LocalSolver
mathematical programming by local search

How LocalSolver qualified with a 100-lines model?

Thierry Benoist Julien Darlay Bertrand Estellon
Frédéric Gardi Romain Megel Karim Nouioua

LocalSolver

LocalSolver in a nutshell

What is LocalSolver?

The first math programming solver based on local search

- Pure model-and-run approach : no extra code to write
- Solve highly nonlinear 0-1 models
- Scale up to 10 million decision variables

→ *Solve problems intractable with IP/CP/SAT solvers*

Portable software

- Fully portable: Windows, Linux, Mac OS (x86, x64)
- Light object-oriented APIs: a few classes only
- Lightweight APIs available for C++, Java, .Net

Comes with an innovative modeling language for fast prototyping

Why local search, why LocalSolver?

Weaknesses of tree search

- Not suited to reach quickly good “integer feasible solutions”
- Designed to <u>prove</u> optimality
- Exponential time: not scalable (the best IP solvers still fail to find feasible solutions for real-life instances with 10,000 binaries)
- An incomplete tree search is not more optimal than a local search

Practitioners need :

- A solver which provides high-quality solutions in seconds
- A scalable solver which tackle problems with millions of variables
- A solver which proves optimality of infeasibility when possible

LocalSolver

How it works?

3 main layers :

LS solver must work as a LS practitioner works

1. **Incremental algorithm, sublinear evaluation**
   exploit the invariants induced by the mathematical operators
   → thousands of solutions explored each second

2. **Structured moves that maintain feasibility**
   moves performed on the hypergraph of decisions
   and constraints (ejection chains, cycles, ...)

3. **Heuristic and search strategy**
   heuristic based on simulated annealing to get out of local optima
   multithreading to ensure faster convergence and robustness

90%
of the
kernel

LocalSolver

The EURO/Roadef Challenge

LocalSolver modeling

Model in 3 parts

Each step corresponds to a specific function in the modeler

1. **Read the input data**
   open file, read the initial assignment, read resources, groups, …

2. **Write the model**
   Declare boolean variables, constraints, objectives, …

3. **Parameterize the resolution**
   Set time or iteration limit, load an initial solution.

100-lines , 1 day of work

LocalSolver

How to model with LocalSolver ?

1. **Declare the decision variables**
   A decision is a variable you can't compute from other variables

2. **Declare the constraints of your problem**

3. **Declare the objectives**

LocalSolver

Decisions and basic constraints

Assignment of processes to machines

These decisions completely determine the solution

```
// 0-1 decisions
x[0..nbProcesses-1][0..nbMachines-1] <- bool();
```

Binary decision variables

Compact loop syntax

$x_{pm} = 1 \Leftrightarrow$ process *p* on machine *m*

Each pocess must be assigned to a single machine

```
for [p in 0..nbProcesses-1]
  constraint sum[m in 0..nbMachines-1](x[p][m]) == 1;
```

Sum of nbMachines terms

Capacity constraints

```
for [m in 0..nbMachines-1][r in 0..nbResources-1] {
  u[m][r] <- sum[p in 0..nbProcesses-1](require[p][r] * x[p][m]);
  constraint u[m][r] <= capacity[m][r];
}
```

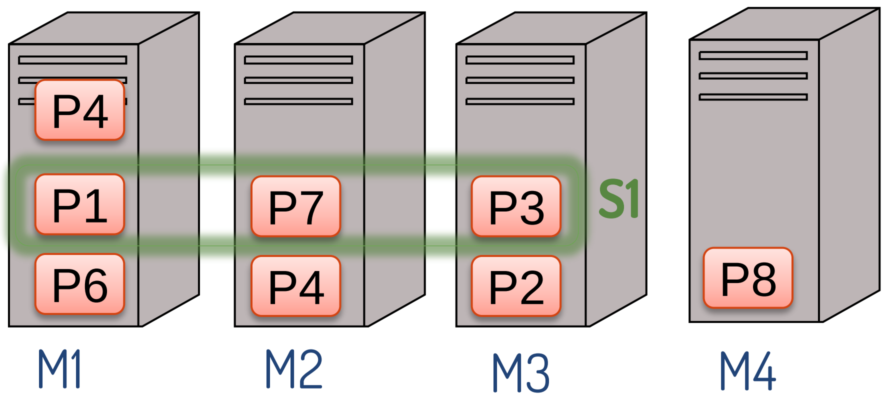
Integer intermediate variables

Other constraints

Conflict constraints

processes of the same service must run on distinct machines

```
for [s in 0..nbServices-1][m in 0..nbMachines-1]
  constraint sum[p in processByService[s]](x[p][m]) <= 1;
```



Spread constraints

processes of the same service must spread on a set of locations

```
for [s in 0..nbServices-1] {
  coveredLocations[s] <- sum[l in 0..maxLocation](
               or[p in processByService[s]][m in machineByLocation[l]](x[p][m]));
  constraint coveredLocations [s] >= spread[s];
}
```

Objectives

Objective : Load cost

```
loadCost[r in 0..nbResources-1] <- sum[m in 0..nbMachines-1](max(u[m][r] - safety[m][r], 0));
totalLoadCost <- sum[r in 0..nbResources-1](rweight[r] * loadCost[r]);
```

Objective : Balance cost

```
a[m in 0..nbMachines-1][r in 0..nbResources-1] <- capacity[m][r] - u[m][r];
for [b in 0..nbBalances-1] {
  r1 = resource1[b];
  r2 = resource2[b];
  tg = target[b];
  balanceCost[b] <-  sum[m in 0..nbMachines-1](max(tg * a[m][r1] - a[m][r2], 0));
}

totalBalanceCost <- sum[b in 0..nbBalances-1](bweight[b] * balanceCost[b]);
```

Objective : Process move cost

```
processMoveCost <- sum[p in 0..nbProcesses-1](pcost[p] * not(x[p][initialMachine[p]]));
```

Objectives

Objective : Service move cost

```
for [s in 0..nbServices-1]
  nbMoved[s] <- sum[p in 0..nbProcesses-1 : service[p] == s](!x[p][initialMachine[p]]);
serviceMoveCost <- max[s in 0..nbServices-1](nbMoved[s]);
```

Objective : Machine move cost

```
for [p in 0..nbProcesses-1] {
  m0 = initialMachine[p];
  machineMoveCost[p] <- sum[m in 0..nbMachines-1 : m != m0](mcost[m0][m] * x[p][m]);
}
totalMachineMoveCost <- sum[p in 0..nbProcesses-1](machineMoveCost[p]);
```

Total cost

```
obj <- totalLoadCost
    + totalBalanceCost
    + wpmc * processMoveCost
    + wsmc * serviceMoveCost
    + wmmc * totalMachineMoveCost;

minimize obj;
```

Qualification results

instance	variables	binaries	solution	best
A1-1	6,020	400	44,306,501	44,306,501
A1-2	1,812,044	100,000	787,434,004	777,532,896
A1-3	1,423,438	100,000	583,014,803	583,005,717
A1-4	753,404	50,000	272,304,480	252,728,589
A1-5	229,213	12,000	727,578,410	727,578,309
A2-1	1,415,324	100,000	5,934,529	198
A2-2	3,769,381	100,000	1,163,672,839	816,523,983
A2-3	3,843,977	100,000	1,555,764,432	1,306,868,761
A2-4	1,537,771	50,000	2,089,185,551	1,681,353,943
A2-5	1,556,017	50,000	575,691,649	336,170,182

100-lines model, 1 day of work,
11 million solutions explored in 5 min

LocalSolver qualified (25/80)

The EURO/Roadef Challenge

For the B instances ?

Boolean model has its limits

- With 4GB of RAM, LocalSolver tackles B1, B2 & B3 instances
- For other instances, a machine with 40GB of RAM is required

Solution : decompose the model

- Take a subset of machines (20.000 decisions)
- Optimize with LocalSolver on this subset for 1 second
- Repeat the operation 300 times

Same model, one more day of work

15 million solutions explored in 5 min

Final stage results

instance	machines	processes	LS 2.0 direct	LS 2.0 based
B1	100	5,000	4,443,248,534	3,997,678,428
B2	100	5,000	1,368,865,436	1,163,729,413
B3	100	20,000	351,813,894	266,280,383
B4	1,000	10,000	5,796,304,487	4,682,013,089
B5	100	40,000	1,048,102,941	1,015,121,228
B6	200	40,000	9,537,599,318	9,550,921,033
B7	4,000	40,000	RAM exploded	16,340,742,734
B8	100	50,000	1,323,157,749	1,316,777,967
B9	1,000	50,000	RAM exploded	15,959,363,471
B10	5,000	50,000	RAM exploded	19,314,990,649

LocalSolver

Black-box local search for combinatorial optimization