LocalSolver9.0
使用手引書

MSI 株式会社

2019/10/10

目次

１．　LocalSolver とは

LocalSolver9.0 は、大規模最適化問題を実用時間内で効率よく求めることを目的とした新しいアプローチの All-In-One ソルバーであり、従来の数理計画法システムを集大成したものとなっている。

LocalSolver はフランスの 6 人の若手 OR 実践者が 10 年の歳月をかけて開発したものであり、All-In-One　Solver として、MIP（混合整数計画法）、CP(制約論理プログラミング)、NLP（非線形計画法）、LP（線形計画法）問題を解くことができる。　特に、MIP、CP では、従来、現実的には解けない大規模最適化問題に対して、最新の各種解法を融合させて進化した解法で解くことを実現している。

LocalSolver の高機能、新機能については、
https://www.localsolver.com/docs/last/advancedfeatures/index.html
で詳しく説明してある。

２．　LocalSolver の実行方法（稼動確認）

LocalSolver9.0 は PC,　Unix,　MAC のそれぞれで 64bits モードで稼働する（32bits が必要なときには、MSI に連絡ください）。

LocalSolver は C++で開発されており、実行プログラムを単体で動かすだけでなく、C++、Java、C#、Python のプログラムから呼ぶことができる。

LocalSolver はインタープリター型で直接実行することができるモデリング言語（LSP）を持っている。インタープリター型で LocalSolver を実行する場合には、最新の関数型プログラミングをベースとした LSP ファイル（xxx.lsp）を作成し、パラメータとして LSP ファイルを指定するだけで実行可能である。

また、LocalSolver が提供するクラスライブラリを使うことで、効率よく、C++, Java,　C#（.net)、python　で、数理計画法システムのアプリケーションを開発することが可能である。

本ドキュメントでは、PC（64bits 版）で DOS のコマンドプロンプトから LocalSolver を直接実行する例を説明する。

LocalSolver を実行するためには、Localsolver をインストールし、ライセンスを獲得する必要がある。その手順については、以下を参照されたい。

https://www.msi-jp.com/localsolver/download/

2.1　LSP ファイルの実行

インストールされた内容を示す。

Localsolver_9_0　フォルダ
- ｜－　bin　　：実行プログラム localsolver.exe、Python 用実行ファイル
- ｜－　docs　　：説明書、C++, C#, Java 等のクラスライブラリの説明
- ｜－　examples：例題集（lsp 言語、C++、C#、Java、Python）
- ｜－　include
- license.dat　　：自分の PC 用のライセンスキーに置き換える。

1）実行手順

- DOS コマンドプロンプトを立ち上げる。
- LSP ファイルが入っているフォルダにディレクトリを移す。
  例）**cd C:¥localsolver_9_0¥examples¥toy**
- LSP ファイル名を指定して、localsolver.exe を実行する。
  例）**localsolver toy.lsp lsTimeLimit=1**
  ※LSP ファイル名：toy.lsp。
  ※パラメタ lsTimeLimit は実行時間を 1 秒と指定。

2）LSP ファイルの例

ここでは、examples¥toy の LSP ファイル:toy.lsp を示す。
Toy モデルは、ナップサック問題である。
品物が、８品あり、それぞれの重さと価値を以下に定義する。
重さ：10、60、30、40、30、20、20、2　　kg
価値：1、10、15、40、60、90、100、15　　円
ナップサックには最大 102kg まで品物を入れることができ、価値が最大になるよう、どの品物を選べばよいか、その時の価値はいくらになるかが問題である。
LSP による定式化を以下に示す（toy.lsp ファイルの内容）。

```
/********** toy.lsp **********/

/* Declares the optimization model. */
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;
    maximize knapsackValue;
}

/* Parameterizes the solver. */
function param() {
    lsTimeLimit = 1;
}
```

※赤字は予約語。この例では、model()　関数で、データと制約式、目的関数を定義している。
※bool()が意志決定変数を意味し、bool()で定義された変数の０と１の値の組合せを高速に評価することで、最適解を求める。

2.3　実行結果

2.2 で示した toy モデル（toy.lsp ファイル）の実行結果を以下に示す。

```
C:¥localsolver_9_0¥examples¥toy>localsolver toy.lsp lsTimeLimit=1
LocalSolver 9.0.20190727-Win64. All rights reserved.
Load toy.lsp...
Run model...
Preprocess model 100% ...
Run param...
Run solver...
Initialize solver 100% ...
Push initial solutions 100% ...

Model:    expressions = 38, decisions = 8, constraints = 1, objectives = 1
```

Param: time limit = 10 sec, no iteration limit

[optimization direction] maximize

[0 sec, 0 itr]: obj = 0
[0 sec, 5165 itr]: obj = 280

5165 iterations performed in 0 seconds

Optimal solution:
  obj = 280
  gap = 0%
  bounds = 280

C:¥localsolver_9_0¥examples¥toy>

※ちなみに答えは、
x_2(30, 15)， x_4(30, 60)， x_5(20, 90)， x_6(20, 100)， x_7(2, 15)の５個であり、重さの合計は 102kg、価値の合計は 280 円となる。
添付資料４に toy2.lsp として、output function を使って答えを出力した例を示す。

実行ログには、以下が示される。

－ 5165 iterations performed in 0 seconds

－ Optimal solution:
  － obj = 280 ：目的関数値（最適解）
  － gap = 0% ：上界値とのギャップ（最適解であることを示す）
  － bounds = 280 ：上界値

計算終了時（時間指定等で打ち切られた場合等）の最終解状態を、以下の３つで出力する。

- infeasible ： 実行不可能解
- feasible ：実行可能状態
- optimal ：最適解

LocalSolver9.0 では、解空間が凸の場合には、解の上界または下界を計算し、最適解とのギャップを表示する。

３．　LocalSolver による定式化

LocalSolver の定式化は意志決定変数を定義することから始まる。

意志決定変数には(bool)として定義した 0-1 変数、上下限を持つ整数変数(int)、上下限を持つ実数変数(float)、変数セット（list）として定義する変数セットの組合せからなり、意思決定変数の値を変化させ、超高速に解の探索を試行することで、大規模最適化問題を実用的に解くことが基本の考え方である。

bool 変数の数がたとえ 1000 万変数を超えても実用的な意味で解を求めることができる。bool 変数で定義した意志決定変数の組合せが解となる。解を構成する変数を一組だけ変化させ解を探索していく。ため、LocalSolver 用に定式化するためには、意思決定変数を使って、目的関数、制約関数を定義する必要がある。

ナップサック問題のように条件に合う品物を選ぶのが目的であれば、品物を選ぶか選ばないかを bool 変数として定義すれば良い。ある品物が選ばれた場合、bool 変数が 1 を取ることを想定すれば、品物の重さ及び価値を直接計算できるため、bool 変数の組合せで制約条件、目的関数を評価することができる。

LocalSolver9.0 では、従来の MIP 問題のように、上下限のある整数変数を意思決定変数として定義することができる。最適化計算途中にマテリアルバランスを求め、どこで何をいくつ作る／使うかを答えとして求めることができる。この場合、拠点配置や設備投資を決める固定費問題として、生産数量等の変数を float 変数として定義し、工場数を制限する場合生産数量を決める意志決定変数の数を LSP で制約条件として定義すればよい（従来は bigM を使用して関連付ける必要があったが技巧的な定式化は不要である）。

※ただし、既存の MIP 問題の定式化で工場制約と倉庫制約を別々整数変数で定義していた場合には、解探索が必ずしも速くならないため、LocalSolver に適した問題定義が重要である。

3.1　LSP モデリングの考え方

以下の手順でモデリングを行う。

１）**意志決定変数(bool 変数及び float 変数等)を定義**する。

２）**上記の意思決定変数をすべて使い、制約条件、目的関数を定義する。**

この時、MIP のように、**線形制約にこだわる必要はなく、制約条件、目的関数ともに、非線形表現が可能**である。

3.2　LSP での bool 変数の定義

以下に典型的な問題毎に bool 変数の定義イメージを示す。

－　ナップサック問題：Xp　　（ｐ：品物）
－　ルート選択問題：　Xr　　(ｒ：ルート)
－　裁断計画問題：　　Xp,q　（ｐ：裁断パターン、ｑ：パターンの使用回数）
－　人員配置問題：　　Xp,t,j　(p：人員、ｔ：時間、ｊ：ジョブ）
－　車両投入計画：　　Xc,p　（ｃ：車両、ｐ：ポジション（順番））
－　SCM：　　Xt,i,j,k,p (t：期、i：工場、j：ライン、k：倉庫、p：製品)
－　スケジューリング：Xt,i,j,p　(t：時間等、i：工場、j：ライン、p：製品等)

,

LocalSolver では数百万以上の bool 変数を定義することができ、MIP の定式化と違い、オーダ単位に意思決定変数(bool)を定義することで、より、自然な形での定式化が可能となる。

LocalSolver89.0 は、最初に事前解析で実行可能解や上界値（下界値）を求めるため、利用者が実行不可能性またはび実行可能性を考慮する必要はない。

3.3　LSP モデル

LSP モデルは、以下の要素から構成される。

- 意志決定変数：**bool()、float**（下限値、上限値）、**int**（下限値、上限値）、**list()**
- 副生変数:任意の変数であり、プログラミングをわかりやすくすることができる。変数の定義には、**<－**　を使用する。
- 制約：**constraint**（予約語）で、制約条件を定義する。
  constraint 制約式で定義された値が実行可能性の判定で使用される。
- 目的関数：**minimize**（予約語）または **maximize**（予約語）で目的関数を定義する。目的関数は複数定義可能であり、定義された順番に最適化を行う。目的計画法として利用可能である。

４．　LSP 言語

LSP 言語は、最適化問題をモデル化し、モデルの検証及び解の検証を行うフェーズでの試行錯誤を行うのに最適な環境を提供することを目的として開発されている。

LSP言語は、最新の関数型プログラミング言語である。関数型プログラミング言語の特長は、型推論を備えた言語であるため、JavaやC言語と異なり、コンパイラが自動的にデータの種類を推定するため、データの種類（型）をプログラマが指定する必要がない。その結果、プログラムの記述はRubyなど軽量言語のように簡潔である。
軽量言語では実現できないコンパイラによるプログラムのチェックが可能になる点にある。

LSP言語の特徴は以下：
－迅速に開発できる（開発生産性が良い。従来に比べて、1/5から1/2の開発量）
－バグを抑えやすい（コンパイラが型の間違い等を自動的にチェックする）
－アプリケーションの性能を向上させやすい
－簡潔かつシンプルなモデリング言語（できるだけ省略できるよう設計）
※大規模問題でも制約条件及びデータがそろっていれば、1 日でモデリングと実行が可能である。
－作成(修正)←→実行が同時にできる（一つはエディタ、もう一つは DOS コマンドプロンプトの二つのウインドウを操作しながら開発が可能である。
－目的計画法のように目的を段階的に設定することができるため、モデルの開発及び解の検証を段階的に行うことができる。

添付資料３に lsp 言語と C++、Java、C#で記述した例を示す。

4.1　Function

LSP にはメインプログラムがなく、以下の５つの基本的なファンクションからなる。
function model()は必須であるが、その他は必要に応じて使用すればよい。
また、以下の基本的なファンクションを使用することができる。

- `input`: for declaring your data or reading them from files.
- `model`: for declaring your optimization model.

- `param`: for parameterizing the local-search solver before running.
- `display`: for displaying some info in console or in some files during the resolution.
- `output`: for writing results in console or in some files, once the resolution is finished.

4.2 演算子

LSP モデルの中で、自由に使用できる。とくに、目的関数、制約条件の記述に利用でき、非線形制約、非線形目的関数として利用可能である。

演算子には、以下の種類がある。詳細は一覧表を参照されたい。

- 算術演算子（sum、min、max、sin、cos、log、exp 等）
- 論理演算子（not、and、or、xor）
- 関係演算子（==、 !=、<=、>=、<、>）
- 複合演算子（if、array+at）

4.3 言語機能

1）変数定義

変数の定義の例を示す。以下はすべて有効である。

```
a = true;   // a = 1
b = 9;      // b = 9

c = a + b;  // c = 10
c = a * b;  // c = 9

c = a == b; // c = 0
c = a < b;  // c = 1
```

2）配列定義

LocalSolver の配列は　map　で定義することができる。

Map は、値とキーを併せ持ったデータ構造になっている。 キーは、整数であり、かならずしも連続的である必要はない。値は、どんなタイプでもで可能であり、 キーに対応させるために、[ブラケット]表記法を用いる。

```
a = map("z", 9); // a[0] = "z", a[1] = 9
a = {"z", 9};    // a[0] = "z", a[1] = 9

a["a"] = "abc";  // a[0] = "z", a[1] = 9, a["a"] = abc
```

3）条件判定

条件判定は、if 文を使用する。記述形式は、以下：

if (C) S_true; else S_false;

または、？・：で簡潔に記述することもできる。

```
if (1 < 2) c = 3; else c = 4;
c = 1 < 2 ? 3 : 4;

if (0) c = "ok";
if (true) c = "ok";
if (2) c = "error"; // ERROR: invalid condition

c = 0 * 9; // c = 0
if (c) {
    a = "L";
    b = 0;
} else { // executed block
    a = "S";
    b = 1;
}
```

4）繰り返し

繰り返しには、while と for がある。

While は、以下で記述する。C が真である限り、S が実行される。

do S; while (C);

for は、以下で記述する。 v が V にある限り、S が実行される。

for [v in V] S;

また,キーと値がセットの場合には、以下で記述する。

for [k,v in M] S;

```
for [i in 0..2] a[i] = i + 1; // a[0] = 1, a[1] = 2, a[2] = 3
s = 0; for [v in a] s = s + v; // s = 6
s = 0; for [k,v in a] s = s + k + v; // s = 9

for[i in 0..9]
    for [j in i+1..9]
        for [k in j+2..9]
            a[i][j][k] = i + j + k;

for[i in 0..9][j in i+1..9][k in j+2..9] // compact
    a[i][j][k] = i + j + k;

for[i in 0..9][j in i+1..9][k in j+2..9]
{
    a[i][j][k] = i + j + k;
    b[i][j][k] = i * j * k;
}
```

5）繰り返し演算

繰り返し演算は、以下で記述する。

for [v in V] a[v] = f(v);

LSP では、以下の省略形で記述可能である。

a[v in V] = f(v);

```
for[i in 0..9][j in i+1..9][k in j+2..9]
    a[i][j][k] = i + j + k;

a[i in 0..9][j in i+1..9][k in j+2..9] = i + j + k;

x[i in 0..n-1][j in 0..m-1] <- bool();.
```

6）関数

LSP では、任意に関数を定義できる。関数の値は、0 (false)または 1 (true)でも良いし、数値でも良い。LSP プログラムは通常、function 間で共通の変数定義(global)になっているため、function 内でローカルに使用したい場合には、local の宣言子でローカル変数であることを定義する必要がある。

```
function isEven(v) {
  if (v % 2 == 0) return true;
  else return false;
}

function computeSumOfEvenNumbers(a,b) {
    local total = 0;
    for [v in a..b : isEven(v)]
        total = total + v;
    return total;
}
```

4.4　主なエラーメッセージ

コマンド・ラインで絶対に必要な引数は、lsp ファイルの名前です。もし lsp ファイルが 利用できないならば、エラーを出す。また、 コマンド・ラインの他の全ての引数（パラメータ等）は、フォーマット identifier=value を持たなければならない。

・<f> doesn't exist or is not accessible. // LSP file
・Invalid argument format for <arg>. Expected format：identifier=value.

LSP は型を強く意識した言語であり、関数のパラメータには正しい型が必要である。

・Function <f> cannot handle argument of type <t>. Argument of type <t2> is expected.
・Function <f> takes <x> argument(s) but <y> were provided.
・Function <f>：<T> expression expected for argument <i>.

同様に、型チェックで不適切な型の場合には、エラーメッセージを出力する。

・Cannot apply <opName> operator on type <T>.
・Cannot apply <opName> operator between types <T1> and <T2>
・Cannot cast <T1> to <T2>.
・Cannot apply ternary operator '?:' on given operators：incorrect argument type.
・Cannot cast 'nil' to <T>. A variable or a map element may not be assigned.

いくつかの関数は引数を持つ。もし、引数の数が合わない場合には、エラーを出力する。：

・Function <f> takes at least <x> argument(s) but <y> were provided.
・Function <f> takes at most <x> argument(s) but <y> were provided.

関数を使う時、関数が未定義であれば、エラーメッセージを出力する。また、既存の関数と同じ名前の関数を定義するのもエラーである。
変数にかんしては、自由に再定義（再利用）可能である。ただし、局所変数だけは、同じ名前で二回使うことはできない。 もし、変数が値を持たない場合は、nil の値を持つ。

・Function <f> already defined.
・Function <f> undefined.
・Variable <name> already defined.

Input/output 関数は、指定されたファイルの入出力チェックを行う。

・File <f> cannot be opened.
・Cannot read from file <f>.
・Cannot write to file <f>.

数字または文字列を入力時に、プログラムとデータが一致しない（データ数、タイプ等が一致しない）場合およびファイルの最後まで読んだ場合にはエラーを出力する。

・Cannot convert the current token to int.
・Cannot convert the current token to double.
・End of file: no more line to read from file <f>.
・End of file reached.

文字列の操作では、文字列が空でないこと及びインデックスが許容範囲内であることが必要である。

・The given index for substring is out of range. Min value: 0, Max value: <len>.
・Number of characters for substring must be greater than 0.
・Search string is empty.

マップの制限として二つある。

－キーは整数または文字であること

－イタレーション中（連続して探索計算している間）は、マップを変更してはならない:

・'nil' provided as key for a map. The key variable may not be assigned.
・Only types 'string' and 'int' are allowed for keys in maps.
・Cannot iterate on a modified map.

LSP モデルに対してパラメータで数値を指定する場合には、許容範囲の数値でなければならない。

・The objective bound must be an integer, a double or a boolean for objective <objIndex>
・The objective bound must be an integer or a boolean for objective <objIndex>
・The number of threads cannot exceed 1024.
・The annealing level size must be an integer between 0 and 9.
・Advanced parameter <key> does not exist.

モデルで式または変数を表現する場合には、<-　で宣言する必要である。.局所変数を <-を使って宣言することは出来ない。
モデルには、必ず目的関数がなければならない。また、目的関数は数式で表現する必要がある。
マップ等は使用できなう。
制約式は、バイナリ表現でなければならない。

・Cannot assign localsolver expressions to local variables.
・At least one objective is required in the model.
・Only boolean expressions can be constrained.
・Only expressions with a value can be added in the objectives list.

setValue 関数は、意志決定変数（bool 変数）にのみ初期値を与えることができる。

.

・The only allowed values are 0 or 1.

x <- a[y] の数式表現では、マップとしてゼロから連続した整数キーが必要となる。また、バリューとして、キーの数ぶんだけ、数値データまたはLS表現が必要である。

・All keys must be integers. Type found: <T>
・Values must be integers, booleans or expressions. Type found: <T>
・The first key must be 0. Key found: <key>
・Keys are not in a continuous range. Next key expected <key1>. Key found: <key2>.

探索の間に、　変数値が制約を満足しない場合が発生することに注意されたい。
例えて言えば、実行可能状態の時に、実行可能解を探索中に起きるケースがある。
ゼロ割や配列オーバーフローが起きた時であり、割算の分母がゼロまたはインデックスが範囲外になったことを意味する。 z <- x/y のような場合には、 z<-x/max(1,y)と表現することが望ましい。

・Division by zero.
・Index out of bounds for 'at' operator (index: <indexId>, array size: <n>

以上

【付録１】演算子一覧表

Operator	Description	Type	Arity	Symb	
Decisional	bool	Boolean decision: decision variable with domain {0,1}.	bool	0	
	float(lb, ub)	Float decision: decision variable with domain {lb,ub}.	double*	0	
	int(lb, ub)	Int decision: decision variable with domain {lb,ub}.	int	0	
	list	List decision: decision variable array .	list array	0	
Arithmetic	sum	Sum of all operands.	double*	n > 0	+
	prod	Product of all operands.	double*	n > 0	*
	min	Minimum of all operands.	double*	n > 0	
	max	Maximum of all operands.	double*	n > 0	
	div	Division of the first operand by the second one.	double*	2	/
	mod	Modulo: mod(a,b) = r such that a = q*b + r with q, r integers and |r| < b.	int	2	%
	abs	Asolute value: abs(e) = e if e >= 0, −e otherwise.	double*	1	
	dist	Distance: dist(a,b) = abs(a−b).	double*	2	
	sqrt	Square root.	double	1	
	cos	Cosine.	double	1	
	sin	Sine.	double	1	
	tan	Tangent.	double	1	

	log	Natural logarithm.	double	1	
	exp	Exponential function.	double	1	
	pow	Power: pow(a,b) is equal to the value of a raised to the power of b.	double	2	
	ceil	Ceil: round to the smallest following integer.	int	1	
	floor	Floor: round to the largest previous integer.	int	1	
	round	Round to the nearest integer: round(x) = floor(x+0.5).	int	1	
Logical	not	Not: not(e) = 1 – e.	bool	1	!
	and	And: equal to 1 if all operands are 1, and 0 otherwise.	bool	n > 0	&&
	or	Or: equal to 0 if all operands are 0, and 1 otherwise.	bool	n > 0	||
	xor	Exclusive or: equal to 0 if the number of operands with value 1 is even, and 1 otherwise.	bool	n > 0	
Relational	eq	Equal to: eq(a,b) = 1 if a = b, and 0 otherwise.	bool	2	==
	neq	Not equal to: neq(a,b) = 1 if a != b, and 0 otherwise.	bool	2	!=
	geq	Greater than or equal to: geq(a,b) = 1 if a >= b, and 0 otherwise.	bool	2	>=
	leq	Lower than or equal to: leq(a,b) = 1 if a <= b, and 0 otherwise.	bool	2	<=
	gt	Strictly greater than : gt(a,b) = 1 if a > b, and 0 otherwise.	bool	2	>
	Lt	Strictly lower than: lt(a,b) = 1 if a < b, and 0 otherwise.	bool	2	<
Conditional	iif	Ternary conditional operator: iif(a,b,c) = b if a is equal to 1, and c otherwise.	int*	3	? :

	at	**Access to an array: T[i] returns the ith value in array T.**	**int**	**2**	[ ]
Operator	**Description**	**Type**	**Arity**	**Symb**	
Decisional	**bool**	Boolean decision: decision variable with domain {0,1}.	bool	0	
Arithmetic	**sum**	Sum of all operands.	double*	$n > 0$	+
	prod	Product of all operands.	double*	$n > 0$	*
	min	Minimum of all operands.	double*	$n > 0$	
	max	Maximum of all operands.	double*	$n > 0$	
	div	Division of the first operand by the second one.	double*	2	/
	mod	Modulo: mod(a,b) = r such that $a = q*b + r$ with q, r integers and $\|r\| < b$.	int	2	%
	abs	Asolute value: abs(e) = e if $e >= 0$, $-e$ otherwise.	double*	1	
	dist	Distance: dist(a,b) = abs($a-b$).	double*	2	
	sqrt	Square root.	double	1	
	cos	Cosine.	double	1	
	sin	Sine.	double	1	
	tan	Tangent.	double	1	
	log	Natural logarithm.	double	1	
	exp	Exponential function.	double	1	

	pow	Power: pow(*a*,*b*) is equal to the value of *a* raised to the power of *b*.	double	2	
	ceil	Ceil: round to the smallest following integer.	int	1	
	floor	Floor: round to the largest previous integer.	int	1	
	round	Round to the nearest integer: round(*x*) = floor(*x*+0.5).	int	1	
Logical	**not**	Not: not(*e*) = 1 − *e*.	bool	1	!
	and	And: equal to 1 if all operands are 1, and 0 otherwise.	bool	*n* > 0	&&
	or	Or: equal to 0 if all operands are 0, and 1 otherwise.	bool	*n* > 0	||
	xor	Exclusive or: equal to 0 if the number of operands with value 1 is even, and 1 otherwise.	bool	*n* > 0	
Relational	**eq**	Equal to: eq(*a*,*b*) = 1 if a = b, and 0 otherwise.	bool	2	==
	neq	Not equal to: neq(*a*,*b*) = 1 if a != b, and 0 otherwise.	bool	2	!=
	geq	Greater than or equal to: geq(*a*,*b*) = 1 if a >= b, and 0 otherwise.	bool	2	>=
	leq	Lower than or equal to: leq(*a*,*b*) = 1 if a <= b, and 0 otherwise.	bool	2	<=
	gt	Strictly greater than : gt(*a*,*b*) = 1 if a > b, and 0 otherwise.	bool	2	>
	Lt	Strictly lower than: lt(*a*,*b*) = 1 if a < b, and 0 otherwise.	bool	2	<
Conditional	**iif**	Ternary conditional operator: iif(*a*,*b*,*c*) = *b* if a is equal to 1, and *c* otherwise.	int*	3	? :
	at	Access to an array: *T*[*i*] returns the *i*th value in array *T*.	int	2	[ ]

【付録2】予約語一覧

Modeling & Solving Parameter

- `lsTimeLimit = {10, 50};` Spends 10 (resp. 50) sec to optimize objective 0 (resp. 1).
- `lsTimeLimit = 60;` Corresponds to `lsTimeLimit = {0,..., 0, 60}`.
- `lsIterationLimit = {1000, 5000};` Spends 1000 (resp. 5000) iterations to optimize objective 0 (resp. 1).
- `lsIterationLimit = 6000;` Corresponds to `lsIterationLimit= {0,..., ,6000}`.
- `lsTimeBetweenDisplays = 5;` Displays info about the search every 5 sec (default: 1).
- `lsSeed = 9;` Sets pseudo-random number generator seed to 9 (default: 0).
- `lsNbThreads = 4;` Parallelizes the search over 4 threads (default: 2).
- `lsAnnealingLevel = 9;` Sets simulated annealing level to 9 (no annealing: 0, default:1).
- `lsVerbosity = 1;` Sets verbosity to 1 (no display: 0, default: 1).

Functions

Input & Output Library

- `f = openRead("data.in");` Opens file "data.in" in reading mode.
- `f = openWrite("data.out");` Opens file "data.out" in writing mode.
- `f = openAppend("data.out");` Opens file "data.out" in append mode.
- `close(f);` Closes the file.
- `eof(f)` Returns true if the end of file is reached.
- `i = readInt();` Prompt the user for an int (in the console standard input).
- `i = readInt(f);` Reads the next int parsed in file.
- `i = readDouble();` Prompt the user for a floating-point number (in the console standard input).
- `i = readDouble(f);` Reads the next floating-point number parsed in file.
- `s = readln();` Prompt the user for a line (in the console standard input).
- `s = readln(f);` Reads the next line of file.
- `s = readString(f);` Reads the next string parsed in file.
- `print("s = " + s + "¥n");` Prints the string in console.
- `print(f, "s = " + s + "¥n");` Prints the string in file.
- `println("s = " + s);` Prints the string followed by a line feed in console.
- `println(f, "s = " + s);` Prints the string followed by a line feed in file.
- `error(msg);` Prints an error message and exits.

Map Library

- `m = map(); m = {};` Creates an empty map.
- `m = map(9, "abc"); m = {1, "abc"};` Creates a map containing values 9, "abc" at keys 0, 1 respectively.
- `nbElems = count(m);` Counts the number of values in the map.
- `elems = values(m);` Returns the values of the map as a map.
- `indices = keys(m);` Returns the keys of the map as a map.
- `add(m, 123);` Adds 123 in the map with key equals to the largest integer key plus one.

String Library

- `i = toInt("123");` Converts the string into the corresponding integer (or throws an error if not possible).
- `i = toDouble("123.45");` Converts the string into the corresponding floating-point number(or throws an error if not possible).
- `m = split("a::b::c::d", "::");` Splits string "a::b::c::d" into substrings (as a map) according to the

separator "::".

- `s = trim("  abcd  ");` Removes white spaces at the beginning and at the end of the string.
- `len = length("abcd");` Returns the length of a string.
- `s = substring("abcd",1,2);` Returns a new string that is a substring of this string.There are two versions of this function: The first one takes two arguments : the string, and the start index of the substring. The second one takes 3 arguments : the string, the start index and the length of the substring.
- `b = startsWith("abcd","ab");` Returns true if the first argument starts with the specified prefix given as a second argument.If the second argument is the empty string, returns true.
- `b = endsWith("abcd","cd");` Returns true if the first argument ends with the specified suffix given as a second argument.
- `s = lowerCase("ABCD");` Returns a new string converted to lower case.
- `s = upperCase("abcd");` Returns a new string converted to upper case.
- `s = replace("abcd","bc","x");` Replaces each substring of a string that matches the literal target string with the specified literal replacement string. The replacement

proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaaaa" will result in "bba" rather than "abb".This function takes 3 arguments :subject string, searched sequence and replace sequence.

Modeling & Solving Library

- `getObjectiveBound(1);` Gets the bound of objective (with index) 1.
- `setObjectiveBound(1, 9999);` Sets the bound of objective 1 to 9999.
- `v = getValue(x);` Gets the value of modeling expression `x` in the best solution found by the solver.
- `setValue(x, 1);` Sets the value of `x` to 1 in the initial solution (or throws an error if `x` is not a decision).

【付録３】　言語比較

モデル名	lsp	C++	lsp 比	Java	lsp 比	C#	lsp 比	備考
Toy	17	49	2.9	46	2.7	48	2.8	ナップサック問題（トイモデル）
car_sequencing	77	211	2.7	211	2.7	246	3.2	車両投入順序問題
Knapsack	44	126	2.9	128	2.9	144	3.3	ナップサック問題
Maxcut	44	138	3.1	137	3.1	149	3.4	裁断計画問題
multiobj_knapsack	93	172	1.8	182	2.0	191	2.1	ナップサック問題（非線形最適化）
pmedian	66	154	2.3	154	2.3	169	2.6	OR-LIB。２点間の輸送コスト最小
平均比較	1.0		2.6		2.6		2.9	
socialgolfer	69							CSPLIB, problem 010
steel_mill_slab_design	99							CSPLIB, problem 038
google_machine	231							Google 問題（Google がスポンサ）
table_layout	243							表配置問題
nurse_rostering	505							看護師スケジューリング

【付録４】　サンプル２（lsp と実行結果）

付録4．１　lspプログラム（toy モデルに関数：output を追加）

```
/********** toy2.lsp **********/
function model()
{
   nbProducts = 8;
   value = {1,10,15,40,60,90,100,15};
// 0-1 decisions
   x[i in 0..nbProducts-1] <- 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];
   maximize knapsackValue;
}

function output()
{
   println("Selected Products:");
   for [i in 0..nbProducts-1 : getValue(x[i]) == 1]
      println("#"+i+" ("+value[i]+")");
}
```

付録4．２　lsp実行結果（toy モデルに関数：output を追加）

```
C:¥localsolver_9_0¥examples¥toy>localsolver toy2.lsp lsTimeLimit=1
LocalSolver 9.0.20190727-Win64. All rights reserved.
Load toy2.lsp...
Run model...
Preprocess model 100% ...
Run solver...
Initialize solver 100% ...
Push initial solutions 100% ...

Model:     expressions = 38, decisions = 8, constraints = 1, objectives = 1
Param:     time limit = 1 sec, no iteration limit

[ optimization direction    ]                maximize

[    0 sec,           0 itr]: obj       =                  0
[    0 sec,       7725 itr]: obj       =                280

7725 iterations performed in 0 seconds

Optimal solution:
   obj       =                280
   gap       =                  0%
   bounds =                 280
Run output...
Selected Products:
#2 (15)
#4 (60)
#5 (90)
#6 (100)
#7 (15)

C:¥localsolver_9_0¥examples¥toy>
```

【付録5】 BNF Syntax（バッカス・ナウア記法）

BNF の表記は次のような導出規則の集合である。

```
<symbol> ::= <expression with symbols>
```

左辺の<symbol>は単一の記号である。また、<expression with symbols> は記号列、または選択を表すバーティカルバー「｜」で区切られた記号列であり、左辺の &code><symbol> の置換となるものを表している。なお、導出規則で使用された記号は「非終端記号」と「終端記号」に分類される。導出規則群の左辺に現れた記号は「非終端記号」と呼ばれ、いずれの導出規則の左辺にも現れなかった記号は「終端記号」と呼ばれる。

以下に LSP 言語の BNF Syntax を示す。.
※＜＞::＝＜＞を省略してある。

```
expression
    : arithm_expression
    | table_expression
    | range_expression
    ;
arithm_expression
    : primary_expression
    | lambda_expression
    | arithm_expression '||' arithm_expression
    | arithm_expression '&&' arithm_expression
    | arithm_expression '==' arithm_expression
    | arithm_expression '!=' arithm_expression
    | arithm_expression 'is' arithm_expression
    | arithm_expression 'is' 'nil'
    | arithm_expression 'is' 'int'
    | arithm_expression 'is' 'double'
    | arithm_expression '<' arithm_expression
    | arithm_expression '>' arithm_expression
    | arithm_expression '<=' arithm_expression
    | arithm_expression '>=' arithm_expression
    | arithm_expression '+' arithm_expression
```

```
    | arithm_expression '-' arithm_expression
    | arithm_expression '*' arithm_expression
    | arithm_expression '/' arithm_expression
    | arithm_expression '%' arithm_expression
    | arithm_expression '?' arithm_expression ':' arithm_expression
    | '+' arithm_expression
    | '-' arithm_expression
    | '!' arithm_expression
    | 'typeof' arithm_expression
    ;
primary_expression
    : identifier
    | 'true'
    | 'false'
    | 'nan'
    | 'inf'
    | 'nil'
    | string
    | integer
    | double
    | primary_expression '[' expression ']'
    | primary_expression '.' identifier
    | function_call
    | '(' expression ')'
    ;
lambda_expression
    : identifier '=>' block_statement
    | '(' function_identifier_list ')' '=>' block_statement
    | '(' ')' '=>' block_statement
    |  identifier '=>' arithm_expression
    | '(' function_identifier_list ')' '=>' arithm_expression
    | '(' ')' '=>' arithm_expression
    | 'function' '(' function_identifier_list ')' block_statement
    | 'function' '(' ')' block_statement
    ;
range_expression
```

```
    : arithm_expression '..' arithm_expression
    ;
table_expression
    : '{' '}'
    | '{' table_list '}'
    ;
table_list
    : expression
    | table_key '=' expression
    | table_key ':' expression
    | table_list ',' expression
    | table_list ',' table_key '=' expression
    | table_list ',' table_key ':' expression
    ;
table_key
    : string
    | identifier
    | integer
    | '-' integer
    ;
function_call
    : primary_expression '(' ')'
    | primary_expression '(' function_argument_list ')'
    | primary_expression variadic_compositor_list '('
function_argument_list ')'
    ;
function_argument_list
    : expression
    | function_argument_list ',' expression
    ;
variadic_compositor_list
    : '[' filter_iterator ']'
    | variadic_compositor_list '[' filter_iterator ']'
    ;
filter_iterator
    : identifier 'in' expression ':' expression
```

```
    | identifier ',' identifier 'in' expression ':' expression
    | identifier 'in' expression
    | identifier ',' identifier 'in' expression
    ;
statement
    : block_statement
    | assignment_statement
    | local_assignment_statement
    | local_statement
    | if_else_statement
    | for_statement
    | while_statement
    | dowhile_statement
    | continue_statement
    | break_statement
    | modifier_statement
    | throw_statement
    | trycatch_statement
    | function_call_statement
    | return_statement
    | ';'
    ;
block_statement
    : '{' '}'
    | '{' statement_list '}'
    ;
statement_list
    : statement
    | statement_list statement
    ;
assignment_statement
    : identifier assignment_operator expression ';'
    | identifier assignment_compositor_list assignment_operator
expression ';'
    ;
assignment_operator
```

```
    : '='
    | '<-'
    | '+='
    | '-='
    | '/='
    | '*='
    | '%='
    ;
assignment_compositor_list
    : assignment_compositor
    | assignment_compositor_list assignment_compositor
    ;
assignment_compositor
    : '[' filter_iterator ']'
    | '[' arithm_expression ']'
    | '[' range_expression ']'
    | '.' identifier
    ;
local_assignment_statement
    : 'local' identifier local_assignment_operator expression ';'
    | 'local' identifier assignment_compositor_list
local_assignment_operator expression ';'
    ;
local_assignment_operator
    : '='
    | '<-'
    ;
local_statement
    : 'local' identifier ';'
    ;
if_else_statement
    : 'if' '(' expression ')'
    | 'if' '(' expression ')' 'else' statement
    ;
for_statement
    : 'for' for_compositor_list statement
```

```
    ;
for_compositor_list
    : for_compositor
    | for_compositor_list for_compositor
    ;
for_compositor
    : '[' filter_iterator ']'
    | '[' range_expression ']'
    ;
while_statement
    : 'while' '(' expression ')' statement
    ;
dowhile_statement
    : 'do' statement 'while' '(' expression ')' ';'
    ;
continue_statement
    : 'continue' ';'
    ;
break_statement
    : 'break' ';'
    ;
modifier_statement
    : modifier expression ';'
    ;
modifier
    : 'minimize'
    | 'maximize'
    | 'constraint'
    ;
throw_statement
    : 'throw' expression ';'
    | 'throw' ';'
    ;

trycatch_statement
    : 'try' statement 'catch' '(' identifier ')' statement
```

```
    ;
function_call_statement
    : function_call ';'
    ;
return_statement
    : 'return' ';'
    | 'return' expression ';'
    ;
function_list
    : function
    | function_list function
    ;
function
    : 'function' identifier '(' function_identifier_list ')'
block_statement
    | 'function' identifier '(' ')' block_statement
    ;
function_identifier_list
    : identifier
    | function_identifier_list ',' identifier
    ;
use_section
    : use_statement
    | use_section use_statement
    ;
use_statement
    :  identifier ';'
    ;
start
    : TOKEN_END
    | function_list TOKEN_END
    | use_section TOKEN_END
    | use_section function_list TOKEN_END
    ;
```

以上