LocalSolver

mathematical programming by local search

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

```
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;
}
```

Binary decision variables

Integer intermediate variables

Quadratic expressions

The user writes the model: nothing else to do!

declarative approach = model & run

Multiobjective knapsack

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

```
function model() {
  // 0-1 decisions
  x[0..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;

  // 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;
}
```

Lexicographic objectives

Mathematical operators

Arithmetic		Logical	Relational
sum	prod	not	==
min	max	and	!=
div	mod	or	<=
abs	dist	xor	>=
sqrt		if	<
			>

Modeling APIs

```
function model() {
  // 0-1 decisi
  x[1..nbItems]

  // weight con
  knapsackWeig
  constraint kn

  // maximize
  knapsackValu
  maximize kna
}
```

C++

```cpp
#include "localsolver.h"

using namespace localsolver;

int ma
{
  int
  int
```

Java

```java
import localsolver.*;

public class Toy {
```

C#

```csharp
using System;
using localsolver;

public class Toy
{
  static void Main()
  {
    int[] weights = {10, 60, 30, 40, 30, 20, 20, 2};
    int[] values = {1, 10, 15, 40, 60, 90, 100, 15};

    LocalSolver localsolver = new LocalSolver();
    LSModel model = localsolver.GetModel();

    // 0-1 decisions
    LSExpression[] x = new LSExpression[8];
    for (int i = 0; i < 8; i++)
      x[i] = model.CreateExpression(LSOperator.Bool);

    // knapsackWeight <- 10*x0 + 60*x1 + 30*x2 + 40*x3 + 30*x4 + 20*x5 + 20*x6 + 2*x7;
    LSExpression knapsackWeight = model.CreateExpression(LSOperator.Sum);
    for (int i = 0; i < 8; i++)
      knapsackWeight.AddOperand(model.CreateExpression(LSOperator.Prod, weights[i], x[i]));

    // knapsackWeight <= 102;
    model.AddConstraint(model.CreateExpression(LSOperator.Leq

    // knapsackValue <- 1*x0 + 10*x1 + 15*x2 + 40*x3 + 60*x4 + 90*x5 + 100*x6 + 15*x7;
    LSExpression knapsackValue = model.CreateExpression(LSOperator.Sum);
    for (int i = 0; i < 8; i++)
      knapsackValue.AddOperand(model.CreateExpression(LSOperator.Prod, values[i], x[i]));

    // maximize knapsackValue;
    model.AddObjective(knapsackValue, LSObjectiveDirection.Maximize);

    // close the model before solving it
    model.Close();
    LSPhase phase = localsolver.CreatePhase();
    phase.SetTimeLimit(1);
    localsolver.Solve();
```

createExpression

addOperand

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

AB | A | B | AB | A | ABC | C | A | B | C

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

$$x_{cp} = 1 \Leftrightarrow \text{ car of class } c \text{ is in position } p$$

```
x[1..nbClasses][1..nbPositions] <- bool();

for [c in 1..nbClasses]
   constraint sum[p in 1..nbPositions](x[c][p]) == card[c];

for [p in 1..nbPositions]
   constraint sum[c in 1..nbClasses](x[c][p]) == 1;

op[o in 1..nbOptions][p in 1..nbPositions] <- or[c in 1..nbClasses : options[c][o]](x[c][p]);

nbVehicles[o in 1..nbOptions][j in 1..nbPositions-Q[o]+1] <- sum[k in 1..Q[o]](op[o][j+k-1]);

violations[o in 1..nbOptions][j in 1..nbPositions-Q[o]+1] <- max(nbVehicles[o][j] - P[o], 0);

obj <- sum[o in 1..nbOptions][p in 1..nbPositions-Q[o]+1](violations[o][p]);
minimize obj;
```

Car sequencing at Renault

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


```
color[p in 1..nbPositions] <- sum[c in 1..nbClasses](color[c] * x[c][p]);
same[p in 2..nbPositions] <- color[p] == color[p-1];
toolong[p in 11..nbPositions] <- and[j in p-10..p](same[j]);
for [p in 11..nbPositions] constraint not(toolong[p]);
```

Decision variables remain the same

Additional objective: minimize the number of paint color changes

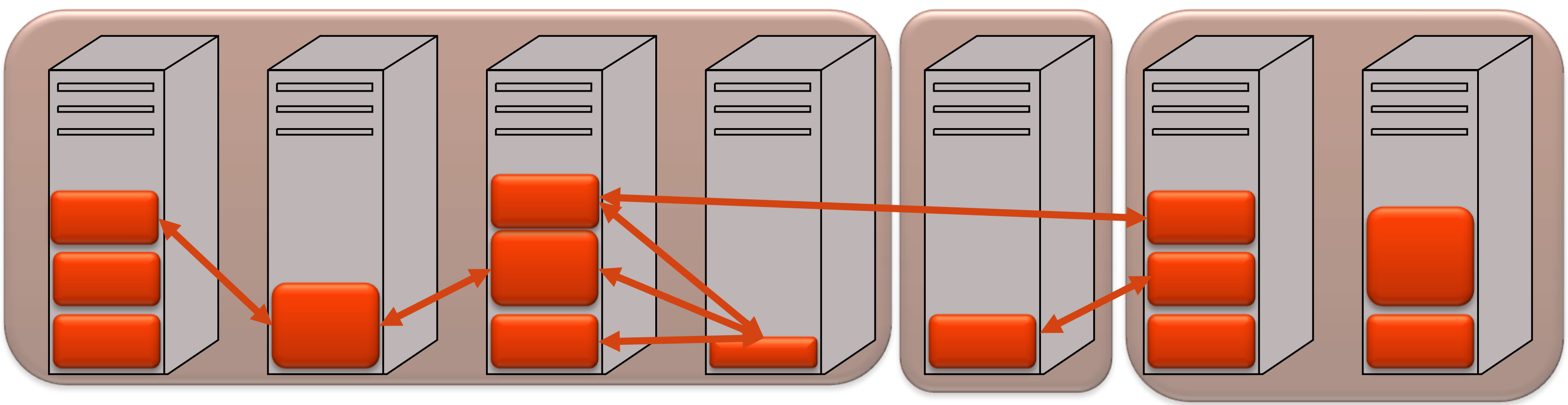
```
minimize sum[i in 2..nbPositions](not(same[i]));
```

Problem posed by Renault as 2005 ROADEF Challenge
LocalSolver competitive with finalists (ranked 16/55)

LocalSolver

2012 ROADEF/EURO Challenge

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

For more details

T. Benoist, B. Estellon, F. Gardi, R. Megel, K. Nouioua.
LocalSolver 1.x: a black-box local-search solver for 0-1 programming.
4OR, A Quarterly Journal of Operations Research 9(3), pp. 299-316.

jdarlay@localsolver.com

http://www.localsolver.com

Meet us on exhibition booth #20

LocalSolver