Embarrassingly Parallel Search

Jean-Charles Régis
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

Jean-Charles Régin, CP 2013
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

Jean-Charles Régin, CP 2013
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.

Jean-Charles Régin, CP 2013
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 equilibrated

- 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 supbproblem 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:
    \[(5+5+5+5)+(20+10+10+40)+(2+5+10+3)+(2+2+8+8)\]
  - \[w_1: 5+20+2+8=35; w_2: 5+10+2+10=27\]
  - \[w_3: 5+10+5+3+2+8=33; w_3=5+40=45\] 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
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 \texttt{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 ($\equiv$ consistent with the propagation)

- This solution generates too many non NDI problems.
Decomposition
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

Jean-Charles Régin, CP 2013
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.
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.
  - If 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

% Max-inactivity time in resolution time for 40 workers

Jean-Charles Régin, CP 2013
Inactivity time
% decomposition time

Jean-Charles Régis, CP 2013
Speedup with \#sppw

Satisfaction problems

Jean-Charles Régin, CP 2013
Speedup with #sppw

Optimization problems (with proof)

Jean-Charles Régin, CP 2013
Speedup with \#sppw

Satisfaction problems and optimization problems

Jean-Charles Régis, CP 2013
Comparison with or-tools

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

Jean-Charles Régin, CP 2013
## Comparison with work stealing

<table>
<thead>
<tr>
<th>instance</th>
<th>seq. time (s)</th>
<th>parallel time (s)</th>
<th>speedup</th>
<th>parallel time (s)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>allinterval_15</td>
<td>262,546</td>
<td>9,724</td>
<td>27,000</td>
<td>8,790</td>
<td>29,869</td>
</tr>
<tr>
<td>magicsequence_40000</td>
<td>328,194</td>
<td>592,648</td>
<td>0,554</td>
<td>37,334</td>
<td>8,791</td>
</tr>
<tr>
<td>sportsleague_10</td>
<td>172,390</td>
<td>7,648</td>
<td>22,541</td>
<td>6,782</td>
<td>25,419</td>
</tr>
<tr>
<td>sb_sb_13_13_6_4.fzn</td>
<td>135,674</td>
<td>9,203</td>
<td>14,742</td>
<td>7,752</td>
<td>17,502</td>
</tr>
<tr>
<td>quasigroup7_10.fzn</td>
<td>292,589</td>
<td>14,532</td>
<td>20,134</td>
<td>10,542</td>
<td>27,755</td>
</tr>
<tr>
<td>non_non_fast_6.fzn</td>
<td>602,222</td>
<td>271,254</td>
<td>2,220</td>
<td>56,843</td>
<td>10,594</td>
</tr>
<tr>
<td>golombruler_13</td>
<td>1355,150</td>
<td>54,889</td>
<td>24,689</td>
<td>44,321</td>
<td>30,576</td>
</tr>
<tr>
<td>warehouses</td>
<td>148,043</td>
<td>25,896</td>
<td>5,717</td>
<td>21,083</td>
<td>7,022</td>
</tr>
<tr>
<td>setcovering</td>
<td>94,431</td>
<td>16,117</td>
<td>5,859</td>
<td>11,089</td>
<td>8,516</td>
</tr>
<tr>
<td>2DLevelPacking_Class5_20_6.fzn</td>
<td>22,612</td>
<td>13,774</td>
<td>1,642</td>
<td>0,748</td>
<td>30,230</td>
</tr>
<tr>
<td>depot_placement_att48_5.fzn</td>
<td>125,154</td>
<td>19,083</td>
<td>6,558</td>
<td>10,17</td>
<td>12,306</td>
</tr>
<tr>
<td>depot_placement_rat99_5.fzn</td>
<td>21,628</td>
<td>6,368</td>
<td>3,396</td>
<td>2,613</td>
<td>8,277</td>
</tr>
<tr>
<td>fastfood_ff58.fzn</td>
<td>23,050</td>
<td>4,520</td>
<td>5,100</td>
<td>3,834</td>
<td>6,012</td>
</tr>
<tr>
<td>open_stacks_01_problem_15_15.fzn</td>
<td>102,753</td>
<td>6,080</td>
<td>16,900</td>
<td>5,759</td>
<td>17,842</td>
</tr>
<tr>
<td>open_stacks_01_wbp_30_15_1.fzn</td>
<td>185,715</td>
<td>15,350</td>
<td>12,099</td>
<td>11,219</td>
<td>16,554</td>
</tr>
<tr>
<td>sugiyama2_g5_7_7_7_7_2.fzn</td>
<td>286,537</td>
<td>22,751</td>
<td>12,594</td>
<td>10,766</td>
<td>26,615</td>
</tr>
<tr>
<td>pattern_setMining_k1_german-credit.fzn</td>
<td>113,724</td>
<td>22,317</td>
<td>5,096</td>
<td>13,782</td>
<td>8,252</td>
</tr>
<tr>
<td>radiation_03.fzn</td>
<td>129,076</td>
<td>33,479</td>
<td>3,855</td>
<td>25,626</td>
<td>5,037</td>
</tr>
<tr>
<td>baccp-7.fzn</td>
<td>227,152</td>
<td>15,647</td>
<td>14,517</td>
<td>9,517</td>
<td>23,868</td>
</tr>
<tr>
<td>talent_scheduling_alt_film116.fzn</td>
<td>254,297</td>
<td>13,510</td>
<td>18,823</td>
<td>35,625</td>
<td>7,138</td>
</tr>
</tbody>
</table>

Jean-Charles Régin, CP 2013
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 processes.
  - 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

Jean-Charles Régin, CP 2013