Toward Local Search Programming: LocalSolver 1.0

Thierry Benoist           Frédéric Gardi
Bouygues e-lab, Paris
{ tbenoist, fgardi }@bouygues.com

Bertrand Estellon         Karim Nouioua
Laboratoire d'Informatique Fondamentale - CNRS UMR 6166,
Faculté des Sciences de Luminy - Université Aix-Marseille, Marseille
{ bertrand.estellon, karim.nouioua }@lif.univ-mrs.fr
Introduction

Integer Programming (IP) is one of the most powerful tools of OR. Why?

- Simple and generic mathematical formalism
- Easy-to-use black-box solvers: “model & run” approach

But tree-search techniques are limited faced with large-scale problems…

Local Search (LS) allow to obtain quickly high-quality solutions.

But engineering LS is not easy. Working time distribution observed on challenging LS projects:

- search strategy = 10 %
- moves = 30 %
- evaluation machinery = 60 % → applied (incremental) algorithmics
State of the art

Several software libraries proposed to facilitate the implementation of the “search strategy” layer (ex: EasyLocal++, ParadisEO).

Two pioneering softwares for automating the “evaluation” layer:
- **Comet** (Van Hentenryck & Michel): CP-oriented language
- **iOpt** (British Telecom): Java library

Some of the best SAT and Pseudo-Boolean solvers are based on local-search techniques (ex: Walksat, WSAT(OIP)).

No effective “model & run” solver based on local search available for combinatorial optimization, as known in IP/CP.
LocalSolver project

2007 : LocalSolver project start

Long-term goals:
1) Simple declarative formalism enabling “LS programming” (*model*)
2) High-performance solver exploiting this formalism (*run*)
Guided by the fundamental principle : “do what LS experts would do”

2009 : First concretization: LocalSolver 1.0 software

- Allows to tackle an important class of combinatorial optimization problems: *matching, partitioning, packing, covering.*
- Binaries freely distributed under BSD license for Windows, Linux, Mac OS X on x86 architecture.
LocalSolver 1.0: formalism

Boolean modeling language, close to IP modeling, but…

1) Offering an enriched range of mathematical operators to define constraints and objectives:

- arithmetical: sum, min, max, product
- logical: and, or, xor, not, if-then-else
- relational: ≤, <, =, >, ≥, ≠

→ Allow to model strongly non-linear problems

2) Allow to define multiple objectives to optimize in lexicographic order.

→ Make goal programming easier: Minimize 1000000 x - 1000 y + z

Warning: modeling = definition of search space
Too much constraints counteract locals-search resolution
LocalSolver 1.0 : formalism

Boolean modeling language, close to IP modeling, but…

1) Offering an enriched range of mathematical operators to define constraints and objectives:
   - arithmetical: *sum, min, max, product*
   - logical: *and, or, xor, not, if-then-else*
   - relational: ≤, <, =, >, ≥, ≠

→ Allow to model strongly non linear problems

2) Allow to define multiple objectives to optimize in lexicographic order.

→ Make goal programming easier: Minimize x ; Maximize y ; Minimize z ;

Warning: *modeling = definition of search space*
Too much constraints counteract locals-search resolution
A small bin-packing problem written in LSP format: 3 items $x$, $y$, $z$ to pack into 2 piles $A$, $B$ in order to minimize the height of the highest pile.

```lsp
xA <- bool(); yA <- bool(); zA <- bool();
xB <- bool(); yB <- bool(); zB <- bool();
constraint booleansum(xA, xB) = 1;
constraint booleansum(xA, xB) = 1;
constraint booleansum(xA, xB) = 1;
heightA <- sum(2xA, 3yA, 4zA);
heightB <- sum(2xB, 3yB, 4zB, 5);
minimize heightMax <- max(heightA, heightB);
```
A small bin-packing problem written in LSP format: 3 items $x, y, z$ to pack into 2 piles $A, B$ in order to minimize the height of the highest pile:

\[
\begin{align*}
&A \leftarrow \text{bool}(); \quad yA \leftarrow \text{bool}(); \quad zA \leftarrow \text{bool}(); \\
&B \leftarrow \text{bool}(); \quad yB \leftarrow \text{bool}(); \quad zB \leftarrow \text{bool}(); \\
&\text{constraint booleansum}(xA, xB) = 1; \\
&\text{constraint booleansum}(xA, xB) = 1; \\
&\text{constraint booleansum}(xA, xB) = 1; \\
&\text{heightA} \leftarrow \text{sum}(2xA, 3yA, 4zA); \\
&\text{heightB} \leftarrow \text{sum}(2xB, 3yB, 4zB, 5); \\
&\text{minimize heightMax} \leftarrow \text{max}(\text{heightA}, \text{heightB});
\end{align*}
\]

If you wish to maximize the height of the smallest pile, as second objective, just add the following line:

\[
\text{maximize heightMin} \leftarrow \text{min}(\text{heightA}, \text{heightB});
\]
LocalSolver 1.0 : solver

Two ways of use:

- **Black box**: autonomous solver, executable in command-line or using high-level API (ISO C++, Java 5.0, C# 2.0).

- **White box**: open solver, to program LS in C++ by letting the evaluation to the solver while overriding heuristic and moves.

Representation of the LSP by a DAG:

```plaintext
x1 <- bool(); x2 <- bool(); x3 <- bool();
x1 <- bool(); y2 <- bool(); y3 <- bool();
sx <- booleansum(x1, x2, x3);
sy <- booleansum(y1, y2, y3);
constraint sx <= 2;
constraint sy >= 2;
obj <- max(sx, sy);
minimize obj;
```
LocalSolver 1.0: solver

Why does it work?
Why does it work?

1) Highly-optimized incremental evaluation:

Propagation of modifications in the DAG: *Lazy Breadth-First Search*

Each node is visited at most once. A node is visited only if the modification of one of its parents makes its value obsolete.

Ex: a node \( x \leftarrow a < b \) whose current value equals true. If \( a \) is decreased or \( b \) is increased, then \( x \) is not visited.

Fine exploitation of invariants induced by mathematical operators

Ex: a node \( z \leftarrow \text{or}(a_1, \ldots, a_k) \) with \( T \) the list of true \( a_i \) and \( M \) the list of modified \( a_i \). If \( |T| \neq |M| \), then \( z \) will be true. In this case, evaluation in \( O(1) \) time.
Why does it work?

2) Abstract but structured moves for preserving feasibility:

Generalization of *ejection chains* in the hypergraph induced by (boolean) decision variables and constraints.

These moves, called *k-Chains and k-Cycles*, simulate *k-Moves* and *k-Exchanges* respectively in packing/covering models.
6 realistic test problems (chosen before project start):

- Car sequencing (CSPLIB + Renault)
- Social golfer (CSPLIB + BySA)
- Steel mill slab design (CSPLIB)
- Spot 5 photographs scheduling (CNES)
- Minimum formwork stock (ByCons)
- Eternity 2 puzzle (Tomy)

And since…

- Bin packing (M. Van Caneghem)
- Graph coloring (M. Van Caneghem)
- University timetabling (M. Van Caneghem)
- Domino portraits (G. Rochart)
- Wedding seating plans (TF1)
- Driving license examinations (Eurodecision)
**LocalSolver 1.0 : benchmarks**

**Realized on a standard computer : 2.33 GHz, RAM 2 Go, L2 4 Mio**

Car sequencing (CSPLIB):

<table>
<thead>
<tr>
<th></th>
<th>60 sec</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10-93</td>
<td>200-01</td>
<td>300-01</td>
<td>400-01</td>
<td>500-08</td>
</tr>
<tr>
<td>State-of-the-art LS</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LocalSolver (black)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>CPLEX 11.2</td>
<td>6</td>
<td>11</td>
<td>27</td>
<td>17</td>
<td>X</td>
</tr>
<tr>
<td>CBLS Comet 2.1</td>
<td>7</td>
<td>8</td>
<td>16</td>
<td>18</td>
<td>91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>600 sec</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10-93</td>
<td>200-01</td>
<td>300-01</td>
<td>400-01</td>
<td>500-08</td>
</tr>
<tr>
<td>State-of-the-art LS</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LocalSolver (black)</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>CPLEX 11.2</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>16</td>
<td>104</td>
</tr>
<tr>
<td>CBLS Comet 2.1</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>18</td>
<td>47</td>
</tr>
</tbody>
</table>
LocalSolver 1.0 : benchmarks

Car sequencing (RENAULT, ROADEF 2005 Challenge):

<table>
<thead>
<tr>
<th></th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>600 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State-of-the-art LS</td>
<td>0, 192, 66 (1/19)</td>
<td>0, 337, 6 (1/19)</td>
<td>0, 160, 407 (1/19)</td>
</tr>
<tr>
<td>LocalSolver (black)</td>
<td>0, 268, 212 (16/19)</td>
<td>36, 544, 187 (16/19)</td>
<td>2, 353, 692 (18/19)</td>
</tr>
</tbody>
</table>

No IP/CP/SAT solvers is able to tackle such instances. Comet is not able to find admissible solutions without relaxing paint color constraints.

X2 : **1260 vehicles**, 12 options, 13 colors
LSP : **516 936 variables**, whose **374 596 booleans**
LocalSolver : **3 M moves per minute**, 450 Mo RAM
LocalSolver 1.0 : benchmarks

Steel mill slab design (CSPLIB):

<table>
<thead>
<tr>
<th></th>
<th>2-0</th>
<th>3-0</th>
<th>4-0</th>
<th>5-0</th>
<th>6-0</th>
<th>7-0</th>
<th>8-0</th>
<th>9-0</th>
<th>10-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art LS</td>
<td>28</td>
<td>6</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LocalSolver (black)</td>
<td>46</td>
<td>52</td>
<td>35</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CPLEX 11.2</td>
<td>178</td>
<td>511</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>275</td>
<td>226</td>
<td>229</td>
<td>201</td>
</tr>
<tr>
<td>CBLS Comet 2.1</td>
<td>136</td>
<td>135</td>
<td>69</td>
<td>65</td>
<td>42</td>
<td>30</td>
<td>26</td>
<td>21</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2-0</th>
<th>3-0</th>
<th>4-0</th>
<th>5-0</th>
<th>6-0</th>
<th>7-0</th>
<th>8-0</th>
<th>9-0</th>
<th>10-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art LS</td>
<td>28</td>
<td>6</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LocalSolver (black)</td>
<td>40</td>
<td>34</td>
<td>35</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CPLEX 11.2</td>
<td>94</td>
<td>65</td>
<td>X</td>
<td>63</td>
<td>X</td>
<td>189</td>
<td>226</td>
<td>97</td>
<td>64</td>
</tr>
<tr>
<td>CBLS Comet 2.1</td>
<td>124</td>
<td>110</td>
<td>43</td>
<td>58</td>
<td>33</td>
<td>33</td>
<td>17</td>
<td>17</td>
<td>15</td>
</tr>
</tbody>
</table>
Conclusion

LocalSolver 1.0:
“Local Search Programming” is possible!

For more details and downloads: Google “LocalSolver”

LocalSolver 1.x:
- Implementing metaheuristics (ex: simulated annealing)
- Reinforcing autonomous moves (ex: + large, + targeted)
- Managing decimal coefficients (and big integers)

LocalSolver 2.0: introducing the notion of sets in the formalism

Future: integer programming → mixed integer programming
Integrating a “continuous” solver (like simplex) in the DAG?
Acknowledgments

LocalSolver 1.0 :
“Local Search Programming” is possible!
For more details and downloads : Google “LocalSolver”

We warmly thank all people who have contributed, directly or indirectly, to the LocalSolver project. Particularly:

Antoine Jeanjean, Guillaume Rochart (Bouygues e-lab)
Michel Van Caneghem (LIF, Université Aix-Marseille)
Romain Megel (École des Mines de Nantes)
Lucile Robin (École Centrale de Lyon)
Sophia Zaourar (ENSIMAG, Grenoble INP)