How to migrate from MIP to LSP ?

Since LocalSolver offers operators `sum`, `product` and arithmetic comparisons, any integer program can be directly written in the LocalSolver modeling language, provided that integer variables are rewritten as a weighted sum of binary variables. However, such a model is often not suited for local search and a set of simple operations should be performed on your model in order to reach the highest performance.

Let us consider the car sequencing problem detailed in our *step by step example*. Here is the standard integer program for this problem, written in LocalSolver syntax, assuming that all variables have been defined beforehand:

MIP から LSP への移行方法
LocalSolver は演算子の合計、積、および算術の比較を提供するので、整数変数は 2 進変数の加重和として書き直されるならば、LocalSolver モデリング言語で直接書くことができます。しかし、そのようなモデルは、ローカル検索には適していないことが多く、最高のパフォーマンスを達成するために、モデルに対して簡単な操作を実行する必要があります。
私たちのステップで詳細に説明されているかごの順序問題を例に挙げてみましょう。すべての変数があらかじめ定義されていると仮定し、LocalSolver 構文で書かれたこの問題の標準整数プログラムを次に示します。

```
// A MIP-like MODEL FOR THE CAR SEQUENCING PROBLEM
for[c in 1..nbClasses]
  constraint sum[p in 1..nbPositions](cp[c][p]) == card[c];
for[p in 1..nbPositions]
  constraint sum[c in 1..nbClasses](cp[c][p]) == 1;
for[o in 1..nbOptions][p in 1..nbPositions]
  constraint op[o][p] >= sum[c in 1..nbClasses :
options[c][o]](cp[c][p]);
for[o in 1..nbOptions][j in 1..nbPositions-Q[o]+1]
  constraint nbVehicles[o][j] == sum[k in 1..Q[o]](op[o][j+k-1]);
for[o in 1..nbOptions][j in 1..nbPositions-Q[o]+1] {
```

```
  constraint violations[o][j] >= nbVehicles[o][j] -P[o];
  constraint violations[o][j] >= 0;
}

constraint obj == sum[o in 1..nbOptions][p in 1..nbPositions-
Q[o]+1](violations[o][p]);

minimize obj;
```

Decision variables and intermediate expressions

As explained in our *modeling principles*, the most crucial modeling decision is the choice of the set of decision variables. Here, the set of `cp[c][p]` is a good set of decision variables since the values of all other variables can be inferred from the values of `cp[c][p]`. Now intermediate variables can be written as such. For instance the constraint `nbVehicles[o][j] == sum[k in 1..Q[o]](op[o][j+k-1])` can be turned into an expression defining the term `nbVehicles[o][j] <- sum[k in 1..Q[o]](op[o][j+k-1])`.

Using non-linear operators instead of linearizations

Now we can observe that some equations in this model are in fact linearizations of arithmetic or logic expressions. For instance the constraint `op[o][p] >= sum[c in 1..nbClasses : options[c][o]](cp[c][p])` merely states that `op[o][p]` is equal to 1 as soon as one of the given terms is equal to 1 what is more naturally expressed as a `or`: `op[o][p] <- or[c in 1..nbClasses : options[c][o]](cp[c][p])`

Finally the pair of constraints on `violations[o][j]` are just the linear fashion of defining the maximum of a certain number of variables, what is directly written in LocalSolver as `violations[o][j] <- max( nbVehicles[o][j] -P[o], 0)`.

Similarly, all linearizations of a maximum, a condition or a conjunction should be translated into their direct LocalSolver formulation with operators `max`, `iif` and `and`. Piecewise linear function can be simply written as well. If your MIP contains a piecewise linear function (possibily with associated binary variables) making `Y` equal to `f(X)` such that on any interval `[c[i-1],c[i]]` we have `Y = a[i] * X + b[i]` with i in (1..3), then you will directly define Y as follows: `Y <- X < c[1] ? a[1]*X+b[1] : (X < c[2] ? a[2]*X+b[2] : a[3]*X+b[3]);`

After these transformations we obtain the following model:

意思決定変数と中間表現
モデリングの原則で説明したように、最も重要なモデリングの決定は、決定変数のセットの選択です。ここで cp [c] [p]の集合は、他のすべての変数の値が cp [c] [p]の値から推測できるので、決定変数の良い集合である。ここで中間変数はそのように書くことができます。例えば、nbVehicle [o] [j] == sum [k in 1..Q [o]] (op [o] [j + k-1]) という制約は、nbVehicles という用語を定義する式に変換できます] [j] < - sum [k in 1..Q [o]] (op [o] [j + k-1]) である。
線形化の代わりに非線形演算子を使用する
ここで、このモデルのいくつかの方程式が実際には算術式または論理式の線形化であることがわかります。 op [o] [p] = sum [1.nbClasses : options [c] [o]] (cp [c] [p]) の制約は op [o] [p]が指定された用語の 1 つが 1 に等しくなると直ちに 1 に等しい.nbClasses : options [c] [o]]の or : op [o] [p] < - または[c] (cp [c] [p])
最後に、違反[o] [j]の制約のペアは、違反[o] [j] < - max (nbVehicles [o] [j]) として LocalSolver に直接書き込まれる変数の最大値を定義する線形方法です。] [j] -P [o]、0) 。
同様に、最大、条件、または結合の線形化はすべて、演算子 max、iif、およびによって直接 LocalSolver 定式化に変換する必要があります。区分的線形関数も単純に書くことができます。あなたの MIP が任意の区間[c [i-1]、c [i]]に Y = a [i]を持つような Y を f (X) に等しくする区分的線形関数 (関連するバイナリ変数を持つ可能性がある) X + b [i]と i を (1..3) で比較すると、Y を次のように直接定義します。Y < - X <c [1]? a [1] * X + b [1] : (X <c [2]?a [2] * X + b [2] : a [3] * X + b [3]) ;
これらの変換の後、我々は以下のモデルを得る:

```
function model() {
  cp[1..nbClasses][1..nbPositions] <- bool();

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

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

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

  nbCarsWindows[o in 1..nbOptions][p in 1..nbPositions-
ratioDenoms[o]+1] <-
      sum[k in 1..ratioDenoms[o]](op[o][p+k-1]);

  nbViolationsWindows[o in 1..nbOptions][p in 1..nbPositions-
ratioDenoms[o]+1] <-
      max(nbCarsWindows[o][p]-ratioNums[o], 0);

  obj <- sum[o in 1..nbOptions][p in 1..nbPositions-
ratioDenoms[o]+1](nbViolationsWindows[o][p]);
  minimize obj;
}
```

Remove useless constraints

If your model contains valid inequalities they can (and should) be removed. Since LocalSolver does not rely on a relaxation of the model, these inequalities would only burden the model.

You must omit symetry breaking constraints as well: since LocalSolver does not rely on tree-based search, breaking symetries would just make it harder to move from a feasible solution to another one. Typically it could prevent LocalSolver from considering a nearby improving solution.

無用な制約を取り除く

モデルに有効な不等式が含まれている場合は削除できます（削除する必要があります）。 LocalSolver はモデルの緩和に依存しないため、これらの不等式はモデルに負担をかけるだけです。

symsry の拘束も省略する必要があります。LocalSolver はツリーベースの検索に依存しないため、symetry を破棄すると実行可能な解から別の解に移動するのが難しくなります。通常は、LocalSolver が近くの改善策を検討するのを妨げる可能性があります。