www.localsolver.com

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

A solver aligned with enterprise needs

• Handle highly combinatorial problems
• Solve problems with millions of decisions
• Provides high-quality solutions in seconds
• Proves optimality when possible (best effort)

A solver aligned with practitioner needs

• « Model & Run »
  • Simple mathematical modeling formalism
  • Direct resolution: no need of complex tuning
• A simple and transparent pricing
LocalSolver

New-generation solver

- **Computing good-quality solutions by local search**
- Scalable (each iteration done in sublinear time)
- Able to optimize in nonconvex and nonsmooth spaces
- Handle multiple nonlinear objective functions
- Computing lower bounds separately (relaxation, inference, cuts)

Portable software

- An innovative modeling language for fast prototyping
- Light object-oriented APIs: C++, Java, .NET
- Fully portable: Windows, Linux, Mac OS (x86, x64)
LocalSolver technology

Autonomous local search

• Generic moves based on ejection chains
• Learning of effective moves
• Sublinear evaluation of moves
• Easy to tune simulated annealing

Efficient implementation in C++

• Multithreaded search
• Incremental algorithmic
• Memory management
LocalSolver

Quick tour
Quadratic Knapsack

8 items to pack in a sack: maximize the total value of items while not exceeding a total weight of 102 kg

```plaintext
function model() {
    // 0-1 decisions
    x_0 <- bool(); x_1 <- bool(); x_2 <- bool(); x_3 <- bool();
    x_4 <- bool(); x_5 <- bool(); x_6 <- bool(); x_7 <- bool();

    // weight constraint
    knapsackWeight <- 10*x_0 + 60*x_1 + 30*x_2 + 40*x_3 + 30*x_4 + 20*x_5 + 20*x_6 + 2*x_7;
    constraint knapsackWeight <= 102;

    // maximize value
    knapsackValue <- 1*x_0 + 10*x_1 + 15*x_2 + 40*x_3 + 60*x_4 + 90*x_5 + 100*x_6 + 15*x_7
                      + 150 * x_0 * x_1 + 200 *x_3*x_4;
    maximize knapsackValue;
}
```

The user writes the model: nothing else to do!

declarative approach = model & run

LocalSolver
function model() {
    // 0–1 decisions
    x[0..7] <- bool();

    // weight constraint
    constraint knapsackWeight <= 102;

    // maximize value
    maximize knapsackValue;

    // secondary objective: minimize product of minimum and maximum values
    knapsackMinValue <- min[i in 0..7](x[i] ? values[i] : infinity);
    knapsackMaxValue <- max[i in 0..7](x[i] ? values[i] : 0);
    knapsackProduct <- knapsackMinValue * knapsackMaxValue;
    minimize knapsackProduct;
}

Nonlinear operators: prod, min, max, and, or, if-then-else, …

Lexicographic objectives
## Mathematical operators

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Logical</th>
<th>Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>prod</td>
<td>not</td>
</tr>
<tr>
<td>min</td>
<td>max</td>
<td>and</td>
</tr>
<tr>
<td>div</td>
<td>mod</td>
<td>or</td>
</tr>
<tr>
<td>abs</td>
<td>dist</td>
<td>xor</td>
</tr>
<tr>
<td>sqrt</td>
<td></td>
<td>if</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
function model() {
    // 0–1 decisions
    x[1..nbItems] <- bool();

    // weight constraint
    knapsackWeight <- sum[i in 1..nbItems](weights[i] * x[i]);
    constraint knapsackWeight <= knapsackBound;

    // maximize knapsack value
    knapsackValue <- sum[i in 1..nbItems](values[i] * x[i]);
    maximize knapsackValue;
}

Modeling APIs

C++

Java

C#
LocalSolver

Real-life applications
Car sequencing

Scheduling cars along an assembly line

- Each car requires some options
- Each option induces a ratio constraint \( P/Q \)
- A class is a set of cars requiring the same options

Objective: to space options over the line

- We wish no more than 2 sunroofs over 5 consecutive cars
- For any window of 5 cars, a penalty is computed as \( \max(n-2, 0) \) with \( n \) the number of cars requiring a sunroof
LSP model

\[ x_{cp} = 1 \iff \text{car of class } c \text{ is in position } p \]

\[
x[1..\text{nbClasses}][1..\text{nbPositions}] \leftarrow \text{bool}();
\]

\[
\text{for } [c \text{ in } 1..\text{nbClasses}]
\quad \text{constraint } \sum[p \text{ in } 1..\text{nbPositions}](x[c][p]) = \text{card}[c];
\]

\[
\text{for } [p \text{ in } 1..\text{nbPositions}]
\quad \text{constraint } \sum[c \text{ in } 1..\text{nbClasses}](x[c][p]) = 1;
\]

\[
\text{op}[o \text{ in } 1..\text{nbOptions}][p \text{ in } 1..\text{nbPositions}] \leftarrow \text{or}[c \text{ in } 1..\text{nbClasses} : \text{options}[c][o])(x[c][p]);
\]

\[
\text{nbVehicles}[o \text{ in } 1..\text{nbOptions}][j \text{ in } 1..\text{nbPositions}–Q[o]+1] \leftarrow \text{sum}[k \text{ in } 1..Q[o])(\text{op}[o][j+k–1]);
\]

\[
\text{violations}[o \text{ in } 1..\text{nbOptions}][j \text{ in } 1..\text{nbPositions}–Q[o]+1] \leftarrow \text{max}(\text{nbVehicles}[o][j] – \text{P}[o], 0);
\]

\[
\text{obj} \leftarrow \text{sum}[o \text{ in } 1..\text{nbOptions}][p \text{ in } 1..\text{nbPositions}–Q[o]+1]\text{violations}[o][p];
\]

\textbf{minimize} \text{obj};
Car sequencing at Renault

Additional constraints: no more than 10 consecutive cars with the same color

Decision variables remain the same

Additional objective: minimize the number of paint color changes

Problem posed by Renault as 2005 ROADEF Challenge
LocalSolver competitive with finalists (ranked 16/55)
Reassignment of processes to machines, with different kinds of constraints (mutual exclusion, resources, etc.)

More than 100,000 binary decisions

Only 1 day of work

LocalSolver in the final round (Challenge stream)
Roadmap

LocalSolver 2.1 (Released in July 2012)
- Support of arrays
- Memory & speed improvements

LocalSolver 2.2 (October 2012)
- Floating coefficients
- More nonlinear operators (exp, log, cos, sin, tan)
- New autonomous moves

LocalSolver 3 (current 2013)
- Continuous decision variables
- Lower bounds by relaxation

[jdarlay@localsolver.com](mailto:jdarlay@localsolver.com)

[http://www.localsolver.com](http://www.localsolver.com)

Meet us on exhibition booth #20