Embarrassingly Parallel Search

Jean-Charles Régin
Université Nice-Sophia Antipolis,
I3S UMR 7271, CNRS, France
jcregin@gmail.com

Joint work with **M. Rezgui** (UNS) and **A. Malapert** (UNS)

Supported by the PAJERO project

Question

- We have a problem P to solve
- We have unlimited resources but they cost money
- We have a limited resolution time

- Question : **how can we solve P for the minimum cost while respecting the resolution time?**

Question

- How can we use the cloud computing to solve optimization problems ?

- How can we estimate the resolution time of a problem ?

Cloud computing uses

- **Problems**
  - MIP solvers work badly in //
  - Heterogeneous machines
  - Time resolutions are not estimated very well

- **Solutions**
  - We use Constraint Programming
  - We propose an algorithm which works well in parallel with heterogeneous machines
  - We try to estimate the time resolution

Plan

- Parallelization of the search of solutions in CP
- Existing techniques
  - Static Decomposition
  - Work stealing
- A new simple and efficient method
- Experimental results

Parallel search of solutions

- We have k workers (CPU, cores, …)
- How can we use the k workers in order to speed up the search of solutions ?

- Hypothesis:
  - If we split a problem into sub-pb then the sum of resolution times of subproblems is equal to the resolution time of the initial problem.
    - In CP, it seems to be right, but not in MIP
    - Be careful with some learning strategies

Static Decomposition

- We have k workers,
  - We split the problem into k subproblems:
    - (x={1,2}) (x={3,4}),(x={5,6}),…
  - We give one subproblem to each worker
- Pros
  - Very simple
  - Not intrusive
- Cons
  - Total time = the time of the longest subproblem
    - Pb with the homogeneity of decomposition

Work stealing

- We have k workers,
  - We split the problem into k subproblems,
  - We give a subproblem for each worker
  - When a worker finishes its work, it asks another worker which works. This latter gives it a part of its remaining work.

- **Pros**
  - Better repartition of the work (dynamic)
- **Cons**
  - Very intrusive in the solver (avoided by the work of B. Le Cun, Bob++)
  - Easy tasks should ve avoided
  - At the end, almost all the workers ask for some work all the time. We need to manage that.

Embarrassingly Parallel Search

- Static decomposition is simple, but it is difficult to split into equal parts

- Solution :
  - We split into more subproblems than workers
- We hope that the sum of resolution times will be equilibrate

- **The greatest importance is not to split into equal parts, but it is to equilibrate the sum of the resolution times of subproblems for each worker**

Embarrassingly Parallel Search

- 2 steps
  - Decomposition
  - Resolution
- **Decomposition**
  - We divide the problem into q subproblems to get a partition of the initial problem (static decomposition)
  - We put these subproblems into a queue
- **Resolution**
  - When a worker needs some work, it takes a supbroblem from the queue. (dynamic choice)

Massive Decomposition

- Decomposition
  - One task requires 140s
  - We divide it into 4 tasks requiring 20,80,20,20s : not well balanced : 80s (max), 20s (min)
  - We split again into 4 parts:
    (5+5+5+5)+(20+10+10+40)+(2+5+10+3)+(2+2+8+8)
  - w1: 5+20+2+8=35; w2: 5+10+2+10=27
    w3: 5+10+5+3+2+8=33; w3=5+40=45 gives : 45s (max) et 27 (min)

- We have more chance to equilibrate the sum of workload for each worker

- We have more chance to break large subproblems and to reduce their relative importance
  - The relative importance of maximum (40 vs 80) is reduced.
    The lost time (max-min) also (80-20=60) vs (45-27=18)

Embarrassingly Parallel Search

- How do we decompose ?
  - We want to split into q subproblems
- Solution 1
  - We take p variables for which the cartesian product of their domains is close to q.
    (we adjust the last domain if needed)
- Results
  - Work very well with some problems
  - Work badly with others, because a lot of generated problems are trivially inconsistent

Decomposition

- Solution 1:
  - We take p variables with which the cartesian product of their domains is close to q.
- If x,y and z are implied in alldiff constraint then
  - the cartesian product is a bad idea. (a,b,a) must not be considered, the same thing for (a,a,b)
  - they are not considered with a sequential resolution

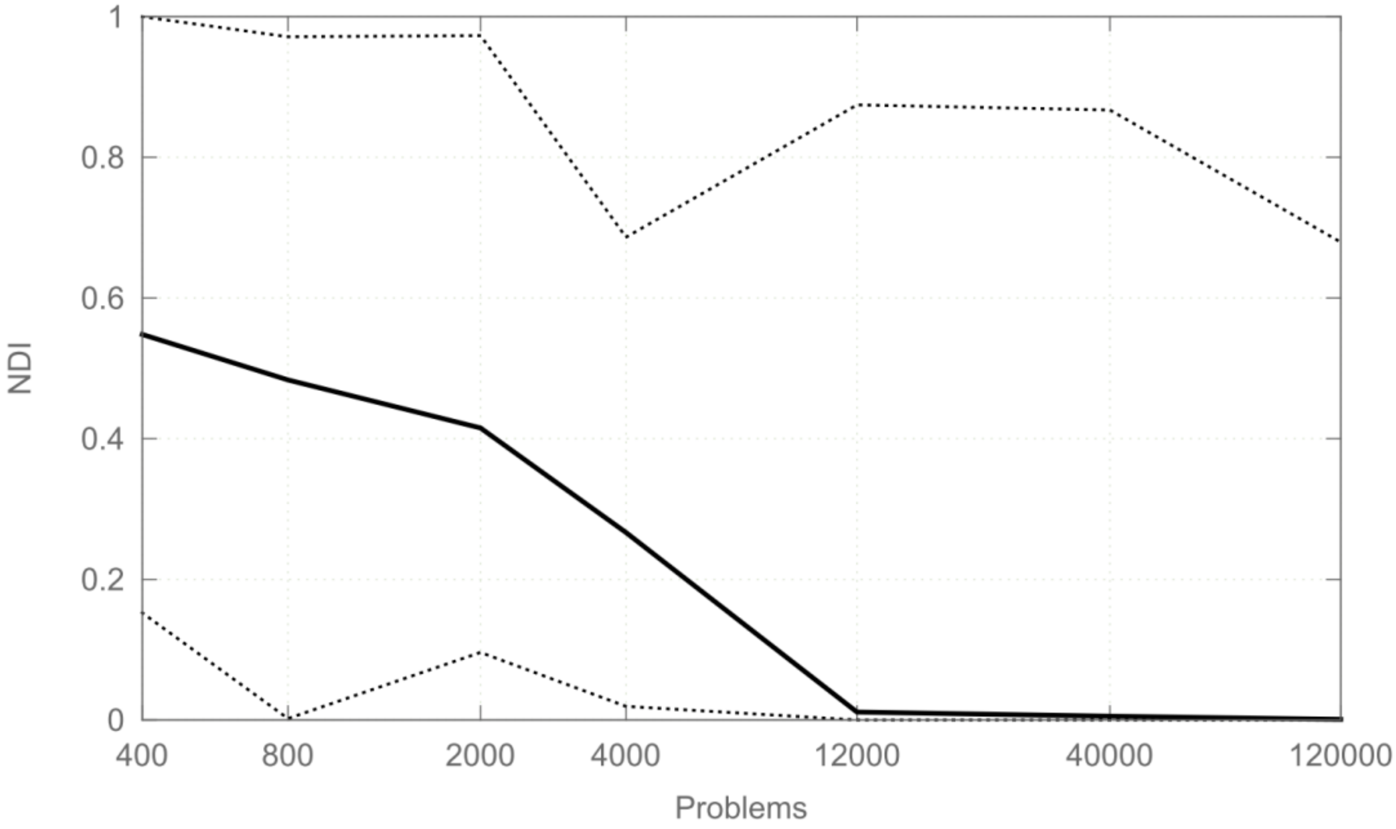
Decomposition

- Solution 1:
  - We take p variables with which the cartesian product of their domains is close to q.

- **We should avoid considering in a parallel resolution , problems that would have not been considered in a sequential resolution**

- **NDI : non detected inconsistent** (= consistent with the propagation)
- This solution generates too many non NDI problems

Decomposition



Jean-Charles Régin, CP 2013

Decomposition

- How do we decompose ?
  - We want to split into q subproblems
- Solution 2
  - We take p variables in order to have a cartesian product close to q.
  - We generate all combinations step by step with eliminating non NDI problems (If the propagation fails then we do not consider the subproblem)
  - If we do not generate the desired number of subproblems then we restart the process with more variables

Decomposition

- To generate all combinations, we simulate a BFS with Bounded DFS
  (fixed number of choiced variables)

- We introduce a table constraint containing combinations for each level to avoid repeating the bad branches between two DFS.

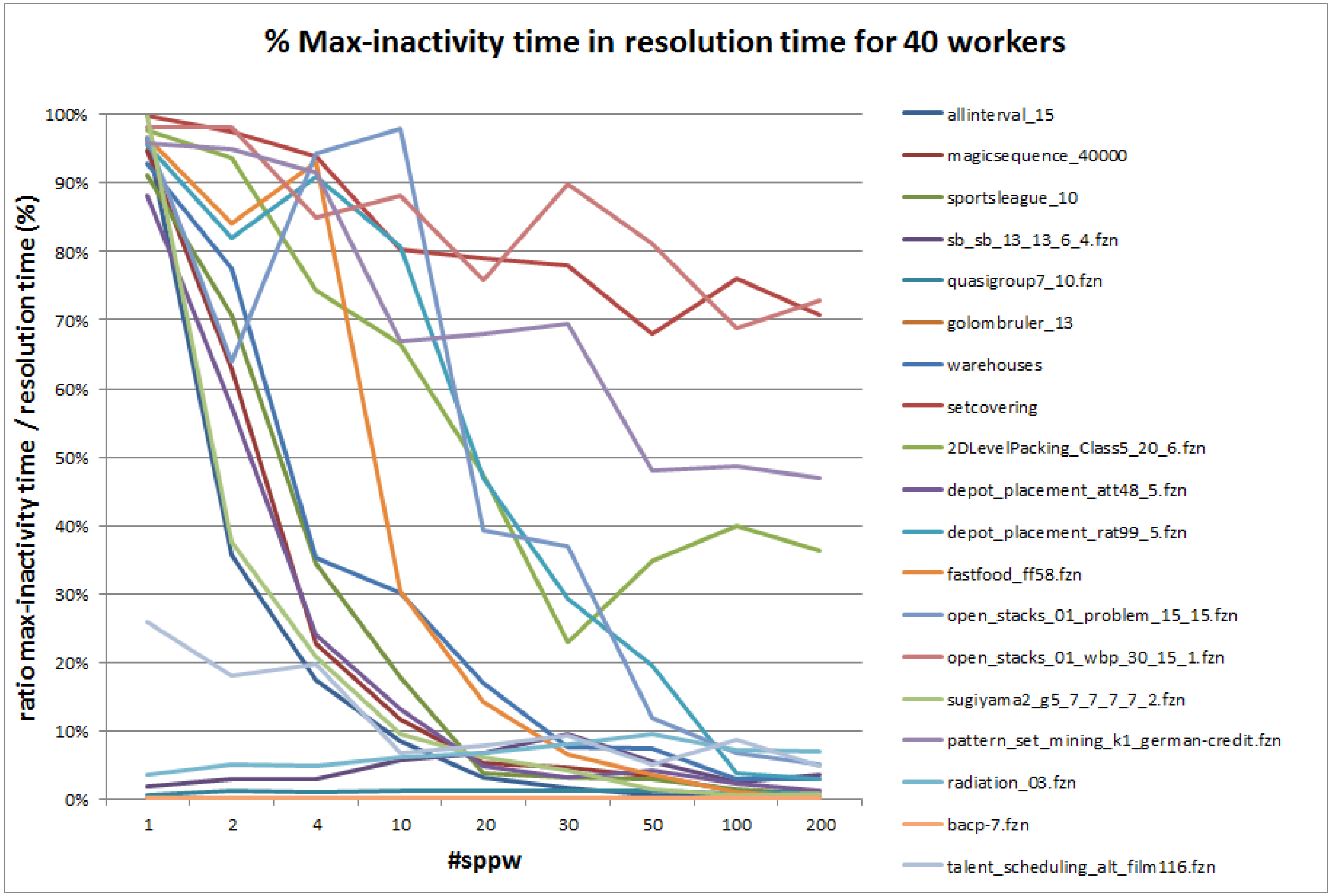
Optimization problems

- The subproblems queue keeps the best current value of the objective
- **A current resolution is never interrupted**
  - Neither to communicate a better solution
  - Or to receive a new value of the objective
- However, when a worker finishes to solve a subproblem, the value of the objective can be used as a better bound

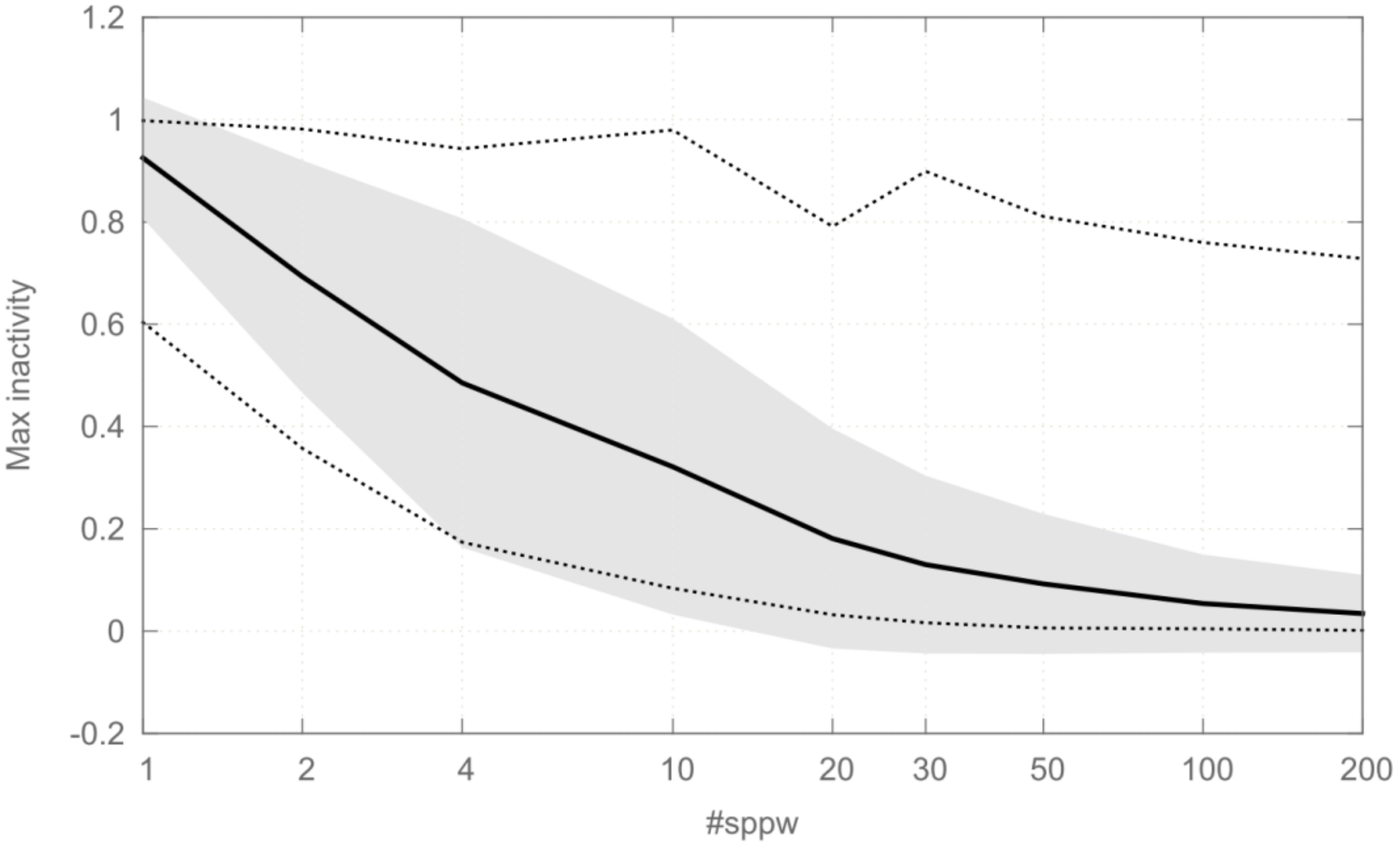
How many subproblems ?

- Tricky question
- As we want to equilibrate the workload of workers,
  we propose to define a number of sub-pb per worker (#sppw)
- According to our experimentations :
  - It appears to be decorrelated the problems !
  - If the value is too small, it is difficult to equilibrate.
  - It the value is too high then the decomposition takes a lot of time
- A value between 30 and 100 subproblems per worker is good.
  The best result is obtained with 30 sub-pb per worker
- The results always include the decomposition time
  (except for precision)
- Means are geometric

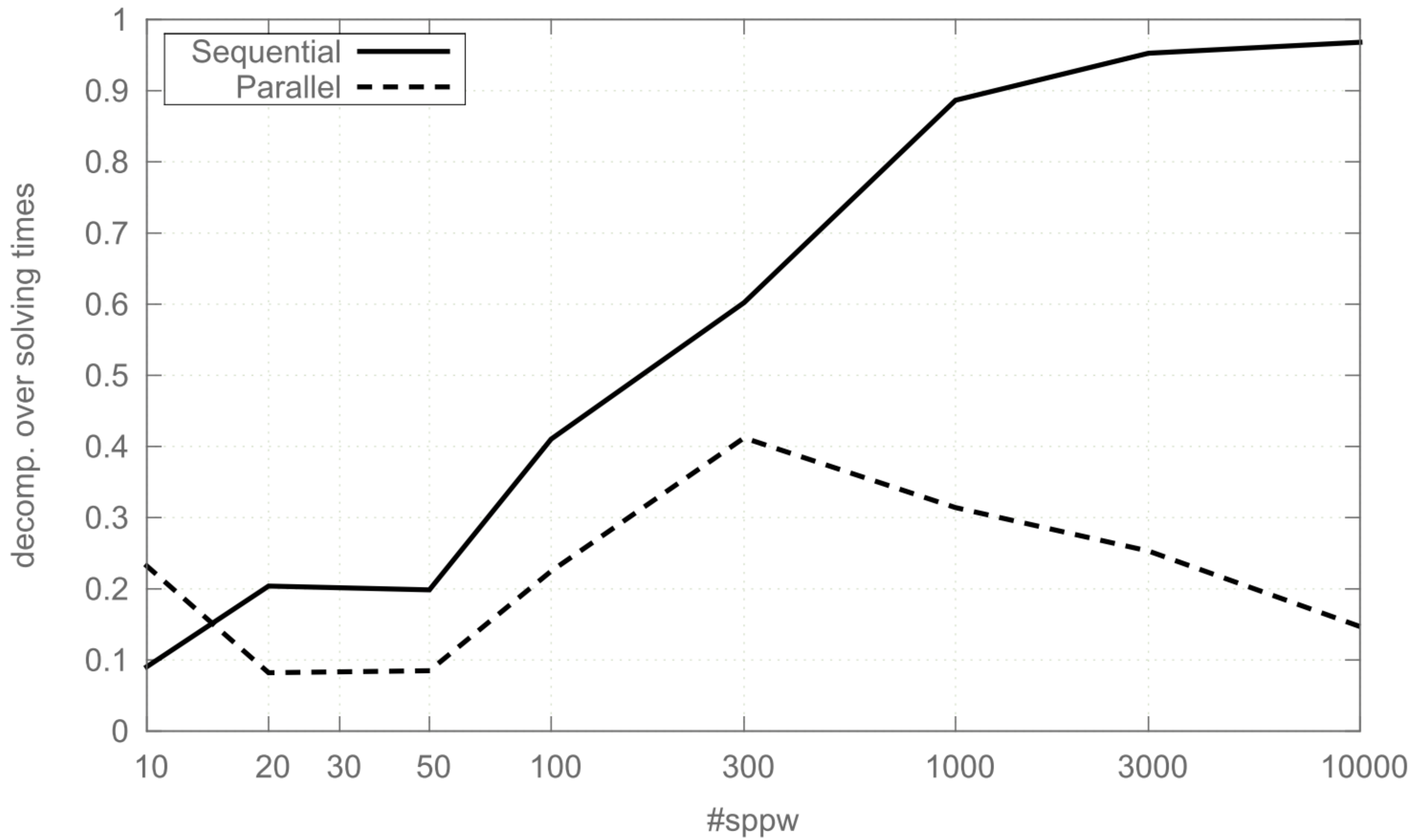
Inactivity time



Inactivity time

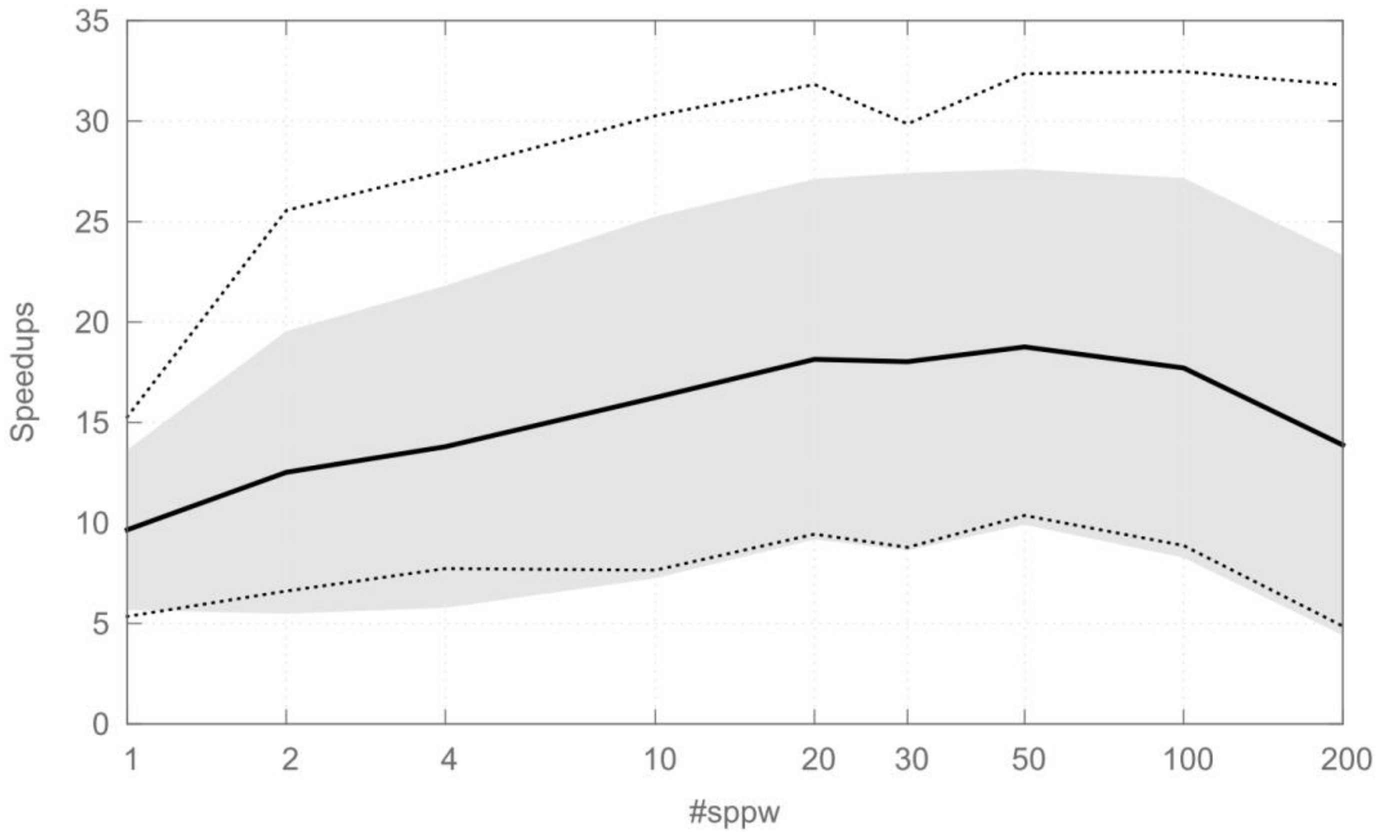


% decomposition time



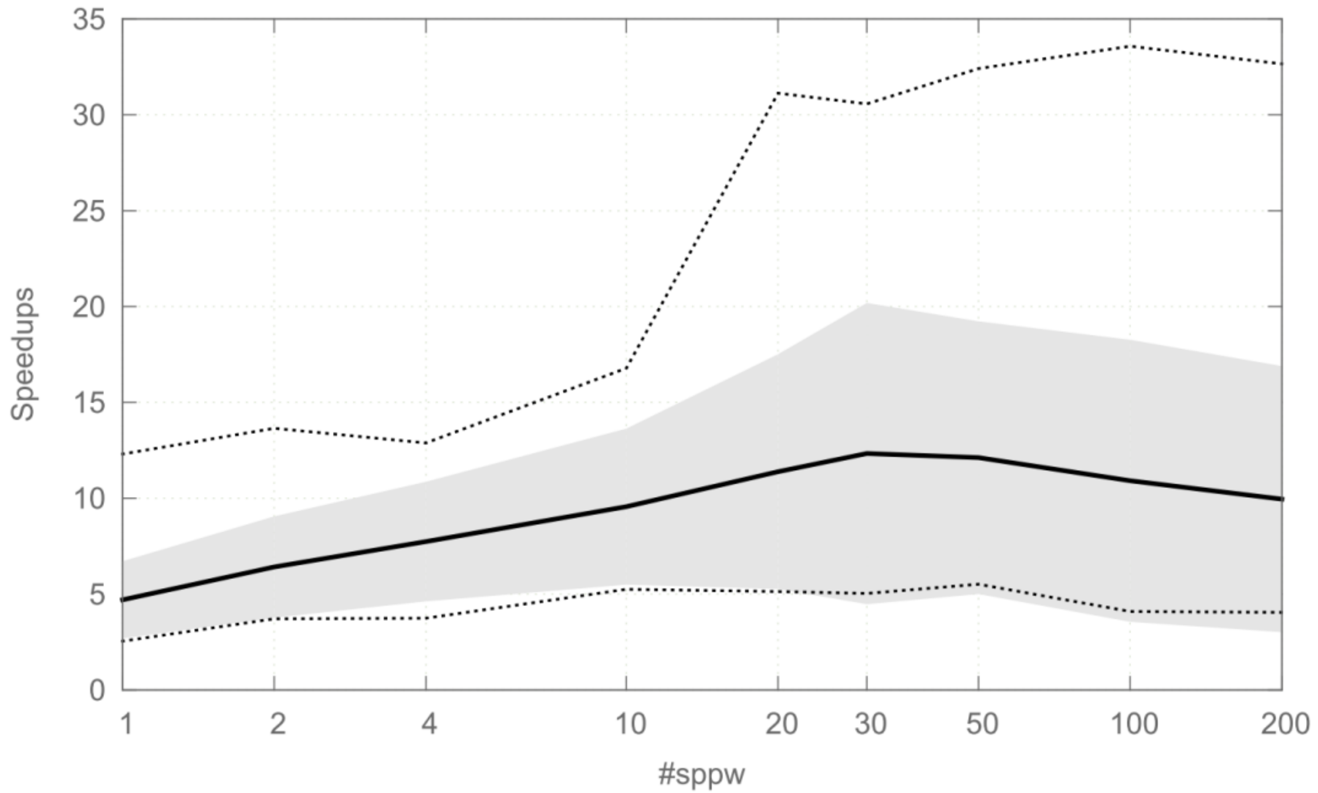
Jean-Charles Régin, CP 2013

Speedup with #sppw



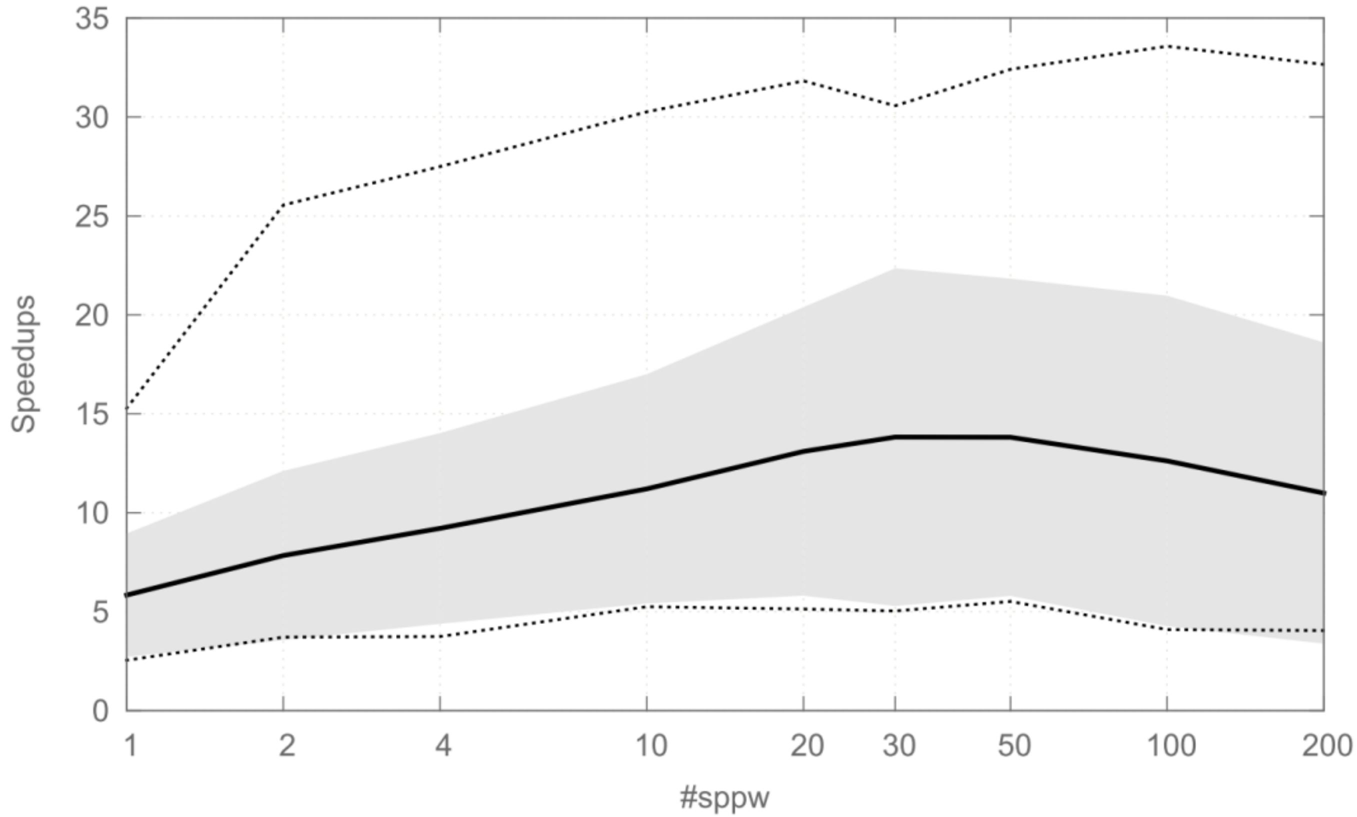
Satisfaction problems

Speedup with #sppw



Optimization problems (with proof)

Speedup with #sppw



Satisfaction problems and optimization problems

Comparison with or-tools

bold for speedup winner **timeout (3600 s)**	Or-tools (rev2555) MapReduceCP		
instance	seq. time (s)	parallel time (s)	speedup
allinterval_15	2169,702	67,685	32,056
magicsequence_40000	timeout	timeout	timeout
sportsleague_10	timeout	timeout	timeout
sb_sb_13_13_6_4.fzn	227,566	18,149	12,539
quasigroup7_10.fzn	timeout	timeout	timeout
non_non_fast_6.fzn	2676,304	310,024	8,633
golombruler_13	16210,218	573,557	28,263
warehouses	timeout	timeout	timeout
setcovering	501,653	33,633	14,916
2DLevelPacking_Class5_20_6.fzn	56,185	3,623	15,508
depot_placement_att48_5.fzn	664,893	13,734	48,412
depot_placement_rat99_5.fzn	67,048	2,824	23,742
fastfood_ff58.fzn	452,438	25,134	18,001
open_stacks_01_problem_15_15.fzn	164,723	7,099	23,204
open_stacks_01_wbp_30_15_1.fzn	164,873	6,345	25,985
sugiyama2_g5_7_7_7_7_2.fzn	298,789	20,491	14,581
pattern_set_mining_k1_german-credit.fzn	270,742	12,818	21,122
radiation_03.fzn	416,624	23,511	17,720
bacp-7.fzn	759,733	23,752	31,986
talent_scheduling_alt_film116.fzn	575,721	15,706	36,656

Comparison with work stealing

bold for speedup winner **timeout (3600 s)**	Gecode 4.0.0	work stealing		MapReduceCP	
instance	seq. time (s)	parallel time (s)	speedup	parallel time (s)	speedup
allinterval_15	262,546	9,724	27,000	8,790	**29,869**
magicsequence_40000	328,194	592,648	0,554	37,334	**8,791**
sportsleague_10	172,390	7,648	22,541	6,782	**25,419**
sb_sb_13_13_6_4.fzn	135,674	9,203	14,742	7,752	**17,502**
quasigroup7_10.fzn	292,589	14,532	20,134	10,542	**27,755**
non_non_fast_6.fzn	602,222	271,254	2,220	56,843	**10,594**
golombruler_13	1355,150	54,889	24,689	44,321	**30,576**
warehouses	148,043	25,896	5,717	21,083	**7,022**
setcovering	94,431	16,117	5,859	11,089	**8,516**
2DLevelPacking_Class5_20_6.fzn	22,612	13,774	1,642	0,748	**30,230**
depot_placement_att48_5.fzn	125,154	19,083	6,558	10,17	**12,306**
depot_placement_rat99_5.fzn	21,628	6,368	3,396	2,613	**8,277**
fastfood_ff58.fzn	23,050	4,520	5,100	3,834	**6,012**
open_stacks_01_problem_15_15.fzn	102,753	6,080	16,900	5,759	**17,842**
open_stacks_01_wbp_30_15_1.fzn	185,715	15,350	12,099	11,219	**16,554**
sugiyama2_g5_7_7_7_7_2.fzn	286,537	22,751	12,594	10,766	**26,615**
pattern_set_mining_k1_german-credit.fzn	113,724	22,317	5,096	13,782	**8,252**
radiation_03.fzn	129,076	33,479	3,855	25,626	**5,037**
bacp-7.fzn	227,152	15,647	14,517	9,517	**23,868**
talent_scheduling_alt_film116.fzn	254,297	13,510	**18,823**	35,625	7,138

Embarrassingly Parallel Search

- In computer science, a problem that is obviously decomposable into many separate subtasks is called **embarrassingly parallel**

- Comes from the french expression “avoir l’embarras du choix”.

- Properties
  - Computation can be easily divided into several independent parts, each part can be executed by a processor.
  - No or very few communication between processus
  - Each process works regardless of others

EPS Advantages

- Simple
- No or very few communication
- Not intrusive in the solver (we just need to get a subproblem and to test the propagation)
- We can easily replay the resolution
  - We just have to save the order of solved problems and the assigned problems to the workers
- Competitive with work stealing