BOUYGUES INNOVATION & OPTIMISATION

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

<ins>Boolean modeling</ins> language, close to IP modeling, but…

1) Offering an <ins>enriched range of mathematical operators</ins> to define constraints and objectives:

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

→ Allow to model <ins>strongly non linear problems</ins>

2) Allow to define <ins>multiple objectives</ins> to optimize in lexicographic order.

→ Make <ins>goal programming</ins> easier: Minimize x ; Maximize y ; Minimize z ;

Warning : <ins>modeling = definition of search space</ins>
Too much constraints counteract locals-search resolution

LocalSolver 1.0 : formalism

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.

```
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);
```

LocalSolver 1.0 : formalism

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:

```
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);
```

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

```
maximize heightMin <- min(heightA, heightB);
```

LocalSolver 1.0 : solver

Two ways of use:

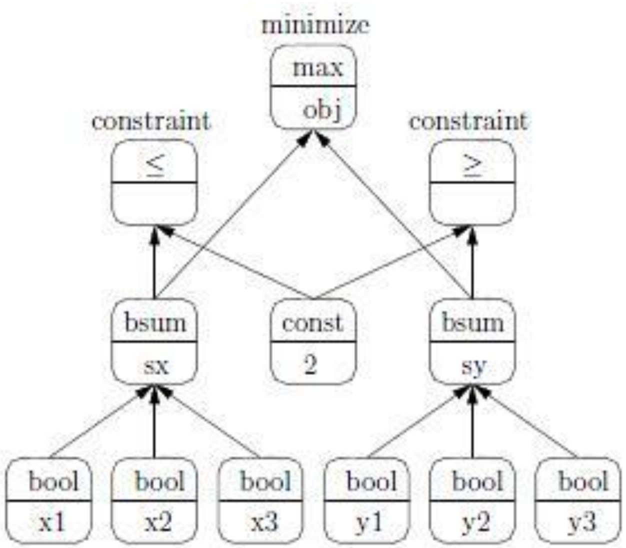
- *Black box* : <u>autonomous</u> solver, executable in command-line or using high-level API (ISO C++, Java 5.0, C# 2.0).

- *White box* : <u>open</u> 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:

```
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;
```

$\longrightarrow$



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 <- 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

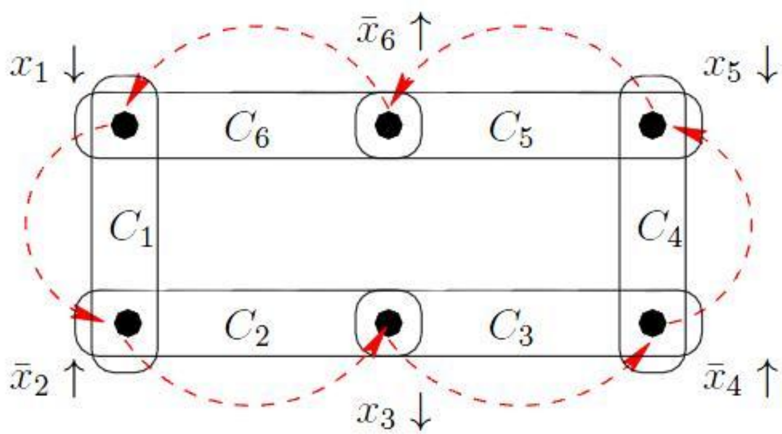
Ex : a node $z \leftarrow or(a_1, \ldots, a_k)$ with T the list of true a_i and M the list of modified a_i. If $|T| \neq |M|$, then z will be true. In this case, evaluation in $O(1)$ time.

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.



LocalSolver 1.0 : benchmarks

6 realistic test problems (chosen before project start):

Car sequencing (CSPLIB + Renault)	matching
Social golfer (CSPLIB + BySA)	matching/packing
Steel mill slab design (CSPLIB)	bin-packing
Spot 5 photographs scheduling (CNES)	knapsack
Minimum formwork stock (ByCons)	set-covering
Eternity 2 puzzle (Tomy)	tiling

And since…

Bin packing (M. Van Caneghem)	bin-packing
Graph coloring (M. Van Caneghem)	coloring
University timetabling (M. Van Caneghem)	matching/packing
Domino portraits (G. Rochart)	tiling
Wedding seating plans (TF1)	matching/packing
Driving license examinations (Eurodecision)	set-covering

LocalSolver 1.0 : benchmarks

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

Car sequencing (CSPLIB):

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

600 sec	10-93	200-01	300-01	400-01	500-08
State-of-the-art LS	3	0	0	1	0
LocalSolver (black)	6	5	4	6	6
CPLEX 11.2	3	3	11	16	104
CBLS Comet 2.1	7	6	10	18	47

LocalSolver 1.0 : benchmarks

Car sequencing (RENAULT, ROADEF 2005 Challenge):

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

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):

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

600 sec	2-0	3-0	4-0	5-0	6-0	7-0	8-0	9-0	10-0
State-of-the-art LS	28	6	34	0	0	0	0	0	0
LocalSolver (black)	40	34	35	3	7	1	0	0	0
CPLEX 11.2	94	65	X	63	X	189	226	97	64
CBLS Comet 2.1	124	110	43	58	33	33	17	17	15

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 <u>sets</u> 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)