LocalSolver: black-box local search for combinatorial optimization

Frédéric Gardi

Bouygues e-lab, Paris
fgardi@bouygues.com

Joint work with T. Benoist, J. Darlay, B. Estellon, R. Megel, K. Nouioua
Bouygues e-lab & LIF UMR 7279 - Université Aix-Marseille

Practical observations

1) What is the most powerful tool provided by OR?

Mixed Integer Linear Programming (MILP):

- Simple and generic formalism
- Easy-of-use solvers: « *model & run* » approach

Indispensable tool for practitioners.

2) What do practitioners when IP solvers are ineffective?

<u>Local Search (LS)</u>: allows to obtain quality solutions in a few minutes. But induces extra costs (development, maintenance).

Why local search works

What are the needs in business and industry?

1) Clients have optimization problems, and rarely satisfaction problems.

“No solution found” is rarely an acceptable answer for users. Thus, once the model is well stated, finding a feasible solution should be easy.

→ Goal programming (constraints → objectives)

2) Optimal solution is not what clients really want.

- *Proof* of optimality is much less what they want
- They want first a nice software providing good solutions quickly

→ Don’t be focused on optimality

Why local search works

LS = good-quality solutions within short running times
LS = practical solution for practical problems

But LS ≠ metaheuristics, LS ≠ cooking. Our vision:

LS = incomplete & non deterministic search
LS = randomized moves + incremental computation (= run fast)

→ Less Maths (analytical), more Computer Science (algorithmic)
→ A lot of software and algorithm engineering

Why tree search is limited

Mixed-integer programming techniques (B&B, B&C, BCP) are:

- Designed for *proving optimality*

- Not designed to find *feasible solutions*

MIP solvers still fail to find feasible solutions for real-life instances with no more than 10,000 binaries.

Our conviction: pure tree-search (TS) techniques will remain powerless for solving very large-scale combinatorial problems (millions of binaries).

Why?

Why tree search is limited

1) Relaxation is often useless but costs a lot in efficiency. So why losing running time to enumerate *partial* solutions?

2) Why an *incomplete* TS should be better than LS? Moreover, TS is not really suited for exploring randomly a search space.

Facts:

State-of-the-art IP solvers integrate more and more LS ingredients (Local Branching, Relaxation Induced Neighborhood Search).

TSP records:
- B&C [Applegate, Bixby, Cook, Chvátal, etc.]: 85,900 cities
- LS [Helsgaun]: 1,904,711 cities (World TSP), and until 10,000,000 cities

An idea

If LS is the only technique allowing to scale, why not considering a solver founded on LS?

A few works in this way in CP, SAT, and even IP communities…

But presently, who knows (and uses) an effective black-box local-search solver for combinatorial optimization?

A project

2007: Beginning of LocalSolver project

Long-term objectives:

1) Defining a simple, generic declarative formalism suited for LS (*model*)

2) Developing an effective LS-based solver with fundamental principle: « doing what an expert would do » (*run*)

2009: First software concretization: LocalSolver 1.0

- Allows to tackle large-scale 0-1 programs

- Binaries freely distributed at www.localsolver.com

2011: LocalSolver 1.1 (multithreading, enriched moves, annealing)

LocalSolver: formalism

Generalized 0-1 programming

1) Mathematical operators for declaring constraints and objectives:

- arithmetic : *sum*, *min*, *max, prod, div, mod, abs, sqrt*
- logical, conditional : *and*, *or*, *xor*, *not, if-then-else*
- relational : ≤, <, =, >, ≥, ≠

→ Allows to model simply highly nonlinear 0-1 problems

2) Lexicographic multiple objectives

→ Facilitating *goal programming* : Minimize *x* ; Maximize *y* ; Minimize *z* ;

Modeling = defining the search space

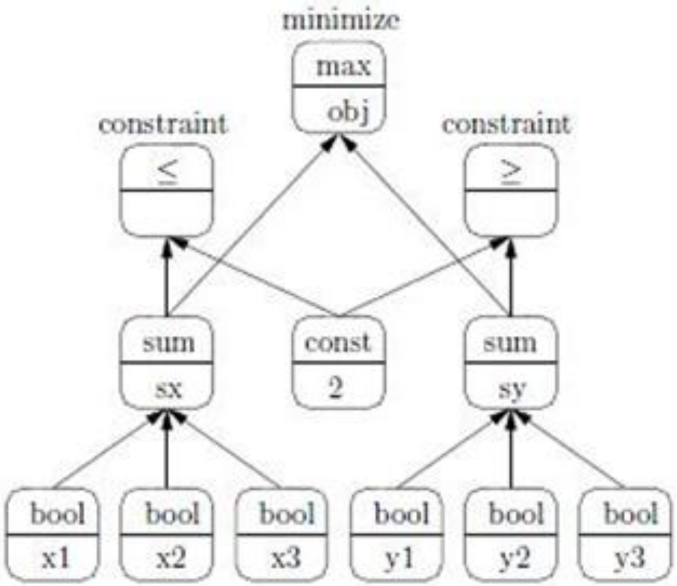
LS-suited model = softly constrained model = large search space

LocalSolver: formalism

Representation of the model as a DAG

```
x1 <- bool(); x2 <- bool(); x3 <- bool();
y1 <- bool(); y2 <- bool(); y3 <- bool();
sx <- sum(x1, x2, x3);
sy <- sum(y1, y2, y3);
constraint sx <= 2;
constraint sy >= 2;
obj <- max(sx, sy);
minimize obj;
```

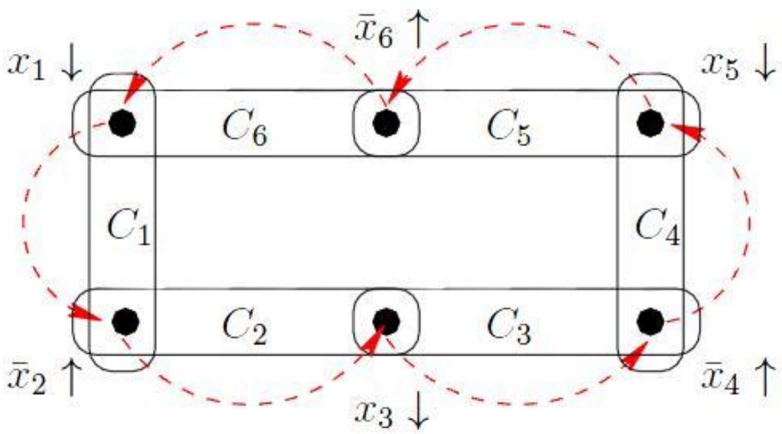
→



1) Moves preserving feasibility

Generalization of ejection chains in the hypergraph induced by decision variables and constraints.

These moves, namely *k*-Chains and *k*-Cycles, correspond to *k*-Moves and *k*-Exchanges in packing and covering models.



2) Highly-optimized incremental evaluation

Lazy propagation of modifications induced by a move in the DAG

Each node of the DAG is visited at most once, only if the value of one of its children is modified.

Ex: `x <- a < b` being true. If `a` is decreased or `b` increased by a move, then `x` is not evaluated.

Exploitation of invariants induced by mathematical operators

Ex: $z \leftarrow or(a_1, \ldots, a_k)$ with T the list of $a_i = 1$ and M the list of a_i modified by a move. If $|T| \neq |M|$, then $z = 1 \rightarrow$ Shortcut in $O(1)$ time.

LocalSolver: benchmarks

Steel mill slab design (CSPLIB): minimization

60 sec	2-0	3-0	4-0	5-0	6-0	7-0	8-0	9-0	10-0
State-of-the-art	22	5	32	0	0	0	0	0	0
LocalSolver 1.1	**37**	**8**	**35**	**1**	**4**	**1**	**0**	**0**	**0**
CPLEX 12.2	136	288	X	126	X	232	226	163	133
CPO 2.3	90	65	58	50	54	46	28	29	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	22	5	32	0	0	0	0	0	0
LocalSolver 1.1	**31**	**7**	**34**	**0**	**4**	**0**	**0**	**0**	**0**
CPLEX 12.2	94	65	X	63	X	189	226	97	64
CPO 2.3	62	38	40	42	36	36	21	23	18

LocalSolver: applications

Inside Bouygues Group:

- TF1 Publicité: TV-ads assignment
- ETDE: lighting maintenance planning
- Colas UK: route maintenance planning
- By Habitat Social: formwork stock optimization
- 1001 Mariages: wedding table planning
- By SA: seminar planning

But also outside: **1000 downloads of LocalSolver 1.1**

Having benchmarked LocalSolver 1.1 on several projects, Eurodecision (French OR service company) is interested in buying LocalSolver…

LocalSolver 2.0

From the free proof-of-concept to a commercial software…

- Major evolutions: functionally and technically
- Release scheduled for February 2012

- Always free for teaching
- No longer free for commercial uses

- All info coming soon at www.localsolver.com

LocalSolver 2.0

1) LocalSolver's modeler (LSP language) for fast prototyping

```
function model() {
    // 0-1 decisions
    x[0..nbItems-1] <- bool();

    // weight constraint
    sackWeight <- sum[i in 0..nbItems](weights[i] * x[i]);
    constraint sackWeight <= sackBound;

    // maximize value
    sackValue <- sum[i in 0..nbItems](values[i] * x[i]);
    maximize sackValue;
}
```

2) Lightweight object-oriented API (C++, Java, .NET) for full integration

3) Quick Start Guide, API reference, modeler reference, tutorials

4) Binaries for Windows, Linux, Mac OS and x86, x64 (lib + exe)

LocalSolver 2.0

LocalSolver is able to tackle very large-scale real-life 0-1 programs (with nonlinear constraints and objectives): 10 millions of binary variables.

LocalSolver attacks ROADEF 2012 Challenge proposed by Google:

Instances	Variables	Binaries	Constraints	Solutions
A2-1	1,415,324	100,000	102,300	1,984,001
A2-2	3,769,381	100,000	19,770	1,268,279,367
A2-3	3,843,977	100,000	20,213	1,683,410,301
A2-4	1,537,771	50,000	13,373	2,035,401,379
A2-5	1,556,017	50,000	13,260	522,930,188

1 million of feasible solutions explored in 5 minutes (with 2 cores).

LocalSolver: roadmap

<u>Mid term</u>: **integer/set programming**

For modeling with integers (as indices of sets).
For tackling <u>scheduling and routing</u> problems.

<u>Long term</u>: **mixed-variable programming**

Dealing with continuous decision variables → MILP, MINLP.

→ Extending modeling capacities while maintaining efficiency

www.localsolver.com