Toward a mathematical optimization solver based on neighborhood search

Thierry Benoist, Julien Darlay, Bertrand Estellon,
Frédéric Gardi, Romain Megel, Clément Pajean

Innovation 24 & LocalSolver

www.localsolver.com

May 5th, 2015
DORS AOO, Copenhagen

Who we are

BOUYGUES

Bouygues, one of the French largest corporation, €33 bn in revenues
http://www.bouygues.com

Innovation24

Operations Research subsidiary of Bouygues
15 years of practice and research
http://www.innovation24.fr

LocalSolver

Mathematical optimization solver commercialized by Innovation 24
http://www.localsolver.com

LocalSolver

Automating local search

Local search

An iterative improvement method

- Explore a neighborhood of the current solution
- Smaller or larger neighborhoods

$\longrightarrow$ Incomplete exploration of the solution space

Essential in combinatorial optimization

- Hidden behind many textbook algorithms (ex: simplex, max flow)
- In the heart of all metaheuristic approaches
- Proved to be inefficient in the worst case
- Largely used because very effective in practice

Why local search?

When it is hopeless to enumerate

- Large-scale combinatorial problems
- When relaxation or inference brings nothing
  (ex: linear relaxation is very fractional)
- When computing relaxation or inference is costly

Adapted to client needs

- Good-quality optima satisfy them
- Fast: each iteration runs in sublinear or even constant time

⟶ Solutions in short running times + ability to scale

Existing tools

Libraries and frameworks

- Complex to handle
- Limited to practitioners having good programming skills
- Don’t address key points (ex: moves)

Solvers integrating “pure” local search

- Pioneering works in SAT community
- MIP & CP: a few attempts but a limited impact (Nonobe & Ibaraki 2001)
- MIP & CP: a lot of heuristic ingredients but no “pure” local search

LocalSolver project

2007: launch

- Define a generic modeling formalism (close to MIP) suited for a local search-based resolution (*model*)
- Develop an effective solver based on pure local search with first principle: “to do what an expert would do” (*run*)

2010: first release

- Large-scale combinatorial problems – especially assignment, packing, covering, partitioning problems – out of scope of classical solvers
- Integration in Innovation 24’s optimization solutions
- First uses outside Innovation 24

P-median

Select a subset P among N points minimizing the sum of distances from each point in N to the nearest point in P

```
function model() {
  x[1..N] <- bool() ; // decisions: point i belongs to P if x[i] = 1
  constraint sum[i in 1..N]( x[i] ) == P ; // constraint: P points selected among N
  minDist[i in 1..N] <- min[j in 1..N]( x[j] ? Dist[i][j] : InfiniteDist ) ; // expressions: distance to the nearest point in P
  minimize sum[i in 1..N]( minDist[i] ) ; // objective: to minimize the sum of distances
}
```

Nothing else to write: “model & run” approach

- Straightforward, natural mathematical model
- Direct resolution: no tuning

Mathematical operators

Decisional	Arithmetic			Logical	Relational
bool	sum	sub	prod	not	==
float	min	max	abs	and	!=
int	div	mod	sqrt	or	<=
	log	exp	pow	xor	>=
	cos	sin	tan	if	<
	floor	ceil	round	array + at	>

New in 5.0: operator piecewise to model piecewise linear functions

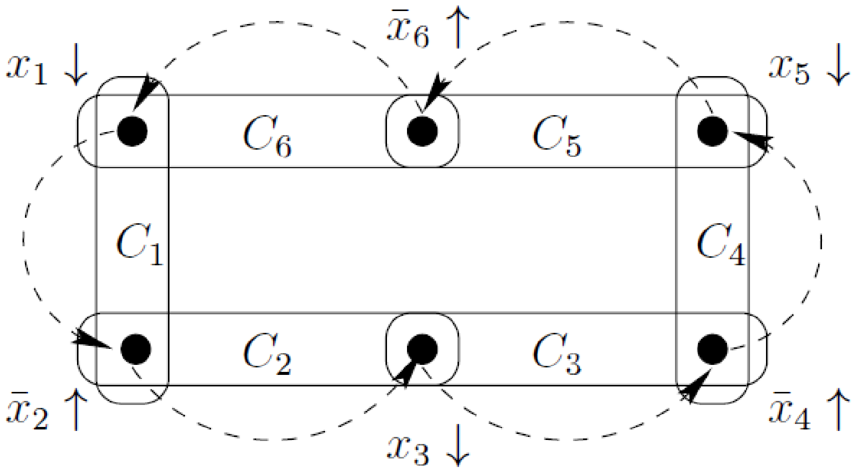
Small, structured neighborhoods

The classic in Boolean Programming: "k-flips"

- Lead to infeasible solutions for structured (= real-life) problems
- Feasibility is hard to recover: slow convergence

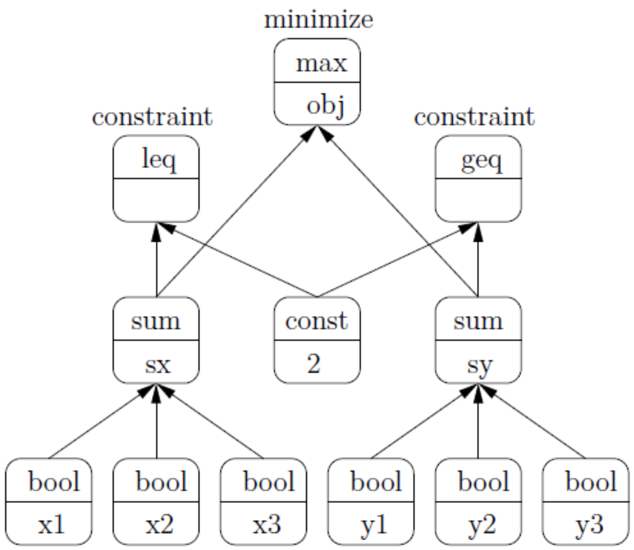
LocalSolver moves tend to preserve feasibility

- Destroy & repair approach
- Ejection paths in the constraint hypergraph
- More or less specific to some combinatorial structures



Fast exploration

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



Incremental evaluation

- Lazy propagation of modifications induced by a move in the DAG
- Exploitation of invariants induced by math operators

⟶ Millions of moves evaluated per minute of running time

Car sequencing

2005 ROADEF Challenge: http://challenge.roadef.org/2005/en

Large-scale instances

- Until 1,300 vehicles to sequence: 400,000 binary decisions

Instance with 540 vehicles

- Small instance: 80,000 variables including 44,000 binary decisions
- State of the art: 3,109 by specific local search (winner of the Challenge)
- Lower bound: 3,103

Minimization

Results

- Gurobi 5.5: 3.027e+06 in 10 min | 194,161 in 1 hour
- LocalSolver 3.1: 3,476 in 10 sec | 3,114 in 10 min

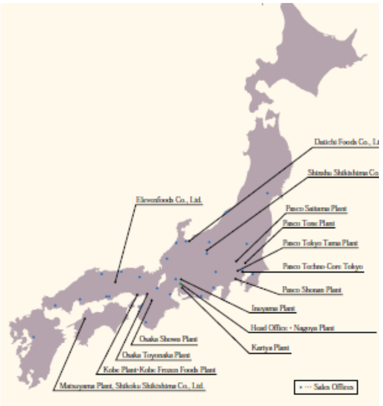
Supply chain optimization

Global supply chain optimization

- Both production and logistics optimization
- 10 factories, each with several production lines
- Large number of stores and distribution centers

A challenging context for LocalSolver

- 20,000,000 variables including 3 million binaries
- Rich model involving setup costs, delivery times, packaging, etc.
- Vain attempts to solve the problem with MIP solvers
- LocalSolver finds high-quality solutions in 5 minutes



Application panorama

TV media planning

Outdoor & indoor advertising

Logistic clustering and routing

Road maintenance planning

Network deployment planning

Loan assembling optimization

Placement of nuclear fuel assemblies in pools

Airline network management

Weapon resource allocation

Packing and transportation of military equipment

LocalSolver

Novelties coming in June

Set-based modeling

Structured decisional operator ulist(n)

- Order a subset of values in domain {0, …, n-1}
- Each value is unique in the list

Classical operators to interact with "ulist"

- count(u): number of values selected in the list
- get(u,i) or u[i]: value at index i in the list
- indexOf(u,v): index of value v in the list
- contains(u,v): equivalent to "indexOf(u,v) != -1"
- disjoint(u1, u2, …, uk): true if u1, u2, …, uk are pairwise disjoint
- partition(u1, u2, …, uk): true if u1, u2, …, uk induce a partition of {0, …, n-1}

Traveling salesman

```
function model() {
  x <- ulist(N) ; // order n cities {0, ..., n-1} to visit
  constraint count(x) == N; // exactly n cities to visit
  minimize sum[i in 1..N-1]( Dist[ x[i-1] ][ x[i] ] )
                    + Dist[ x[N-1] ][ x[0] ] ; // minimize sum of traveled distances
}
```

Could you imagine simpler model?

- Natural declarative model: straightforward to understand
- Common set-oriented concepts: easy to learn
- Even easier for people with basic programming skills
- Compact: linear in the size of input → highly scalable

Vehicle routing

```
function model() {
  x[1..K] <- ulist(N) ; // for each truck, order the clients to visit
  constraint partition( x[1..K] ); // each client is visited once
  distances[k in 1..K] <- sum[i in 1..N-1]( dist( x[k][i-1], x[k][i]) )
              + dist( x[k][N-1], x[k][0] ); // traveled distance for each truck

  minimize sum[k in 1..K]( distances[k] ); // minimize total traveled distance
}
```

To go further, to make it simpler

- Sets (unordered) versus lists (ordered)
- Multi-sets/lists: multiple occurrence of the same values
- Collections of objects instead of values
- Ability to iterate and project over collections (lambda expressions)

CVRP benchmarks

CVRP - instances A

- 32 to 80 clients, 10 trucks max.
- 27 instances
- 5 minutes of running time
- LS binary: 3 % avg. opt. gap
- LS ulist: 1 % avg. opt. gap

CVRP - instances X100-500

- 100 to 500 clients, 138 trucks max.
- 67 instances
- 5 minutes of running time
- LS binary: N/A
- LS ulist: 9 % avg. opt. gap

CVRPTW benchmarks

CVRPTW – instances Solomon R100

- 101 to 112 clients, 19 trucks max.
- 13 instances
- 5 minutes of running time
- LS binary: N/A
- LS ulist: 3 % avg. opt. gap

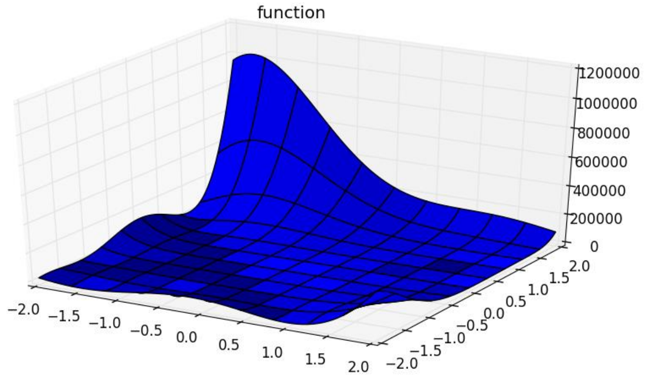
CVRPTW – instances Solomon R200

- 201 to 208 clients, 4 trucks max.
- 8 instances
- 5 minutes of running time
- LS binary: N/A
- LS ulist: 8 % avg. opt. gap

Black-box optimization

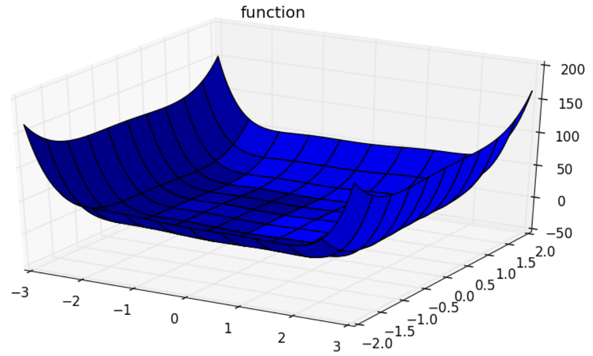
Context

- Unknown objective (oracle)
- Costly to evaluate
- Derivative free
- Continuous & integer decisions
- Bounds on variables



Application

- Parametric optimization
- Simulation optimization



Features

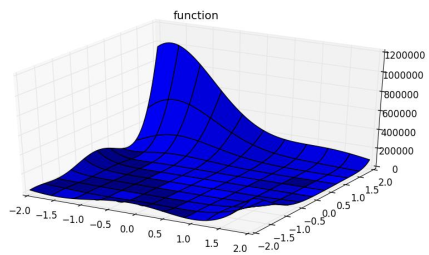
Learning objective via Radial Basis Functions

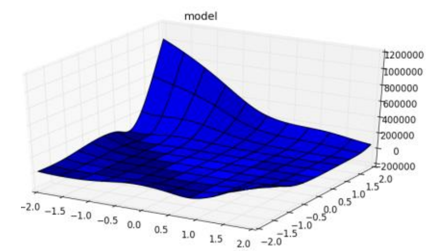
Automatic model selection

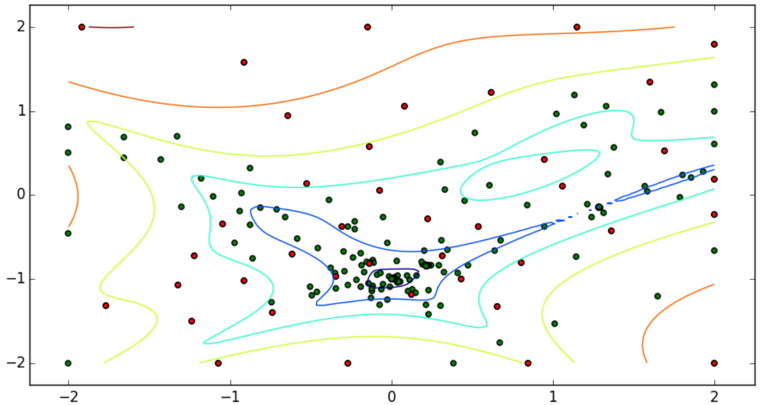
Optimization & Exploration

- Find the best point on the model
- Explore a new region

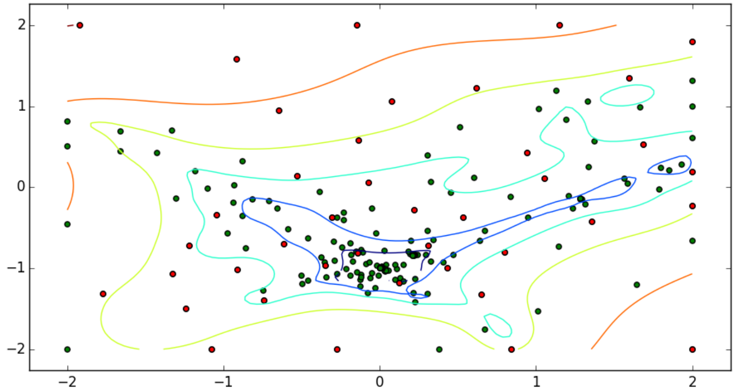
→ NLP subproblems solved through LocalSolver techniques







Function



Model

Black-box benchmark

Instances

- 25 instances from the recent paper by A. Costa and G. Nannicini.

  RBFOpt: an open-source library for black-box optimization with costly function evaluations. Optimization Online. (Under review)

- 20 runs per instance, 150 calls max. to the BB per run

Preliminary results

- RBFOpt: 345 opt. solutions found, 82 calls avg. per run
- LocalSolver: 310 opt. solutions found, 94 calls avg. per run
- NOMAD: 170 opt. solutions found, 150 avg. calls on per run

Roadmap

John N. Hooker (2007)

"Good and Bad Futures for Constraint Programming (and Operations Research)"
Constraint Programming Letters 1, pp. 21-32

"Since modeling is the master and computation the servant, no computational method should presume to have its own solver.

This means there should be no CP solvers, no MIP solvers, and no SAT solvers. All of these techniques should be available in a single system to solve the model at hand.

They should seamlessly combine to exploit problem structure. Exact methods should evolve gracefully into inexact and heuristic methods as the problem scales up."

LocalSolver

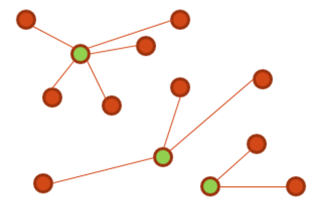
Hybrid math programming solver

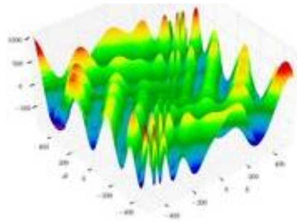
For combinatorial, numerical, or mixed-variable optimization

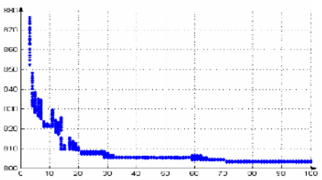
Particularly suited for large-scale non-convex optimization

High-quality solutions in seconds without tuning

LocalSolver
=
LS + CP/SAT + LP/MIP + NLP







free trial with support – free for academics – renting licenses from 590 €/month – perpetual licenses from 9,900 €

www.localsolver.com