テクニカル Q/A：LocalSolver に関する質問／回答（Innovation24）

質問-1 ）
Is it possible the tuning of Search by add-on base?

回答）
You can tune the search by modifying parameters like IsAnnealingLevel or by adjusting the time allocated to each each objective function. But usually, reconsidering the model is the best way to improve performance. LocalSolver is quite similar to MIP from this point of view: as a *model and run* solver it does not requires implementing manually a search strategy as in CP solvers.

質問-２）
Does quick iteration of LocalSolver make fastest calculation in order to get the good solution?

回答 ）
LocalSolver has highly optimized algorithms for moving from a solution to another. Thanks to this algortihm LocalSolver can visit millions of solutions in a few minutes.

質問-３）
About the good modeling, (a)1st Step:Starting the relaxation of constraints, and then,(b)2nd Step: Taking the evaluation index into account the solution of satisfying constraints to get the suboptimal solution. If you have any father technical know-how than (a), (b) above, please teach us.

回答）
Maybe the most important point on modeling is the choice of the right set of decision variables.

質問-4）
LocalSolver has 0~9 of SA parameters.
What is the meaning of 0~9 levels?

回答）
Level 0 stands for ″pure descent″ (no deteriorating move is accepted). The higher the level the higher the frequency of accepted deteriorating moves. 9 is the highest allowed value.

質問-5）
From the viewpoint of SA (Meta-heuristics), LocalSolver also seems to have similar characteristics.
In the problem which has wide feasible area and has many good solutions, LocalSolver can find one of best solutions as quickly as possible.
Is this correct??
On the contrary, the problem which has narrow and/or incline the range of feasible solutions
is suited MIP Solver rather than LocalSolver.
Is this correct??

回答）
You views are quite correct. But below I give more insights about LocalSolver capabilities which will be very informative for you and your prospects.
LocalSolver includes some metaheuristics features. In particular, the search strategy is an adaptive simulated annealing. Moreover, the search is multithreaded. What we call ″adpative″ is that the search is automatically managed by a learning algorithm, in order to concentrate on the most promising regions of the solution space, with the most promising local-search moves.
Relying on a direct local search, LocalSolver is powerful to tackle problems which induce a large solution space with many feasible solutions. But this is a matter of modeling. Even if your problem is defined with very hard constraints and then a narrow solution space, you can model it in such a way to enlarge the solution space, mainly by relaxing some constraints and penalizing their violations in the objective functions. This is why we introduce the feature concerning multiobjective optimization (very appreciated by our users) .
In general, practitioners are quite reluctant to model problems with very hard constraints, because it induces risks of ″no solution found″ in operations. For convenience, they prefer to offer to user some solutions, even some business constraints are violated. The violations can show to users how modifying data to obtain feasible solutions. Indeed, in many cases, infeasibility came from bad or not sufficiently accurate data. That is why in many cases, business optimization models are very suited for LocalSolver.
In fact, our experience shows that good models for LocalSolver are also good models for the business.
In summary, this is more a matter of size and structure of the problem that your client has to tackle.
Here is a kind of decision tree in steps that we recommend to follow for the client:
1) If the problem is small (-> at most thousands of variables), then we recommend to use MIP solvers. They are generally able to find the optimal solution to the problem in short running times.

2) If the problem is medium (-> between 10,000 and 100,000 variables) and the problem is easy to write as a MIP (-> its structure is very linear), then we recommend to try MIP. If ineffective or too slow, then we recommend to use LocalSolver to boost the search for the first feasible solution. The translation from MIP to LSP is easy.

3) If the problem is medium but with many non linearities, or if the problem is large (-> more than 100,000 variables), then we recommend to use LocalSolver directly.

Note that in case 2), the modeling is so easy with LocalSolver that clients who are not focused on having a *proven* optimal solution now choose LocalSolver directly, in order to take no risk of ″no integer solution″ with MIP solver.

質問-6）
We have learned that LocalSolver under the severe constraints,
(1) 1st step: relax constraints and find a solution.
(2) 2nd Step: Within the constraint satisfaction solution, find the sub-optimal solution, referring to some evaluation index.
As for (1) above, we think relax most severe constraint and find the solution taking into the objective function referring the evaluation Index.
Is this recognition agreeable??

回答）
You have perfectly understood the way to use LocalSolver faced with severe constraints: to identify the difficult business constraints and to relax them by introducing a primary objective minimizing the violations of these constraints.
As explained above, we always try to explain to our clients that some constraints which are not likely to always satisfy in practice are in fact some primary objectives! This is just a matter of wording. So we recommend to identify the hard business constraints which are likely to be unfeasible in practice, and then to define the appropriate objectives to reach the feasibility over these business constraints.
LocalSolver supporting multiple linearly-ordered objectives, it is very easy to do.
By doing this kind of analyze with the client, we generally see that many of the *business* constraints are in fact preferences (that is, objectives in mathematical language) in practice. And these objectives can be classified according to their importance. What I call a business constraint here is a constraint which is not physical in some sense. For example, the constraint imposing that ″no two vehicles share the same position in the sequence″ can be viewed as a physical constraint according to me. On the other hand, the constraint imposing ″no subsequence of 10 or more vehicles with the same painting color in the sequence″ can be viewed as a business constraint: because this constraint is likely to be violated, even if it is in some rare extreme situations. But clearly it is a matter of modeling, and this work has to be done closely with the end-users.

質問-7）
We have learned that LocalSolver moves until it finds initial satisfaction solution under the constraints conditions. If LocalSolver can't find the solutions fitting for constraints, how LocalSolver moves in order to find initial solution??

回答）
The search for solutions is done in two steps in LocalSolver. First, LocalSolver searches for a feasible solution according to all the constraints defined by the user. Then, once this feasible solution is found, LocalSolver searches for an optimal feasible solution. We can say that LocalSolver first satisfies constraints, and then optimizes objectives (several objectives can be defined).
If constraints defined as hard in the LocalSolver are very hard to satisfy (inducing only a few feasible solutions), then LocalSolver is likely to spend time in finding such solutions. In some cases, LocalSolver may not find any feasible solution. Note that if at least one feasible solution exists, it is rare that LocalSolver does not find it. Generally, when LocalSolver does not find any feasible solution, this is because no feasible solution exists (due to bad data or a bad model). The user can look at the constraints which are not satisfied, if no feasible solution has been found by LocalSolver. In this way, he can understand the reasons of the infeasibility and relax some hard constraints into soft constraints (= objectives) to ensure feasibility.
If the hard constraints are not so hard to satisfy (that is, the problem is well modeled), then LocalSolver generally finds a solution in less than 1 second and starts optimization immediately. This is the good way of using LocalSolver.
Note that we have improved a lot the feasibility phase in LocalSolver 3.0. In this first phase, we use specific local-search moves and a surrogate objective based on constraint violations to quickly converge toward a feasible solution. Now LocalSolver is able to find feasible solutions even for hardly-constrained models. But we recommend users to avoid hard business constraints.
Because it always induces risk of ″no solution found″ for end-users, which is generally useless for people in operations.

質問-8）
About the console log and the resolution of the toy model.

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:¥Documents and Settings¥miyazaki>cd C:¥localsolver_3_0¥bin

C:¥localsolver_3_0¥bin>localsolver toy.lsp lsTimeLimit=3
LocalSolver 3.0 (Win32, build 20121129)
Copyright (C) 2012 Bouygues SA, Aix-Marseille University, CNRS.
All rights reserved (see Terms and Conditions for details).

Run model...
Close model...
Run solver...

Model:
  expressions = 38, operands = 50
  decisions = 8, constraints = 1, objectives = 1

Param:
  time limit = 3 sec, no iteration limit
  seed = 0, nb threads = 2, annealing level = 1

Objectives:
  Obj 0: maximize, no bound

Phases:
  Phase 0: time limit = 3 sec, no iteration limit, optimized objective = 0

Phase 0:

[0 sec, 0 itr]: obj = (0), mov = 0, inf < 0.1%, acc < 0.1%, imp = 0

[1 sec, 436874 itr]: obj = (280), mov = 881133, inf = 39.4%, acc = 43.9%, imp =14

[2 sec, 872730 itr]: obj = (280), mov = 1762132, inf = 38.9%, acc = 44.6%, imp = 14

[3 sec, 1034725 itr]: obj = (280), mov = 2088473, inf = 38.8%, acc = 44.8%, imp= 14

1034725 iterations, 2088473 moves performed in 3 seconds
Feasible solution: obj = (280)


C:¥localsolver_3_0¥bin>
```

回答）
1) What are expressions and operands? When declaring something like ″x <- sum(a, b, c)″ or like ″y <- a * b″, you declare a so-called expression. This corresponds to intermediate variable from your model. Expressions could be the decisions of your model, for example ″x <- bool()″. The fundamental difference between decisions and the other expressions is that by assigning the decisions, the values of all the other expressions of your model can be deduced. When declaring an expression, you provide its so-called operands. For example, the operands of ″x <- sum(a, b, c)″ are the expressions a, b and c. The number of expressions and operands are given in the LocalSolver log. Here you have 38 expressions, and the total number of operands is 50. Among these 38 expressions, we count 8 decisions. Only 1 expression is constrained, and only 1 objective is declared. These values give to the user (and to us) some insights about the size of his model, that is, of the input given to solve to LocalSolver. This is a kind of analog of the number of variables and nonzeros in MIP. LocalSolver is able to tackle some models involving millions of expressions on some basic computers, in minutes only.

2) The resolution process. ″itr″ means iterations. This is the number of iterations made by LocalSolver during the resolution process. Roughly speaking, this corresponds to the cumulated number of local search iterations performed. This number should be rather large. This is the force of LocalSolver: to perform millions of iterations per minute of running time. ″mov″ corresponds to the number of local search moves performed over all threads. When the number of thread is equal to 1, then this number is equal to the number of iterations. The number of moves corresponds approximately of the the number of threads multiplicated by the number of iterations.

At each move, a solution is visited by LocalSolver. But this solution may be infeasible according to the constraints declared by the user. ″inf″ gives you the percentage of infeasible solutions encountered during the search (that is, during the resolution process). More the problem is hardly constrained, more this number will be large, more difficult will be the search for LocalSolver. ″acc″ gives you the percentage of solutions accepted by LocalSolver during the search. When a move leads to a solution which is accepted by LocalSolver, this solution becomes the new incumbent for the next moves. So ″acc″ corresponds to the diversification rate of the local search process. Larger is this number, better is the search. ″imp″ is the number of strictly improving solutions found along the search. Here this number is 14, it means that we have found 14 solutions increasing strictly the objective given by the user. The number of improving solutions is not necessarily very large. We only provide the best solution found to the user, at the end of the search.

3) The resolution process in LocalSolver is highly randomized. Note that it does not mean that its results are not reproducible: by fixing the number of iterations, you can reproduce exactly the results of LocalSolver (without running time overhead, contrary to many MIP solvers). To make the search randomized, we use a pseudo-random number generator. This one is a linear congruential generator, based on a recurrence. The recurrence starts from a number, called ″seed″, which can be changed by the user. By changing the seed, you change a bit the walk of the resolution process, and so the results. Note

that running several threads allows to you to benefit from multiple parallel searches, each starting from a different seed. So to reproduce the results exactly from one run to another one, you have to fix: the seed and the number of iterations. By default, the seed is equal to 0.

4) LocalSolver uses simulated annealing to espace from local optima during his local search. This simulated annealing is adaptive in the sense that all its parameters are automatically tuned during the search, to reach the best behavior for the instance solved by the user. Nevertheless, we give to users a leverage to tune themselves the simulated annealing. This is the so-called ″annealing level″, which is an integer between 0 and 9. When equal to 0, the simulated annealing is totally desactivated, in the sense that no degrading solution is accepted during the search. In this way, the search looks like a standard descent heuristic. When equal to 9, the simulated annealing is fully activated, in the sense that a large number of uphill moves are accepted during the search. This allows to escape local optima and so to reach better solutions, but for larger running times. By default, this value is equal to 1.

質問-9)
Can you provide some examples of (a) fitting case, and also (b)unfitting case which have severe constraints, and difficult to get the satisfactory solution, for example?

回答)
The test cases that we sent to you a few weeks ago are good examples of fitting cases. An example of unfitting case is the controlled rounding problem which is a system of linear equalities for which LocalSolver can be beaten by MIP solvers, because they are designed for this type of purely linear integer programs.

質問-10)
How can LocalSolver take measure in the operational troubles
to get the same solution, if they run LocalSolver under the same conditions?

回答)
LocalSolver is fully deterministic. In particular if you set the number of iterations to 10 millions and launch LocalSolver several times, even on multiple threads, then you will always get exactly the same solution.

質問-11)
It says that LocalSolver 4.0 will support continuous variables.
Which does this4.0 mean, Integer variables or Real number variables? or both?

回答)
LocalSolver 4.0 will include Real number variables

質問-12)
We have learned that the fitting area for LocalSolver applications,
for example,
-Allotment
-Second Allotment
-Meeting coating
-Shift Scheduling
-Class Scheduling
Is this correct??

回答)
All these kinds of problems are very suited for LocalSolver. It is now the best model-and-run tool to tackle: assignment/allotment/allocation, partitioning, packing/grouping, covering, sequencing, or scheduling problems, in particular, when they are very large. Thanks to our very high-level modeling features, users can even try several models for a same problems (from very compact to very extended formulation). We will of course help you and your clients to find the better model for their problems. It is an important part of the LocalSolver. I think that your team and our teams of experts in both LocalSolver and Operations Research has an important plus-value for the client. We do not only sell an optimization product, we also sell an outstanding expertise in optimization

---

質問-13)
Can LocalSolver be designed to work under UNIX O.S like AIX??

回答)
We can compile LocalSolver binaries for some UNIX or AIX platforms. Indeed, LocalSolver is developed in ISO C++ language, with no use of third-party library. So it is fully portable on nearly all computing platforms. But it has to maintain a continuous integration (compiling, testing, etc.) of LocalSolver for all existing platforms. Since such platforms are used in big enterprises, we reserve this possibility to clients who have already paid or who are going to acquire for sure LocalSolver business licenses.
To ensure our capabilities in providing LocalSolver for any platform: we have recently done some tests

of LocalSolver on UNIX systems (Solaris operating system + SARC processors) with the CC compiler natively provided on this system. We had no problem. LocalSolver performs perfectly on such a platform.
Moreover, LocalSolver can fully profit from such platforms, through multithreading. As explained in my previous email, LocalSolver is ready for the many-core world. LocalSolver can be launched on big systems with dozen or even hundred cores with only a small memory (RAM) overhead. We have done the test for a prospect (Thales, the French company working for Defense) with a computer with 48 cores. It works perfectly.

質問-14)
LocalSolver3.0
It's shown in the website that Linux(32bit/64bit), Linux libc 2.3.2 or more, Libstdc++3.4 or more.
Now, we want to know more about ″From which version LocalSolver support,
Redhat, CentOS of Linux, for example??″

回答)
LocalSolver binaries are compatible for any Linux-based systems (RedHat, CentOS, SUSE, Ubuntu, Debian, etc.) which are equiped with the C Standard Library (libc) version 2.3.2 and the C++ Standard Library version 6.0.9. To be more concise, we can say that LocalSolver binaries are compatible with any Linux-based system equipped with the compiler GCC version 4.2 (released in 2007). Note that we have tested LocalSolver on all the Linux-based platforms mentioned above. LocalSolver runs perfectly, for both x86 and x64 processors.
LocalSolver is developed in ISO C++ and use only core C and C++ libraries. This is an important and nice feature of LocalSolver: it is lightweight and fully portable on almost any computing platform. We even have built and launched LocalSolver on a Smartphone equipped with an ARM processor! Therefore, we can provide LocalSolver binaries for older versions of C and C++ Standard Libraries, as well as for older versions of GCC compiler. But as for UNIX and AIX, we only provide them on demand, for clients.
We will make more and better communication on these aspects in the future (in commercial slides and on our web pages).
