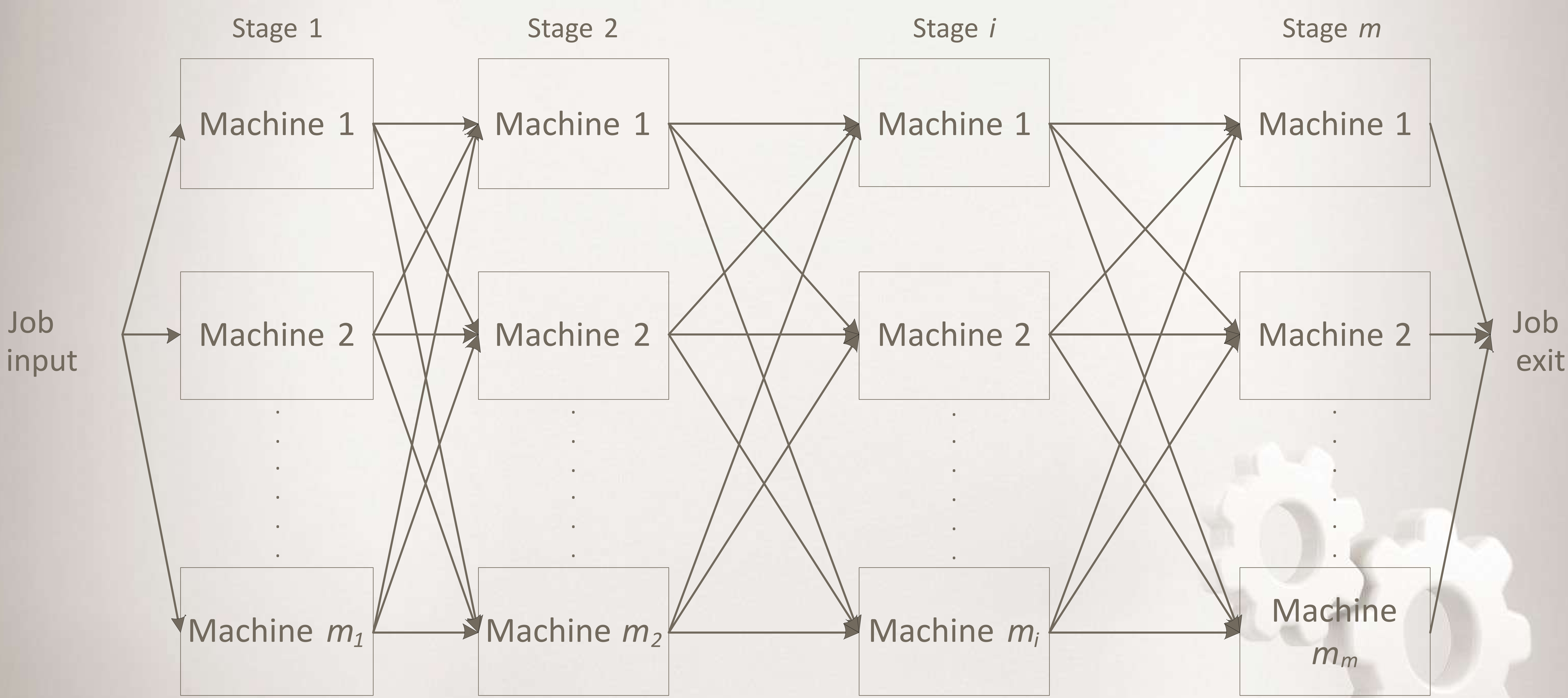
6. Complex hybrid problems

Hybrid flowshops coalesce regular flowshops and parallel machines

Instead of a set of machines in series (flowshop) or in parallel (parallel machines) we have a set of stages in series where each stage has several parallel machines

It is a multiple problem with sequencing and assignment

Complex hybrid problems

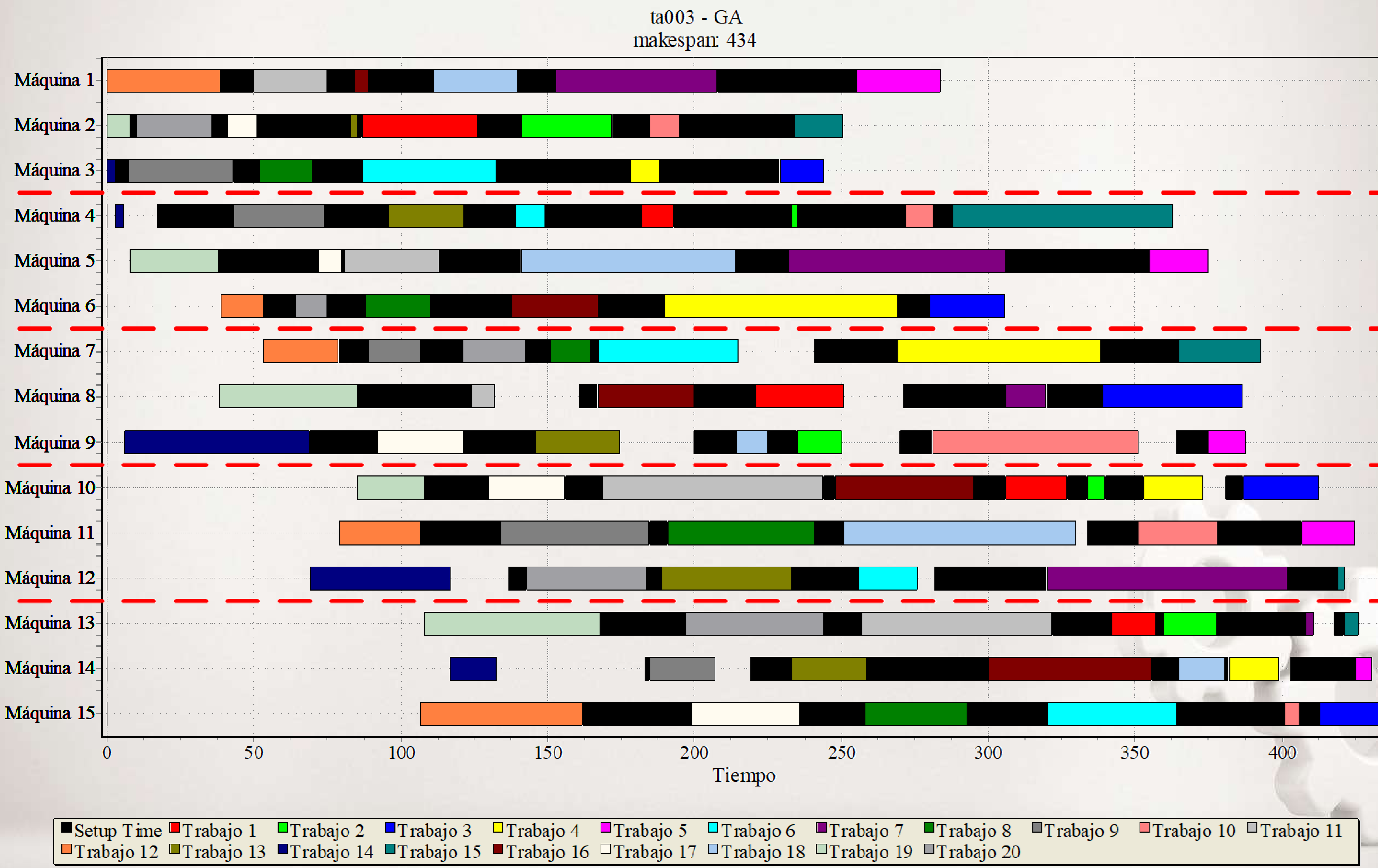


Complex hybrid problems

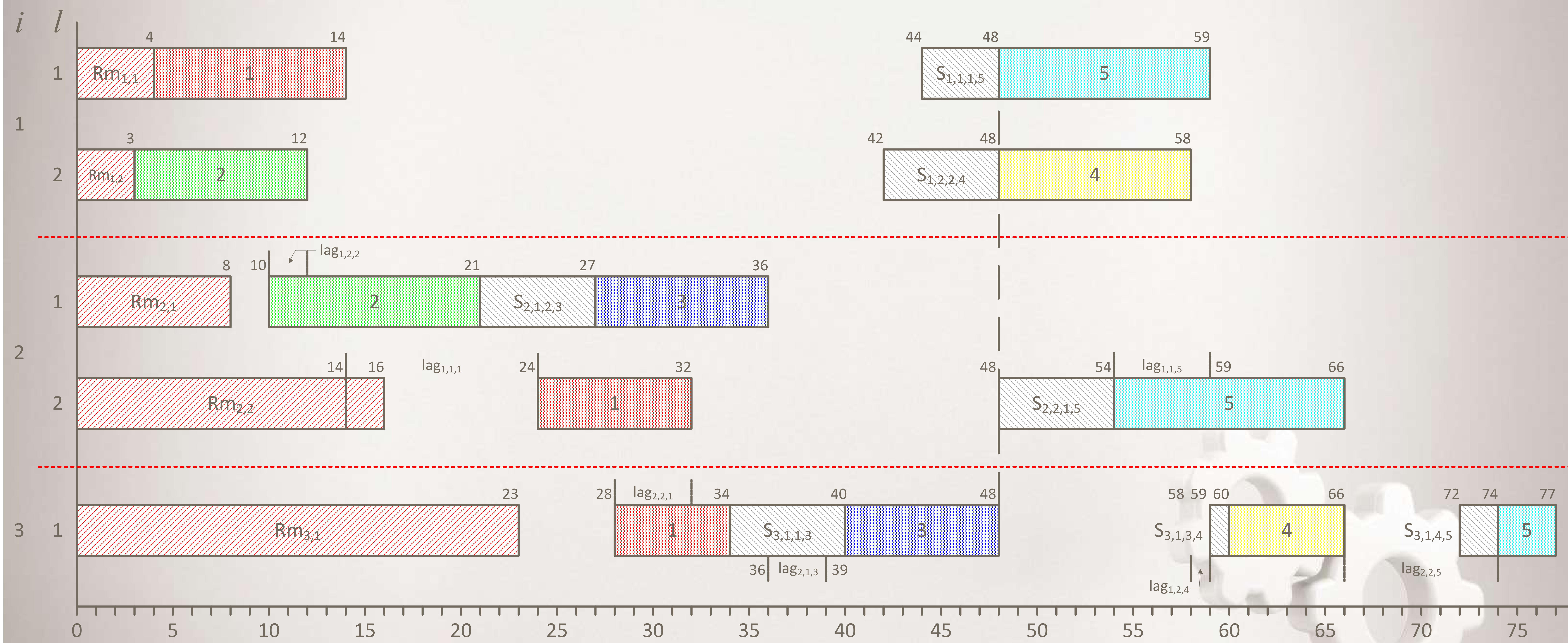
Let us consider a large number of constraints:

- Sequence dependent setup times in all machines
- Unrelated parallel machines at all stages
- Eligibility
- Stage skipping
- Anticipatory and non-anticipatory setup times
- Precedence relationships among jobs
- Lag times and overlaps between operations
- Release times for machines

Complex hybrid problems



Complex hybrid problems



Complex hybrid problems

Solution representation can get really complex

Stage 1

Machine 1

1	2	3	4	5	6
1	2	8	7	5	1

Machine 2

1	2	3	4	5
10	4	9	6	3

Stage 2

Machine 1

1	2	3	4	5
2	10	5	19	2

Machine 2

1	2	3
8	1	6

Machine 3

1	2	3	4	5
3	7	10	3	4

Stage 3

Machine 1

1	2	3	4	5
5	7	5	2	9

Machine 2

1	2	3	4	5	6
1	6	4	3	8	10

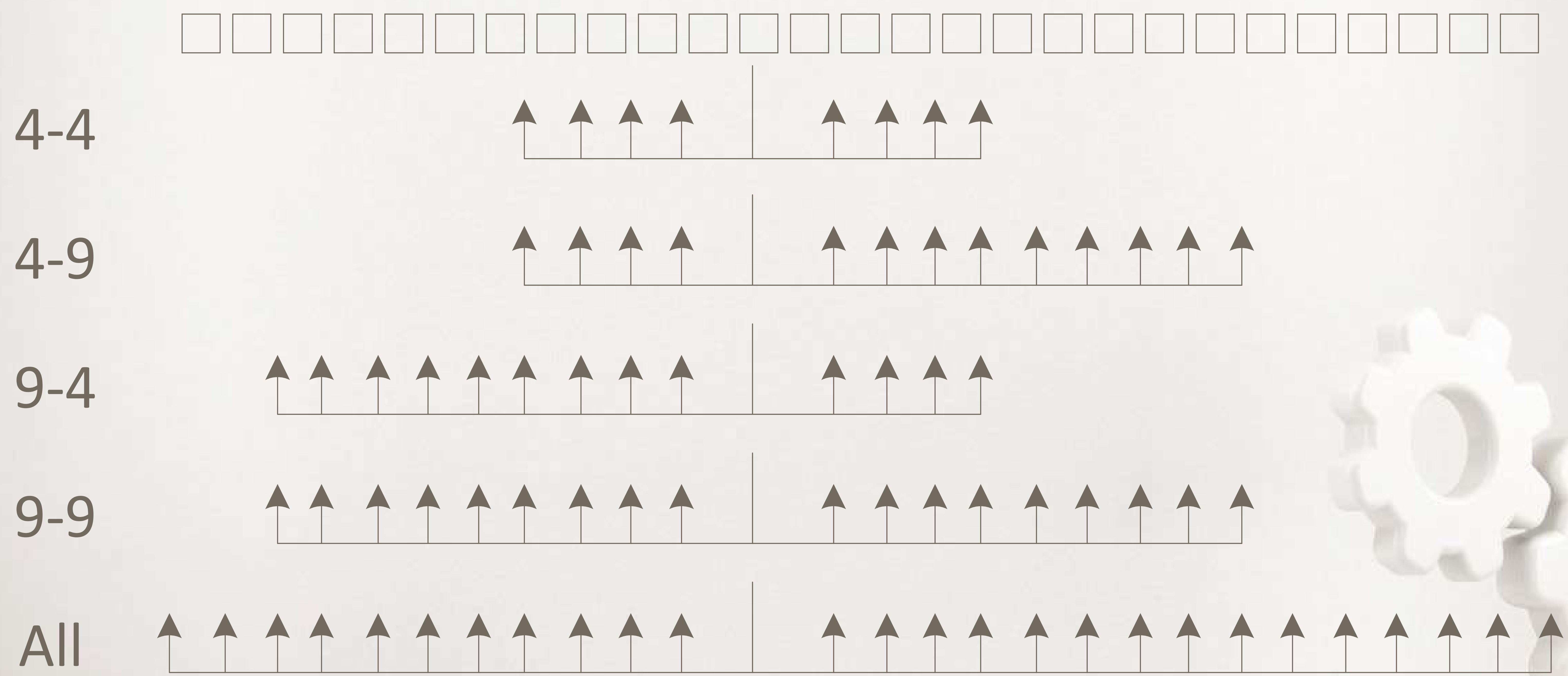
Hybrid flowshops. *Iterated Greedy* algorithm

Solution evaluation very complex. Some changes are needed

Use only a permutation representation (order in which jobs are launched to the shop) and use assignment heuristics to decide which machine should process each job at each stage

Hybrid flowshops. *Iterated Greedy* algorithm

Local search still insertion but with increased span

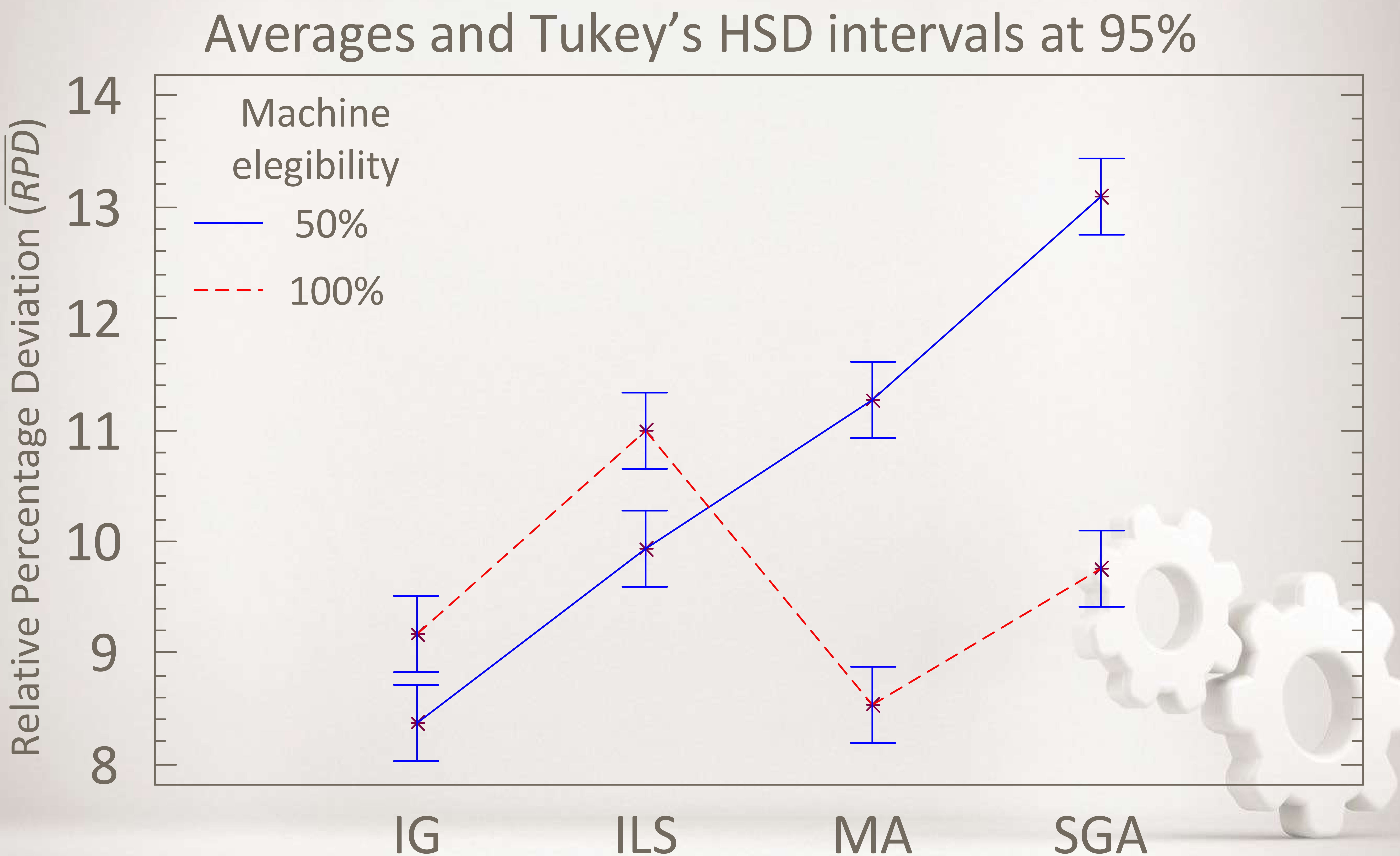


Hybrid flowshops. *Evaluation*

Let us compare with three methods:

1. Steady State GA (Ruiz and Maroto, 2006) (SGA)
2. Hybrid GA (memético) (MA)
3. Iterated Local Search (ILS)

Hybrid flowshops. *Evaluation*



Hybrid flowshops. *Evaluation*

Again, a simple IG is better than complex approaches

In order to keep performance we had to simplify solution representation and local search

General outline of IG remains the same

Hybrid flowshops

Reference

T. Urlings, R. Ruiz and F. Sivrikaya-Şerifoğlu (2007). Local Search in Complex Scheduling Problems, In Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, editors T. Stützle, M. Birattari, and H. H. Hoos, H. H. Lecture Notes in Computer Science, vol. 4638.

Later better results combining two IG the first one with a permutation representation and the second with an exact representation

T. Urlings, R. Ruiz and T. Stützle (2010). Shifting representation search for hybrid flexible flowline problems, *European Journal of Operational Research*, 207(2):1086-1095.

7. Multiobjective

Reality is multiobjective

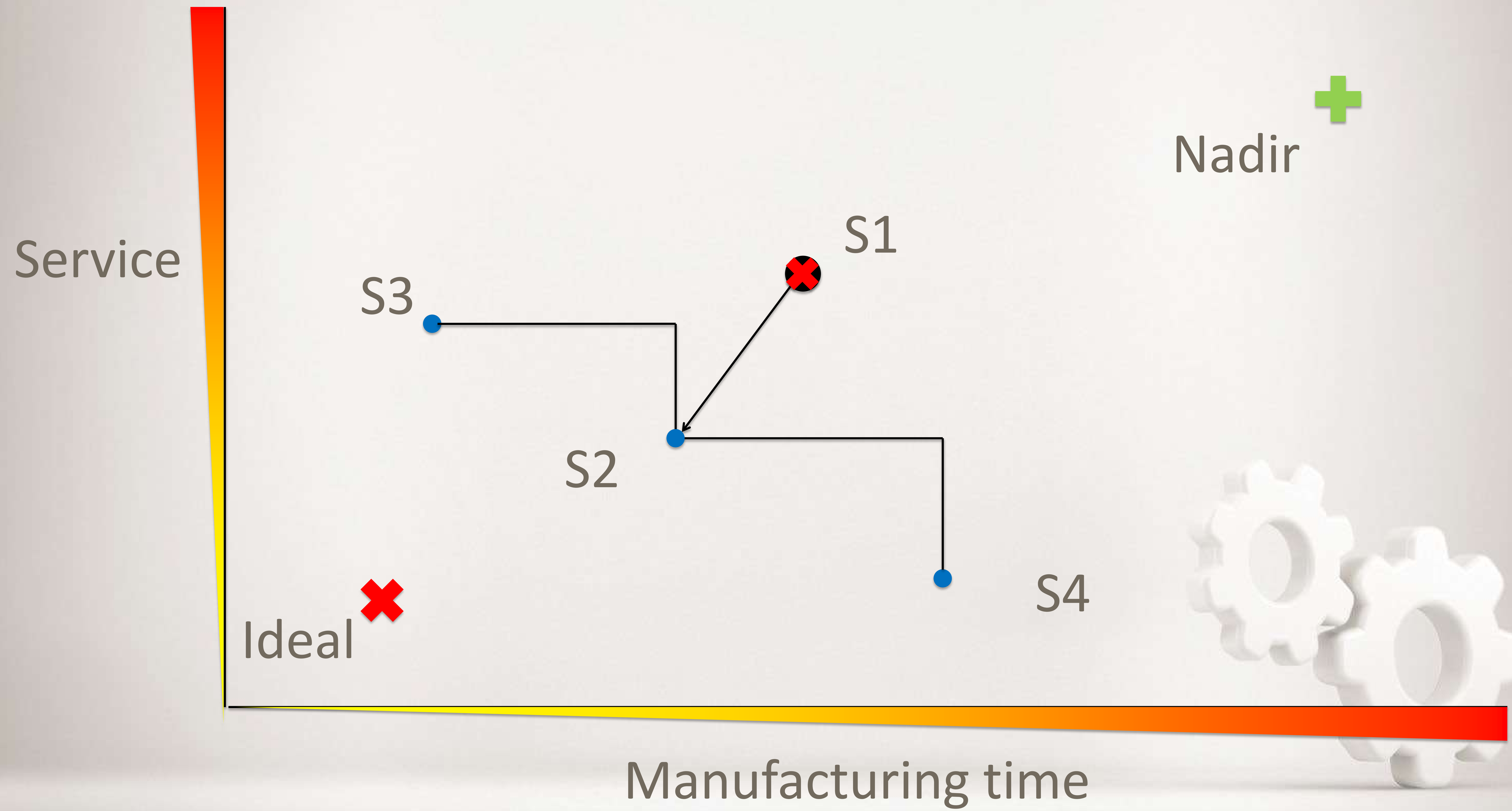
Pareto front concept

In multiobjective optimization we mainly seek:

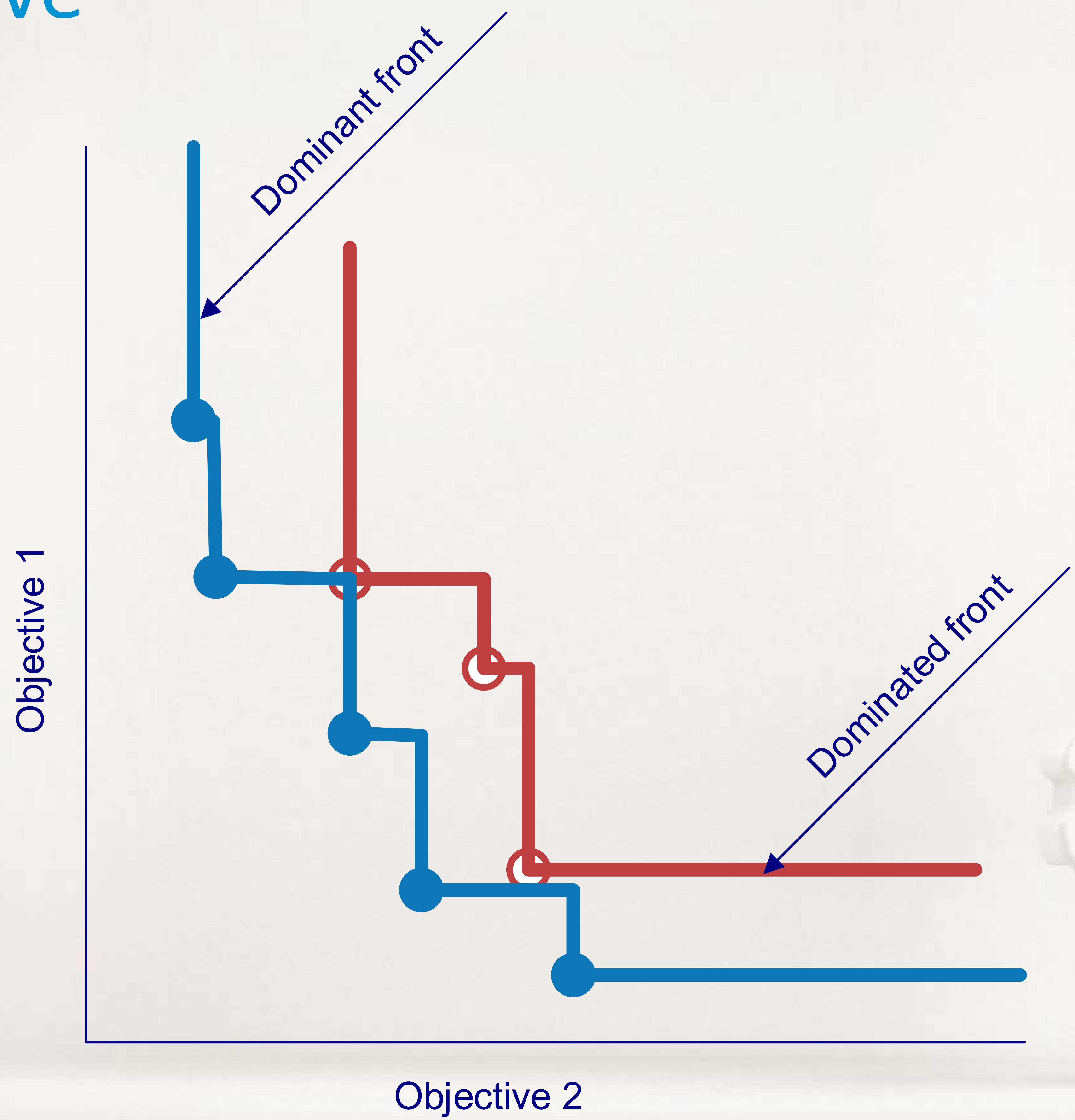
- Good approximation of the ideal Pareto front
- Good coverage of both objectives

To demonstrate that a method A gives better results than a method B is far from trivial

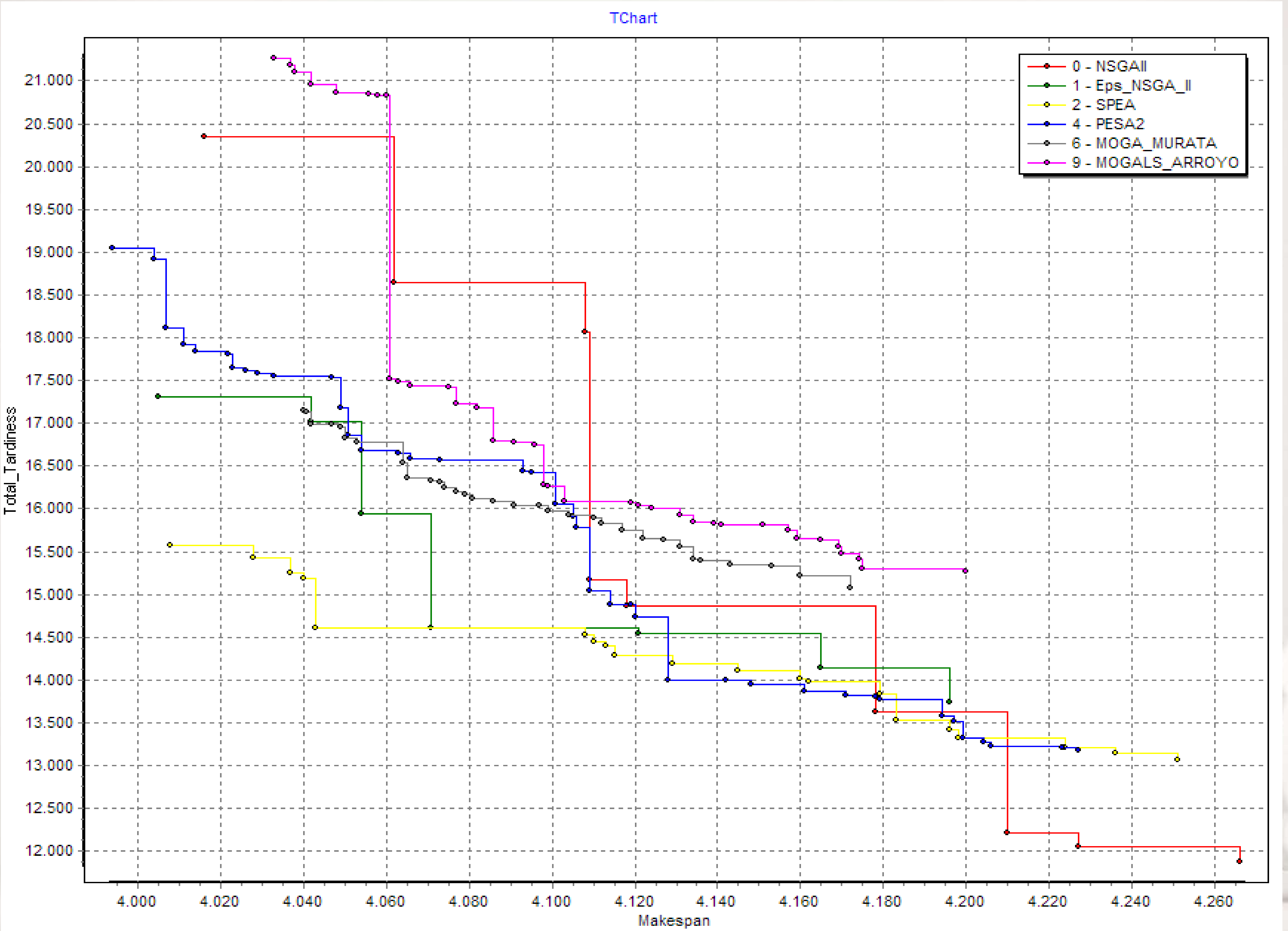
Multiobjective



Multiobjective



Multiobjective



Multiobjective. *Iterated Greedy* Algorithm

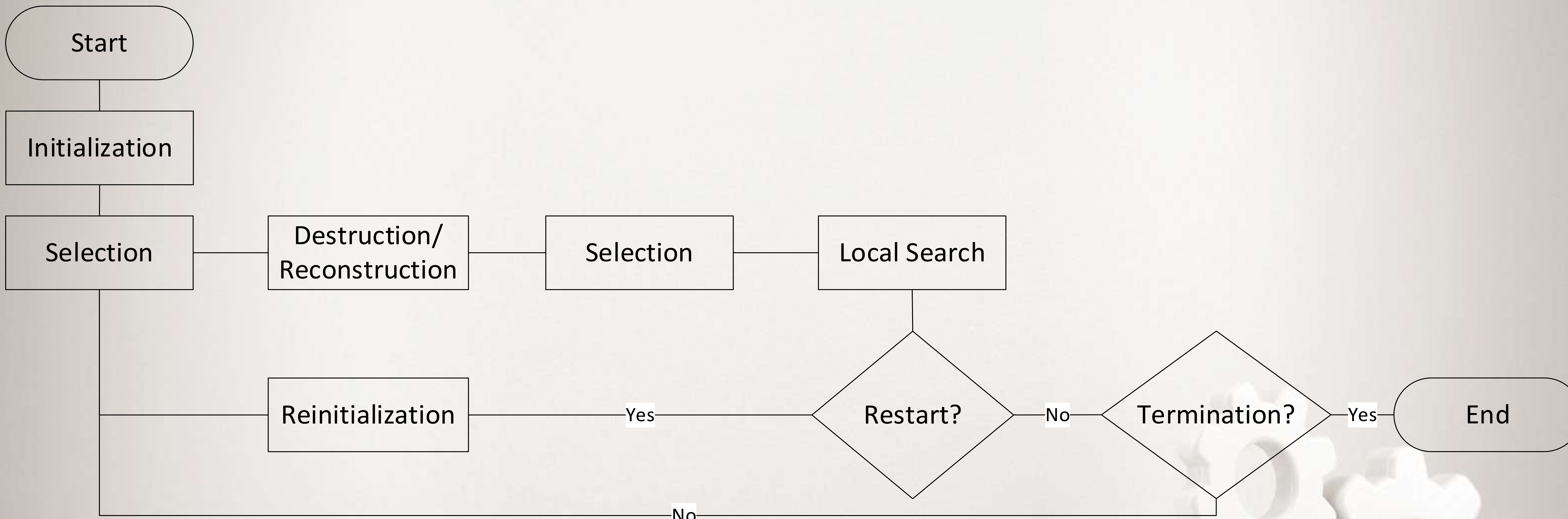
We work with an external archive of non-dominated solutions

We need to select one solution from the archive at each iteration of the IG

Several solutions generated at the reconstruction

At the end of each generation we add the solutions to the archive and eliminate the dominated ones

Multiobjective. *Iterated Greedy* Algorithm



Multiobjective. *Iterated Greedy* Algorithm

Initialization: 4 good solutions, 2 per objective

Selection: Modification of the *Crowding Operator* of Deb (2002)

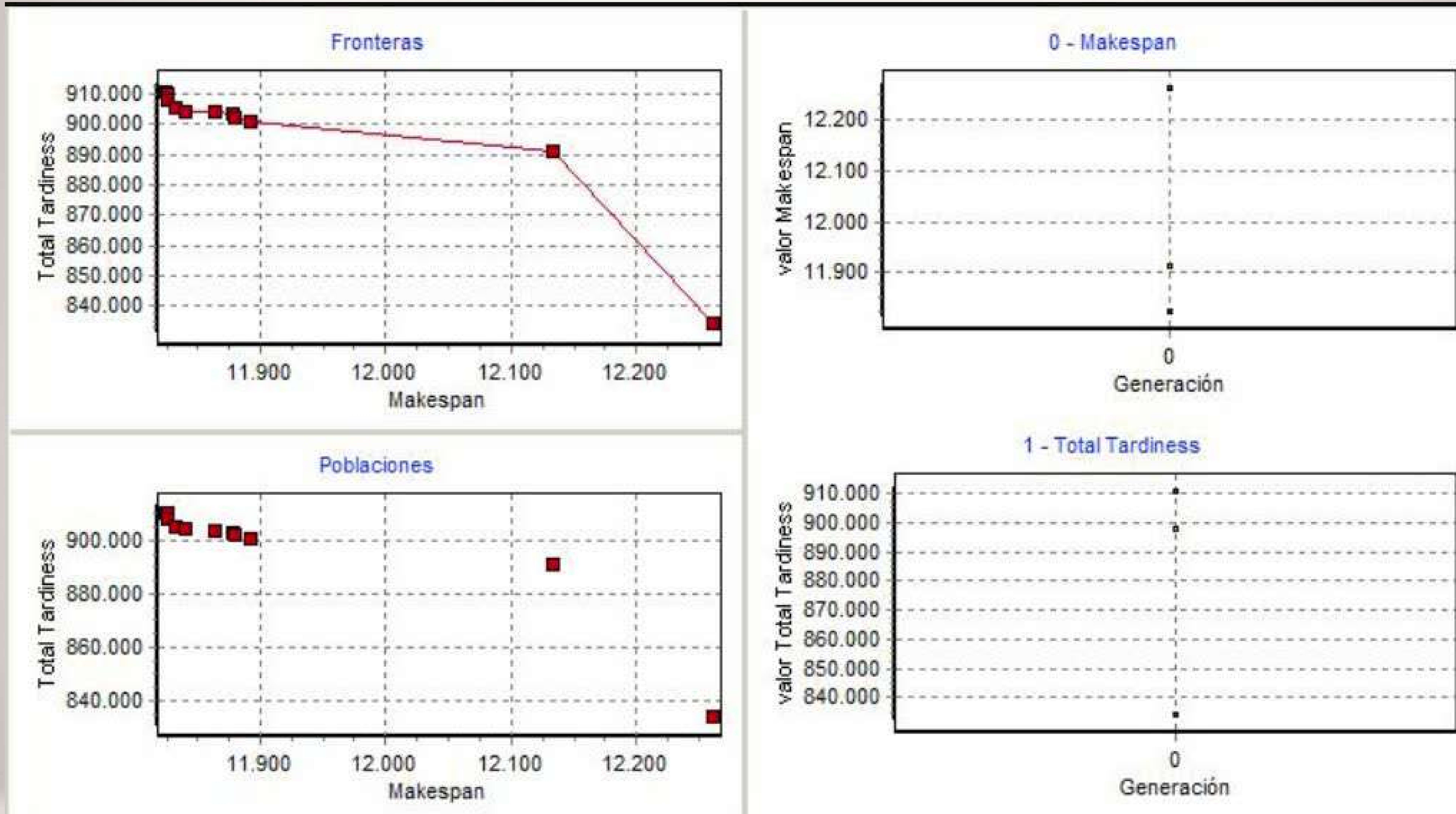
Solutions that are isolated and seldom selected are favored

Destruction by blocks of consecutive jobs

Reconstruction by insertion, saving partial non-dominated solutions

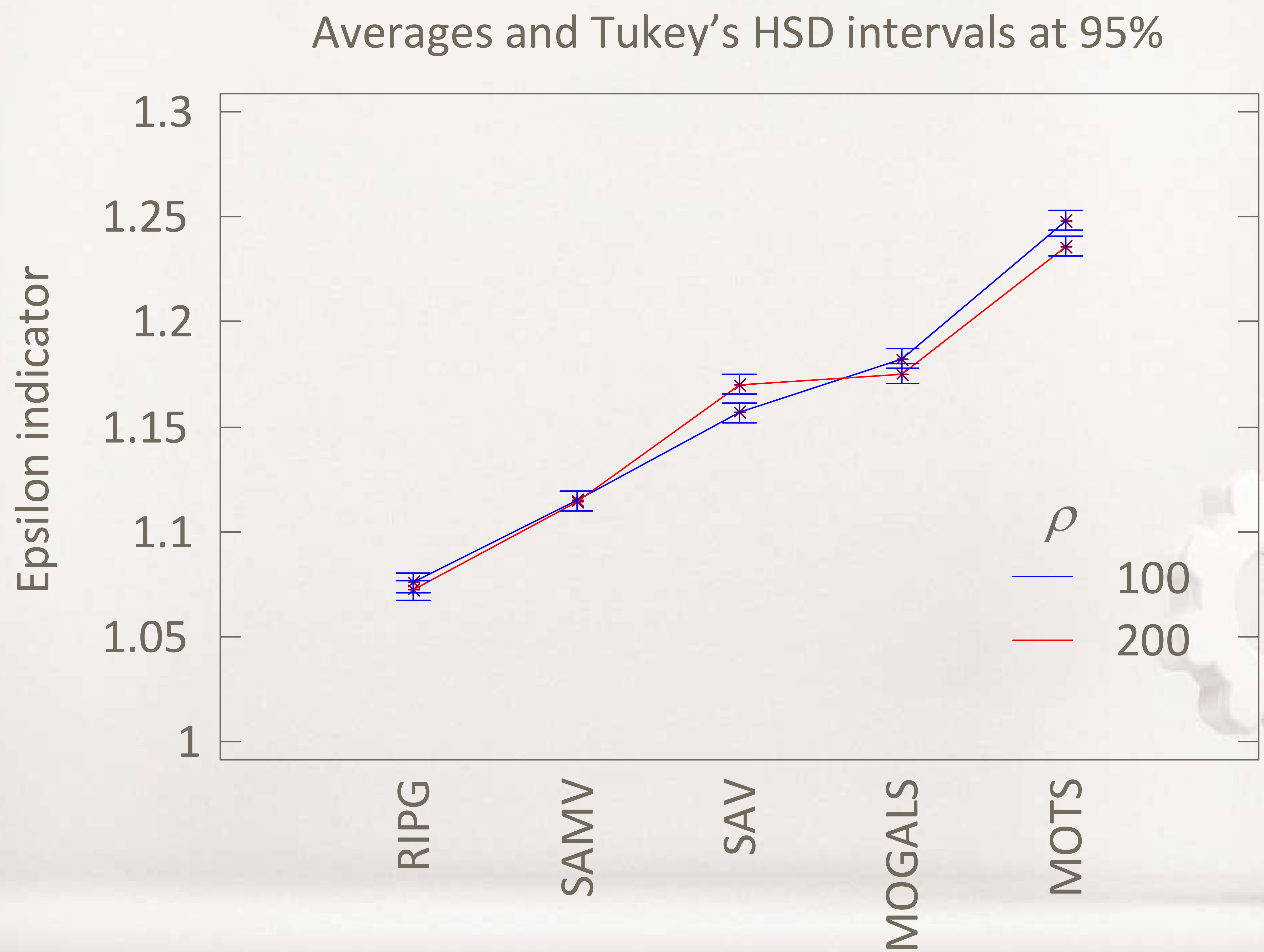
Reinitialization after a number of iterations without improvements

Multiobjective. *Iterated Greedy* Algorithm



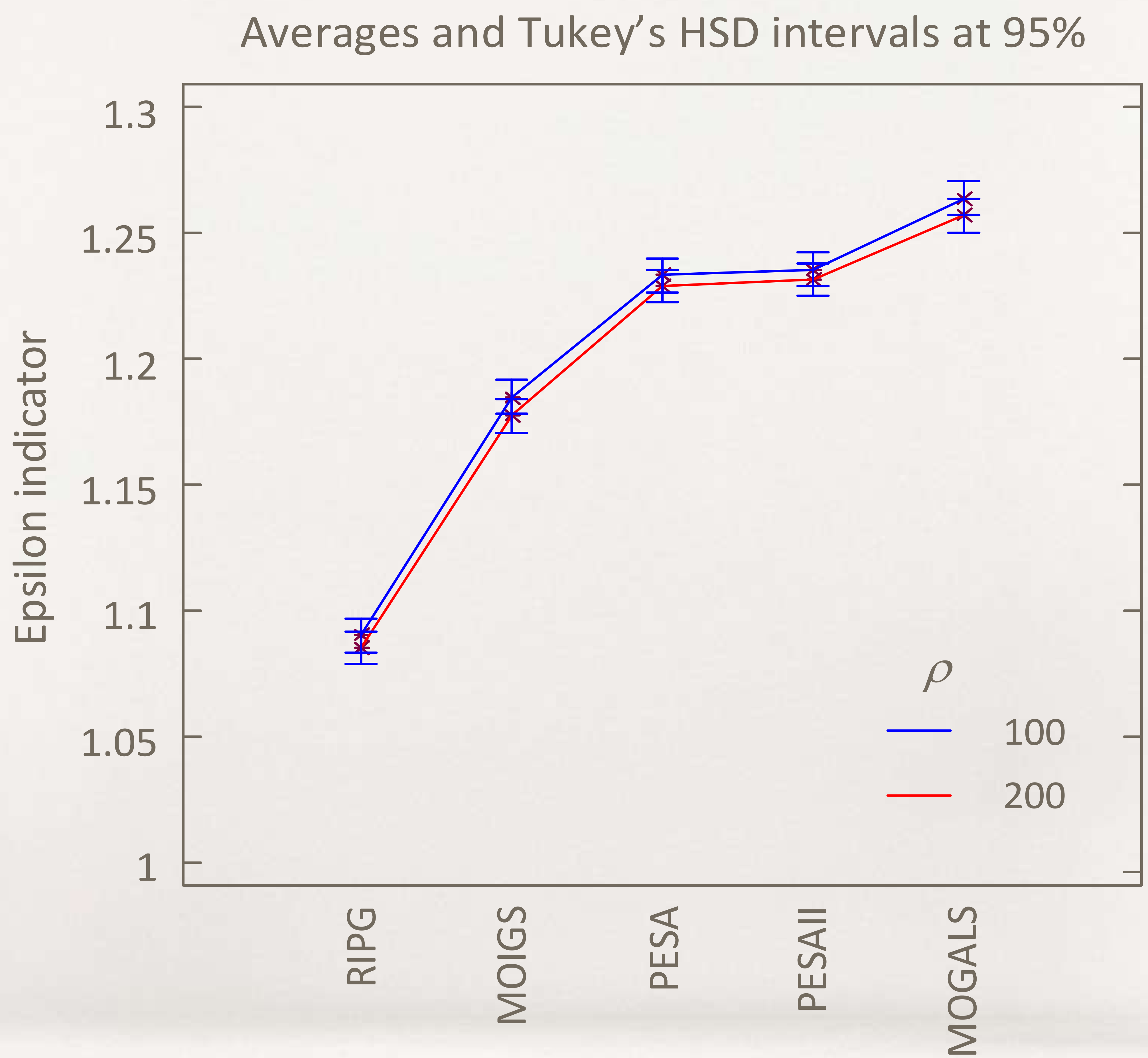
Multiobjective. Evaluation

Makespan and total flowtime without setup times

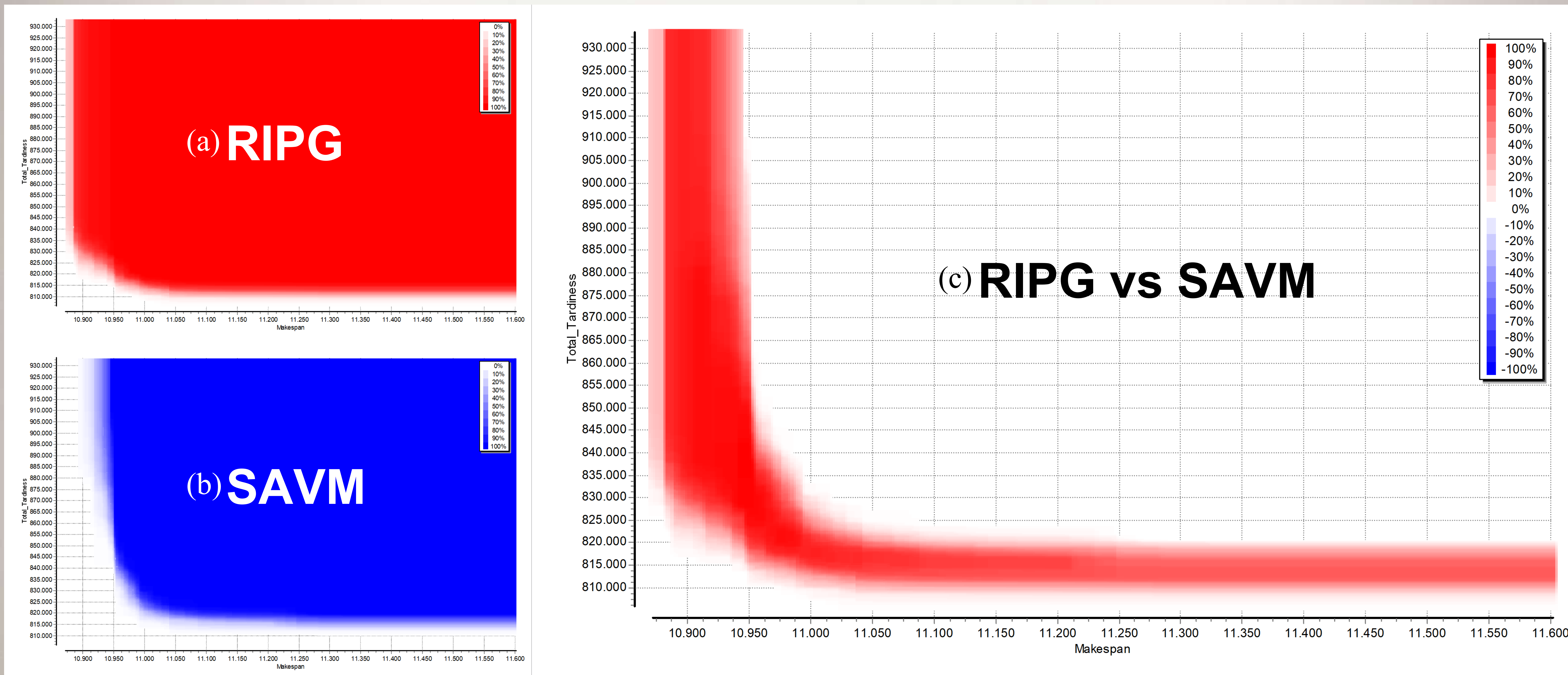


Multiobjective. Evaluation

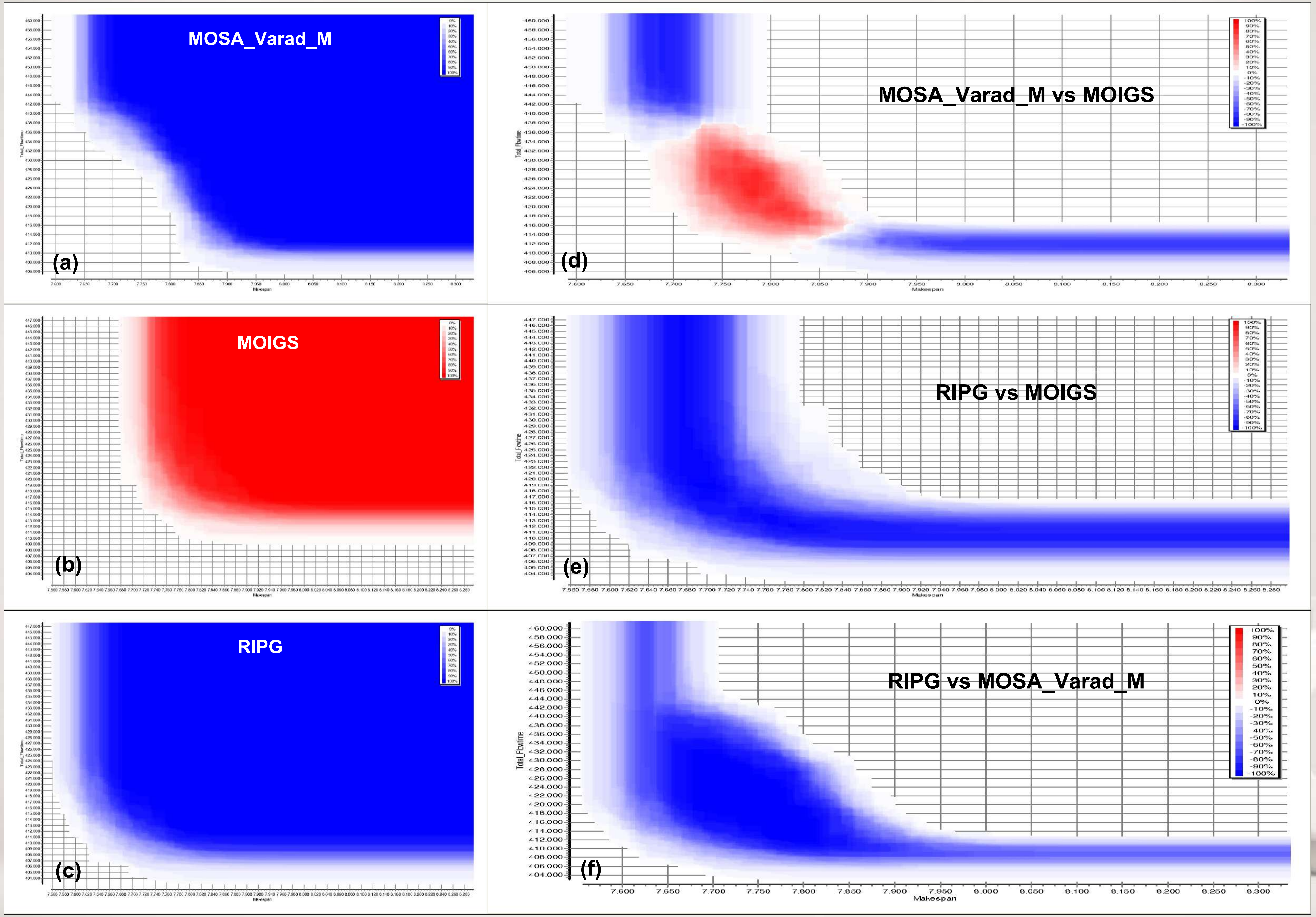
Makespan and weighted tardiness WITH setup times



Multiobjective. Evaluation



Multiobjective. Evaluation



Multiobjective. Evaluation

Multiobjective problems are much more difficult

But we do not need complex techniques to solve them

IG method can be adapted, with little changes, to multiobjective flowshop problems

Results much better than competing methods with and without setups

Multiobjective

References

G. Minella, R. Ruiz and M. Ciavotta (2011). Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems, *Computers & Operations Research*, 38(11): 1521-1533.

M. Ciavotta, G. Minella and R. Ruiz y (2013). Multi-objective sequence dependent setup times flowshop scheduling: a new algorithm and a comprehensive study. *European Journal of Operational Research*, 227(2): 301-313.

8. Distributed scheduling

Scientific literature assumes that there is only one production shop or factory where jobs are processed

Nowadays, single-factory enterprises are uncommon. Supply Chains are an example of multiple factory systems that have been shown to be very efficient (Moon et al., 2002)

Distributed scheduling

F identical factories where jobs can be processed

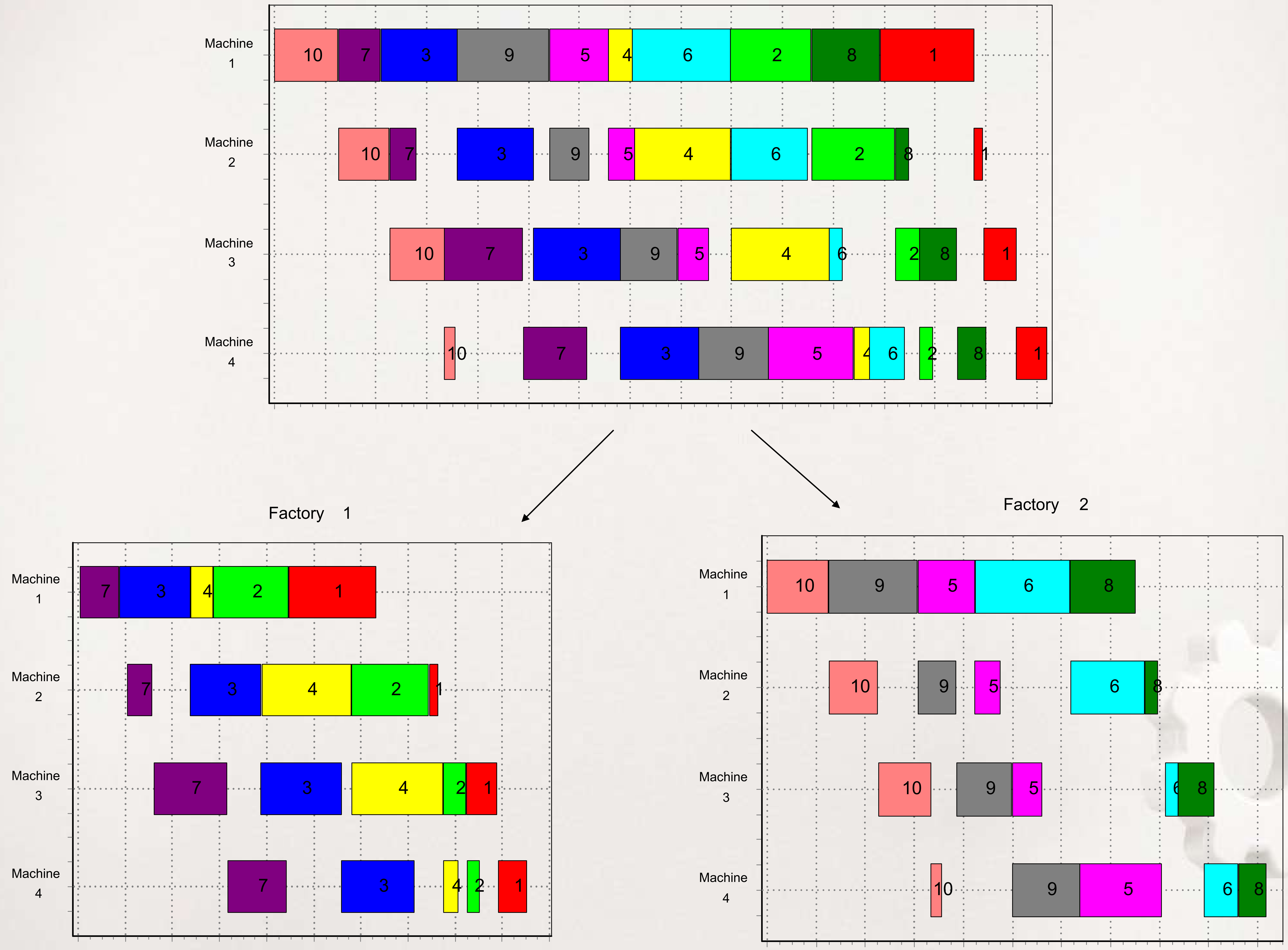
We have two interrelated decisions: assignment of jobs to factories and sequencing of the assigned jobs at each factory

Obviously, at each factory, the sequencing problem depends on the jobs assigned

Objective: to minimize the maximum makespan among the F factories

This problem is also NP-Hard if n>>F

Distributed scheduling



Distributed scheduling

Naderi and Ruiz (2010) “The distributed permutation flowshop scheduling problem” Computers & Operations Research 37(4):754-768. The first work about the multi factory flowshop

Naderi and Ruiz (2014) “Scatter search algorithm for the distributed permutation flowshop scheduling problem” European Journal of Operational Research, 239(2), 323-334. 30+ methods proposed, a comparison with the best 11

Distributed scheduling

Existing IG methods and variants focus on small changes:

procedure Iterated_Greedy
$\pi_0 :=$ GenerateInitialSolution
$\pi :=$ LocalSearch(π_0)
while (termination criterion not satisfied) **do**
$\pi_D :=$ Destruction(π)
$\pi' :=$ Reconstruction(π_D)
$\pi'' :=$ LocalSearch(π')
$\pi :=$ AcceptanceCriterion(π'', π)
endwhile
end

Distributed scheduling

Algorithm POIs:

- Solution representation
- Initial solution
- Destruction
- Reconstruction
- Local search
- Acceptance criterion

Distributed scheduling

Solution representation

The permutation of n jobs is divided among the F factories. Therefore there is an array of F lists, one per factory, each containing a partial permutation of jobs

It is the same representation used by Naderi and Ruiz (2010, 2014) and most of the published research

Distributed scheduling

Initial solution

- Most papers employ NEH (Nawaz et al., 1983) variants
- We carry out limited reinsertions of adjacent jobs in the NEH, adapting previous results of Rad et al. (2009) and Pan and Ruiz (2014)

Distributed scheduling

procedure NEH2_en
  Calculate $P_j = \sum_{i=1}^{m} p_{ij}, \forall n \in N$
  π^{LPT} := Sort the n jobs according to P_j in decreasing order
  for $f := 1$ **to** F **do** $\pi_f := \varnothing$ % (empty initial solution)
  $\pi := \{\pi_1, \pi_2, \ldots, \pi_F\}$
  for $step := 1$ **to** n **do**
    $j := \pi^{LPT}[step]$
    for $f := 1$ **to** F **do**
      Test job j in all possible positions of π_f % (Taillard's accelerations)
      $C^f_{\max}$ is the lowest $C_{\max}$ obtained
      p^f is the position where the lowest $C_{\max}$ is obtained
    endfor
    $f_{\min} = \arg\left(\min_{f=1}^{F}(C^f_{\max})\right)$
    Insert job j in $\pi_{f_{\min}}$ at position $p^{f_{\min}}$ resulting in the lowest $C_{\max}$
    Extract at random job h from position $p^{f_{\min}} - 1$ or $p^{f_{\min}} + 1$ from $\pi_{f_{\min}}$
    Test job h in all possible positions of $\pi_{f_{\min}}$ % (Taillard's accelerations)
    Insert job h in $\pi_{f_{\min}}$ at the position resulting in the lowest $C_{\max}$
  endfor
end

Distributed scheduling

Destruction Three operators tested:

1. Original of Ruiz and Stützle (2007) (same used by Fernandez-Viagas and Framinan, 2015)

2. Biased operator of the IG of Lin et al. (2013). This is a rather complex destruction method.

3. New simple operator. Same as Ruiz and Stützle (2007) where d/2 jobs are removed from the factory generating the Cmax and the others at random from the other factories

Distributed scheduling

Reconstruction

- Similar to the two existing IGs from the literature (IG Lin et al, 2013, BSIG Fernandez-Viagas and Framinan, 2015)
- Just adding reinsertions of the adjacent job at random, much like in the NEH2_en

Distributed scheduling

Local search. Most important step in the IG

Three operators tested:

1. LS_1 and LS_2 in a VND loop (VND(a) of Naderi and Ruiz, 2010)

2. Same as 1 but the job to extract at each step is selected at random, without repetition. This adds diversification in the local search (VND(a)R)

3. A new Local search with incorporates LS_1 and LS_2 in the same loop, without VND. It is driven by the factory generating the Cmax

Distributed scheduling

procedure LS3($\pi = \{\pi_1, \pi_2, \ldots, \pi_F\}$)
  $C^*_{\max} = \max_{f=1}^{F}\{C_{\max}(\pi_1), C_{\max}(\pi_2), \ldots, C_{\max}(\pi_F)\}$
  $f_{\max} = \arg(C^*_{\max})$ % (factory with the largest $C_{\max}$)
  $Cnt := 0$
  while $Cnt < |\pi_{f_{\max}}|$ **do** % (all jobs in factory $f_{\max}$)
    Randomly extract, without repetition, a job j from position k of $\pi_{f_{\max}}$
    for $f := 1$ **to** F **do**
      Test job j in all possible positions of π_f % (Taillard's accelerations)
      $C^f_{\max}$ is the lowest $C_{\max}$ obtained
      p^f is the position where the lowest $C_{\max}$ is obtained
    endfor
    $f_{\min} = \arg\left(\min_{f=1}^{F}(C^f_{\max})\right)$
    if $C^f_{\max} < C^*_{\max}$ **then**
      Place job j at position p^f of factory $f_{\min}$
      $C^*_{\max} = \max_{f=1}^{F}\{C_{\max}(\pi_1), C_{\max}(\pi_2), \ldots, C_{\max}(\pi_F)\}$
      $f_{\max} = \arg(C^*_{\max})$ % (factory with the largest $C_{\max}$)
      $Cnt := 0$
    elseif
      Return job j to position k of $f_{\max}$
      $Cnt := Cnt + 1$
    endif
  endwhile
end

Distributed scheduling

Acceptance criterion

Recent developments in parameter-less procedures (Hatami et al, 2013, Hatami et al., 2015)

Did not yield very good results

We stick to the original criterion of Ruiz and Stützle (2007) which is SA-like with a constant temperature

Distributed scheduling

- Nested/Two stage Iterated Greedy
  - Nested IG: Replace the LS by another IG
    - An IG inside another IG
- Two stage IG
  - Two IGs working one after the other, usually focusing on different neighborhoods

Distributed scheduling

Nested/Two stage Iterated Greedy focusing only on the factory generating the Cmax

Very similar to the original IG of Ruiz and Stützle but with descent acceptance criterion

Nested variants did not yield good results

Two stage better results. Start second stage after a given proportion ρ of the CPU time has elapsed

Distributed scheduling

Set of 720 instances of Naderi and Ruiz (2010):

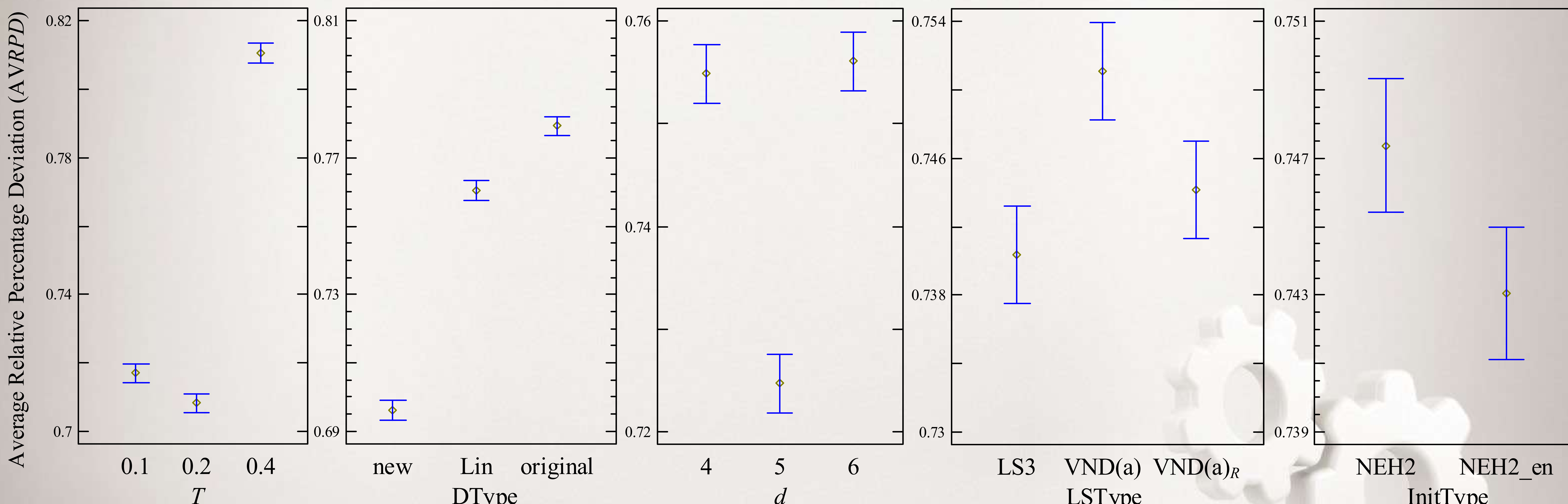
- 120 instances of Taillard (1993) with different number of factories, from 2 to 7
- 50 different random instances for calibration

Average relative percentage deviation from best solution known (RPD)

5 different stopping times: $t=n \cdot m \cdot C$ milliseconds where C is tested at 5 levels: 20, 40, 60, 80 and 100

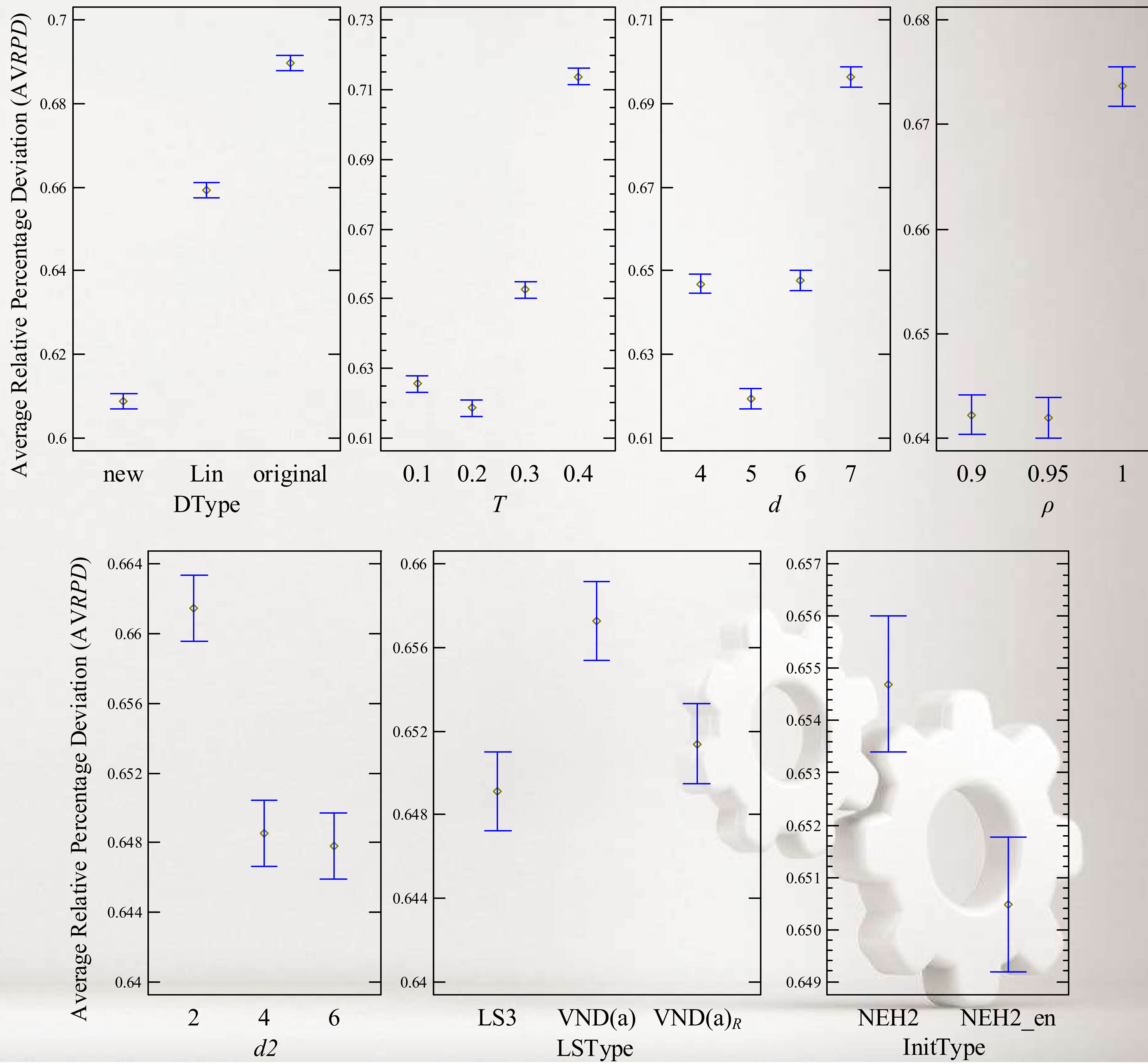
Distributed scheduling

Calibration IG1S by means of DOE+ANOVA:



Distributed scheduling

IGS2 calibration:



Distributed scheduling

We test the following methods:

1. Proposed single stage IG (IG1S)
2. IG1S version with VND(a)R local search and regular NEH_en initialization (IG1S^{-})
3. Proposed two stage IG (IG2S)
4. The hybrid immune algorithm of Xu et al (2014) (HIA)
5. The Scatter search of Naderi and Ruiz (2014) (SS)
6. The bounded IG of Fernandez-Viagas and Framinan (2015) (BSIG)

Distributed scheduling

All methods recoded in C++, Visual Studio 2015 X64 all optimization flags enabled

Experiments performed on virtual machines with 2 virtual processing cores and 8 GBytes of RAM running Windows 10 Enterprise 64 bits operating system. 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 or RAM, for a total of 576 cores and 3 TBytes of RAM

No parallel computing, just a random distribution of all computations among the virtual machines

Distributed scheduling

6 methods

720 instances

10 replicates per instance

5 independent executions with different stopping times (C=20, 40, 60, 80, 100)

A total of 216,000 results and almost 6,600 CPU hours to complete the experiments

Same language, many shared functions, same computers, identical CPU time stopping criterion

Apples to apples comparison

Distributed scheduling

Average Relative Deviation from Best Known Solution

C	HIA	SS	BSIG	IG1S⁻	IG1S	IG2S
20	10.54	1.80	0.97	0.66	0.62	0.60
40	10.06	1.64	0.83	[illegible]	0.47	0.45
60	9.78	1.55	0.77	0.43	0.39	0.37
80	9.58	1.49	0.72	0.37	0.33	0.32
100	9.37	1.45	0.69	[illegible]	0.29	[illegible]
Average	9.87	1.59	0.80	[illegible]	0.42	[illegible]

37% lower

58% lower

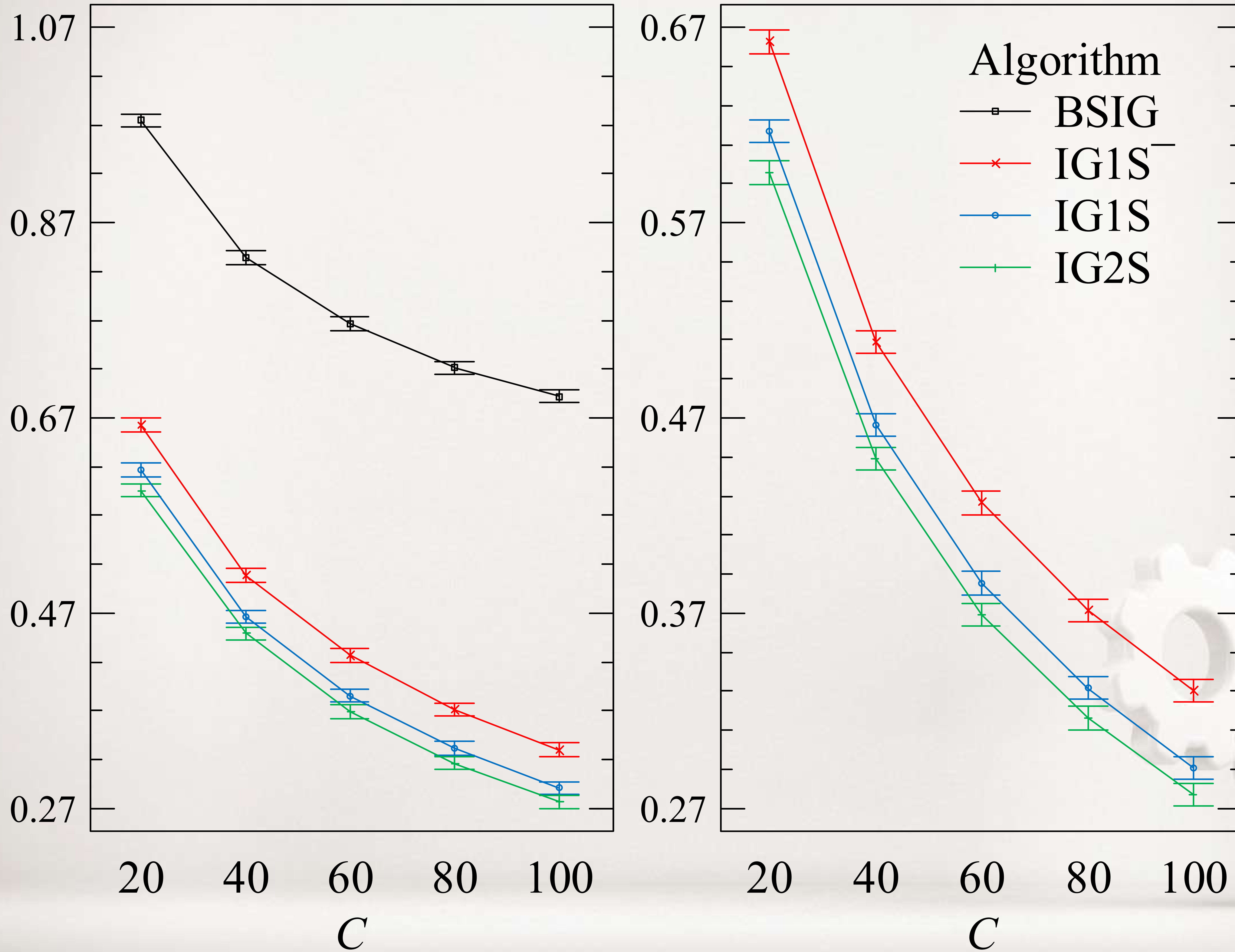
50% lower

4% lower

47% lower

lower

Distributed scheduling



Distributed scheduling

497 out of 720 new upper bounds for the problem

IG1S /IG2S able to get statistically better results than BSIG in 1/5th of the CPU time

New state-of-the-art

No need of bizarre nature-inspired methods or metaphor-based procedures

Just simple IG with more intensification and randomization in the search and a good implementation

9. Conclusions

IG is basically the iteration of a constructive greedy heuristic

Many similarities with GRASP and ILS

We have seen different problems and examples. The pattern is clear: the simpler, the better

Metaheuristics do not have to be complex to yield good results

Advantages

Usually very few parameters to calibrate

Very fast and small memory footprint

Does not use problem specific knowledge

Very easy to implement

Easy to extend to other problems and objectives

Almost always state-of-the-art results

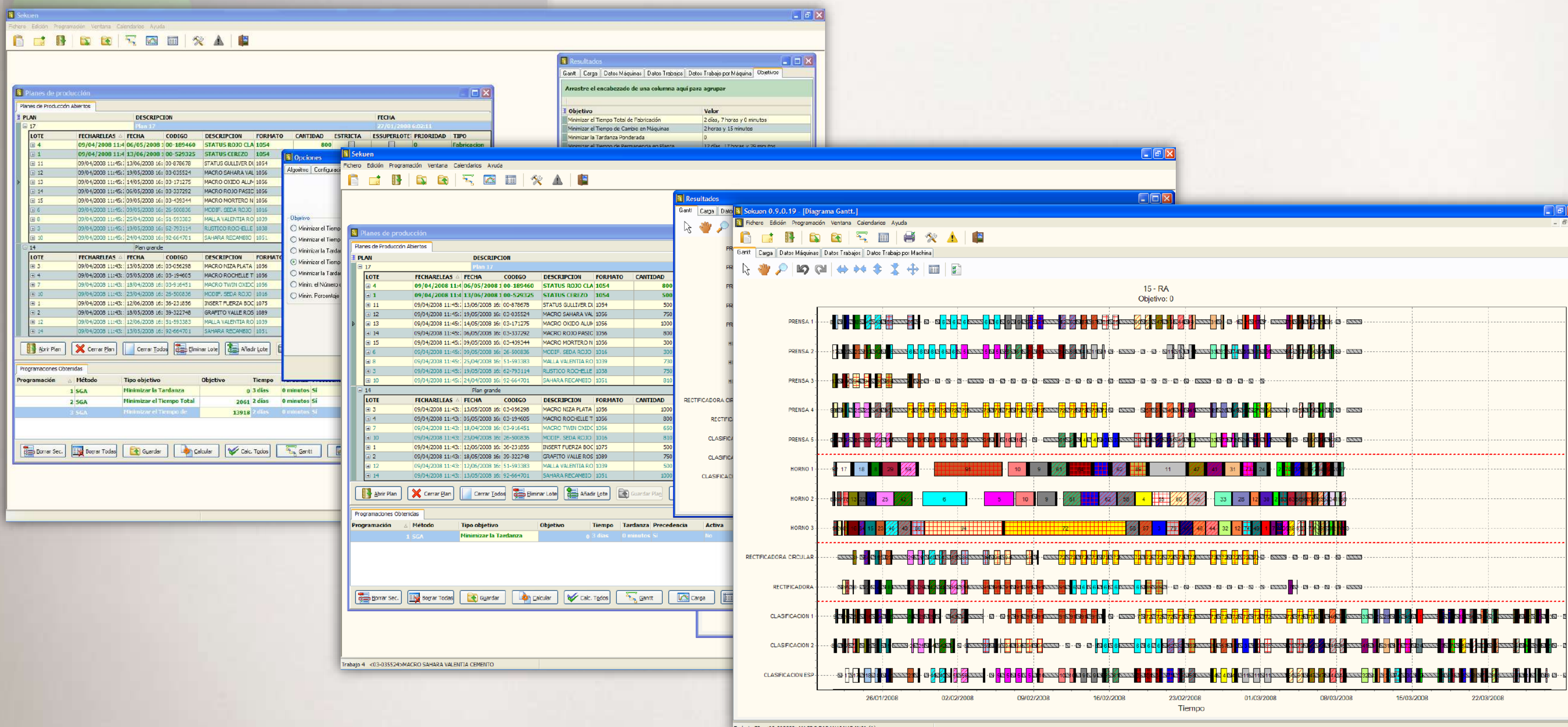
Drawbacks

Not competitive if there is no good heuristic to start and to base on

Not competitive is solutions are very expensive to evaluate

Very hard to convince referees that simple methods yield better results than complex and exotic metaheuristics

IG in practice



Algoritmos Iterados Golosos: fundamentos, aplicaciones y resultados

Rubén Ruiz

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SISTEMAS DE OPTIMIZACIÓN APLICADA