All-In-One を目指した LocalSolver数理計画法システムの最新動向

2015年7月24日

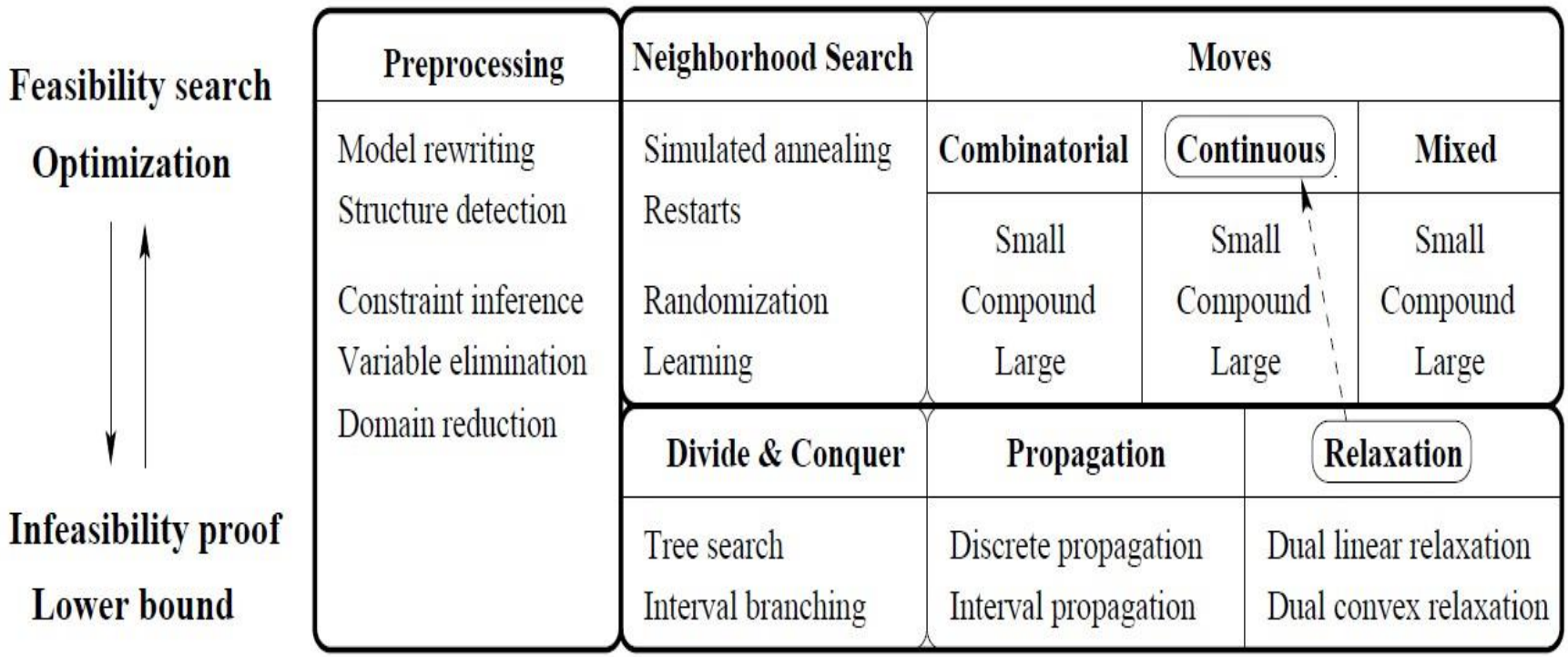
※宮崎　知明（MSI株式会社）

０．はじめに

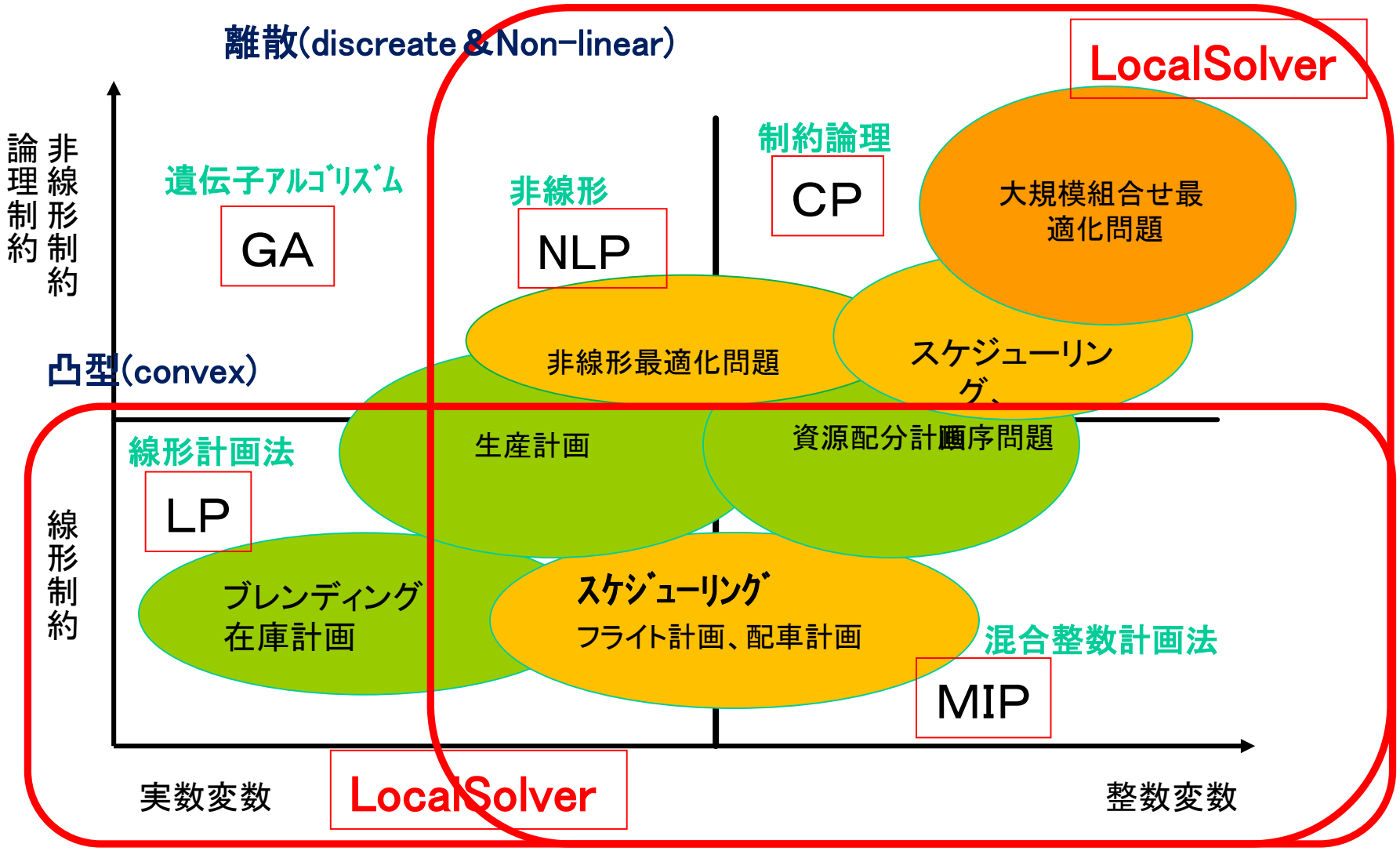
- SCMの浸透により、大規模最適化問題の解決がますます重要になっているが、数理計画法システムの限界で苦労しているのが、現状。
- 従来の方法（MIP：混合整数計画法）では解けない**大規模組合せ 最適化問題（800万以上のパターンの組合せから最適な組み合わせを求める）**を実用的に解くことができるようなソフトが出現。
- 従来の数理計画法システムの良い点と最新のIT技術を活用した**近傍探索による全く新しいハイブリッドな統合数理計画法システム（LocalSolver）**が実現された。
- どんな**大規模現実問題をも汎用的に直接モデリング**できる（**非線形制約、論理制約、非線形目的関数**）ようになった。

LocalSolver(様々なアーキテクチャを統合)

大規模混合整数変数、非凸最適化問題に対して、
既存の最適化技術(LS, LP/MIP, CP/SAT, NLP, ...)を統合した**All-In-One**の数理計画法システムを実現。



―LocalSolverの適用範囲―



MIPLIBでのベンチマーク結果

Instances	Status	Variables	LocalSolver 3.1	Gurobi 5.5	Cplex 12.4	Optimum
opm2-z10-s2	hard	6,250	* -25,719	-19,601	-18,539	-33,826
opm2-z11-s8	hard	8,019	* -33,028	-21,661	-18,883	-43,485
opm2-z12-s14	hard	10,800	* -46,957	-11,994	-36,469	-64,291
opm2-z12-s7	hard	10,800	* -46,034	-12,375	-30,887	-65,514
pb6	hard	462	-62	-62	-62	-63
queens-30	hard	900	-38	-36	-39	-40
dc11	open	37,297	11,100,000	21,300,000	1,840,402	unknown
ds-big	open	6,020	9,844	62,520	5,256	unknown
ex1010-pi	open	25,200	249	251	247	unknown
ivu06-big	open	1,812,044	* 479	9,416	678	unknown
ivu52	open	1,423,438	4,907	16,880	3,285	unknown
mining	open	753,404	* -65,720,600	902,969,000	no solution	unknown
pb-simp-nonunif	open	23,848	* 90	140	94	unknown
ramos3	open	2,187	* 223	274	267	unknown
rmine14	open	32,205	* -3469	-170	-968	unknown
rmine21	open	162,547	* -3657	-184	no solution	unknown
rmine25	open	326,599	* -3052	-161	no solution	unknown
siena1	open	13,741	256,620,000	315,186,152	54,820,419	unknown
sts405	open	405	342	342	354	unknown
sts729	open	709	648	648	665	unknown

□最少化問題
□実行CPU時間：5分、
□PC： Intel Core i7-820QM (4 cores, 1.73 GHz, 6 GB RAM, 8 MB cache)

LocalSolver で拡がるソリューション分野

従来、大規模問題では、探索空間が離散的であるもしくは離散的なもので表現できる問題は実用的に解けなかったが解くことができるようになった。

集合，順序，割当て，グラフ，論理，整数など離散的な構造を持つ。
多くの場合、**混合整数計画問題として定式化**できる.

現実世界の多くの問題は大規模組合せ最適化問題となる。

- **車両の優先順位付け（組立）**問題
- **裁断計画**問題 (フィルムなど)
- **SCM問題** (製造－輸送－在庫－販売など)
- **最短路問題** (カーナビのルート検索など)
- **ネットワーク**問題 (交通網，通信網，電気、ガスなどの設計)
- **配送計画**問題 (宅配便，店舗への商品配送，ゴミ収集など)
- **施設配置**問題 (工場，店舗，公共施設などの配置など)
- 人員**スケジューリング**問題 (乗務員・看護師の勤務表，時間割の作成など)
- 機械**スケジューリング**問題 (工場の運転計画、装置稼働計画など)

1． LocalSolverとは

- **Bouyguesは、フランスで最も大きい企業グループの1つ--収入は330億€/年**

- **Innovation 24は、ビジネス分析及び最適化を実現するBouyguesの子会社**

- **LocalSolverは、Innovation 24が開発、提供を始めたハイブリッド型統合数理計画法システム**

LocalSolverの機能

- **バイナリ(bool)**意思決定変数、**連続(float(下限、上限))** 意思決定変数及び**整数（int (下限、上限))** 意思決定変数でモデルを定義可能
- ローカルサーチを基本とし、MIP、LP、CP等の解法の利点を組み込んだハイブリッド型最適化エンジン
- 数百万以上の意思決定変数問題を実用時間（数分程度）で解く
- 見つけた実行可能解から近傍探索を開始し、最適解を目指すロジック

適用問題: supply chain optimization,
unit commitment,
portfolio optimization,
numerical optimization arising in engineering (ex: mechanics)

http://msi-jp.com/localsolver/ 参照

LocalSolver開発経緯

- 仏ブイグ社の最適化部門による2000年からの実績
（汎用化を試行2000-2005年）
- フランスの**ORチーム（准教授2人及び実践家4人）**による
2007年から研究開発プロジェクトで成果（国の支援）
- MIP（混合整数計画法）では解けない**大規模組合せ 最適化問題（800万以上の0-1整数変数）**を実用的に解く
- メタヒューリスティク解法（局所探索解法）をベースにシンプレックス法をも取り込んだ**全く新しい汎用解法**を実現
- 非線形制約、非線形目的関数まで拡張した問題を解く:
Mixed-variable non-convex programming
- 最新のIT技術、数理最適化技術を活用
（関数型プログラミング言語、問題解析機能、並列処理等）

LocalSolver（何故局所探索か）

局所探索は大規模最適化では、標準的な考え方

- 多くの教科書でも採用されているアルゴリズム
- 多くの一般的なメタヒューリスティックアプローチの利点を吸収
- 最悪の場合でも効率的でないことを結果として出力
- 現実的な時間で実践的な最適な解を提供

逐次改善法を実現

- 現在のソリューションの近傍を探索
- 小さい探索空間から徐々に必要な探索空間を拡大
  → 解空間全体から効率的に探索

局所探索による実用化及び汎用化を試行

LocalSolver（アルゴリズムの特長）

適切な探索領域の設定

- 探索空間の確率を高め、拡大していく
- 制約条件からではなく、目的関数に基づいて探索
- 現実の問題は良い探索空間を持つモデルになりやすい

局所探索:「基本」に帰って

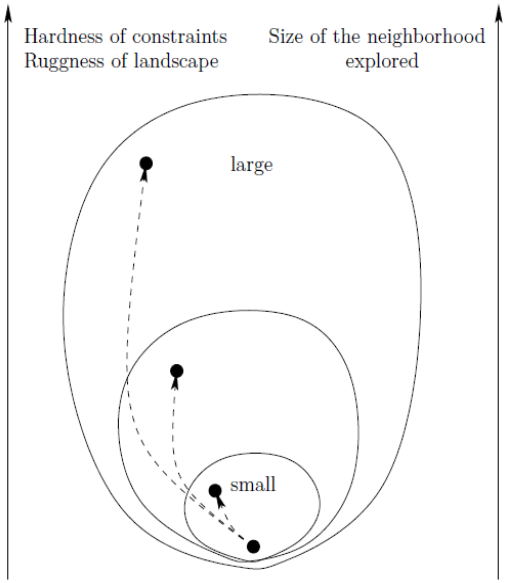
- 「メタ」に焦点を合わせない
- MOVE（イタレーション）の速さと効率性に集中
- クライアントからのフィードバックとテストが重要

→理論の専門技術とコンピュータ活用技術で高性能な近傍探索を実現（従来の全領域探索をせず）

LocalSolver（探索イメージ）

グローバル探索としての近傍探索

- 小さい近傍から探索を広げていく
- 近傍探索からダイナミックに探索領域を調整：縮小、拡大、特化
- 大規模領域探索にツリー探索(MIP、CP)を導入
- 完全な近傍と正確な探索では最適解に到達



2. LocalSolver5.5の新機能(1):

- LP and MPS 標準及び二次形式ファイルの直接入力.
- **GAMS** モデリング言語からの使用(**www.gams.com** 参照).
- **Python 2.7、3.2、3.4** 用のAPIを装備.
- **AMPL**からの利用

LocalSolver5.5の新機能(2):

- LSP 言語プログラムから解情報の取得: ***getSolutionStatus()*** 関数（**built-in variables and functions** を参照）.
- **Piecewise（区分線形）** 関数の導入による非線形最適化問題の実用化
- **List**意思決定変数の導入による大規模組み合わせ問題の強化(Set based modeling).

LP，MIP変換機能:

- **localsolver** foo.lp **lsTimeLimit**=10 **lsSolutionFile**=foo.sol
- **localsolver** foo.mps **lsTimeLimit**=10 **lsSolutionFile**=foo.sol

機能及び性能に関する改良ポイント:

- Improved **preprocessing**: try to reformulate your model to be suitable for LocalSolver resolution.
- Improved **detection and preprocessing** of global structures and constraints: knapsack subproblems and chronological flow subproblems.
- Improved **numerical accuracy** (without degrading performance) when dealing with floating-point expressions, through scaling techniques.
- improved **constraint propagation techniques** for the computation of lower bounds.
- Improved **constraint propagation techniques** integrated in neighborhood search.
- Improved **feasibility search process** to find a first feasible solution quicker.
- Improved **self-adaptive capabilities** of the neighborhood search heuristic to your problem.

３．Piecewise linear functions　の概要

Piecewise linear（区分線形）関数とは

- LocalSolver5.0から導入（MIPでのSOSセット等に対応）。
- 区分線形関数は端点間を直線で結んだ関数として表現できる。
- LocalSoverでは、区分線形関数を piecewise オペレータとして定義する。
- piecewise (x,y,z)オペレータは、x、y、z　の3つのパラメータを持つ：

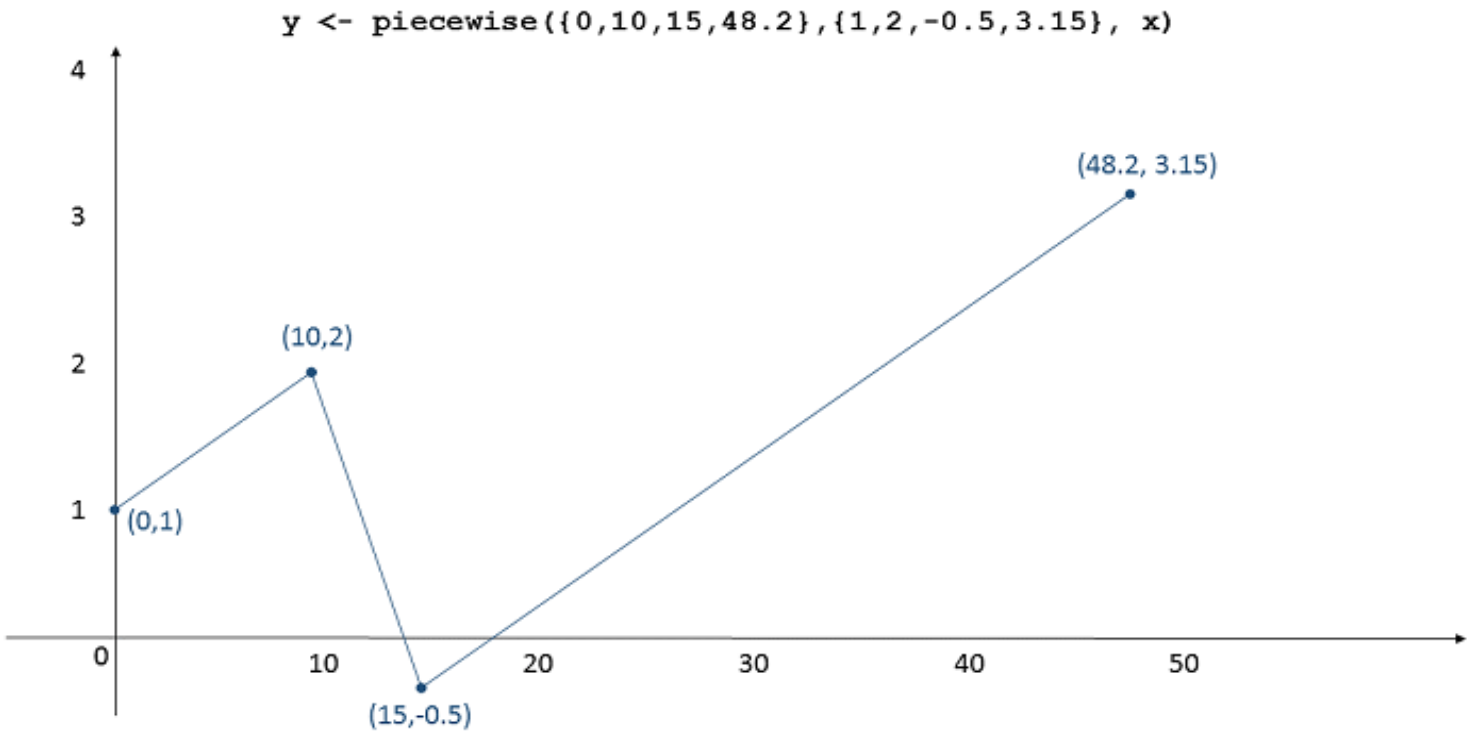
X：X軸の端点リストを定義する。
Y：Y軸の端点リストを定義する。
Z：区分線形関数内のXの値を定義する
　（対応するY軸の値が関数値としてもとまる）。

例題：piecewise({0,50,100},{0,10,100},75)　答え：55.

Piecewise関数の例題1.

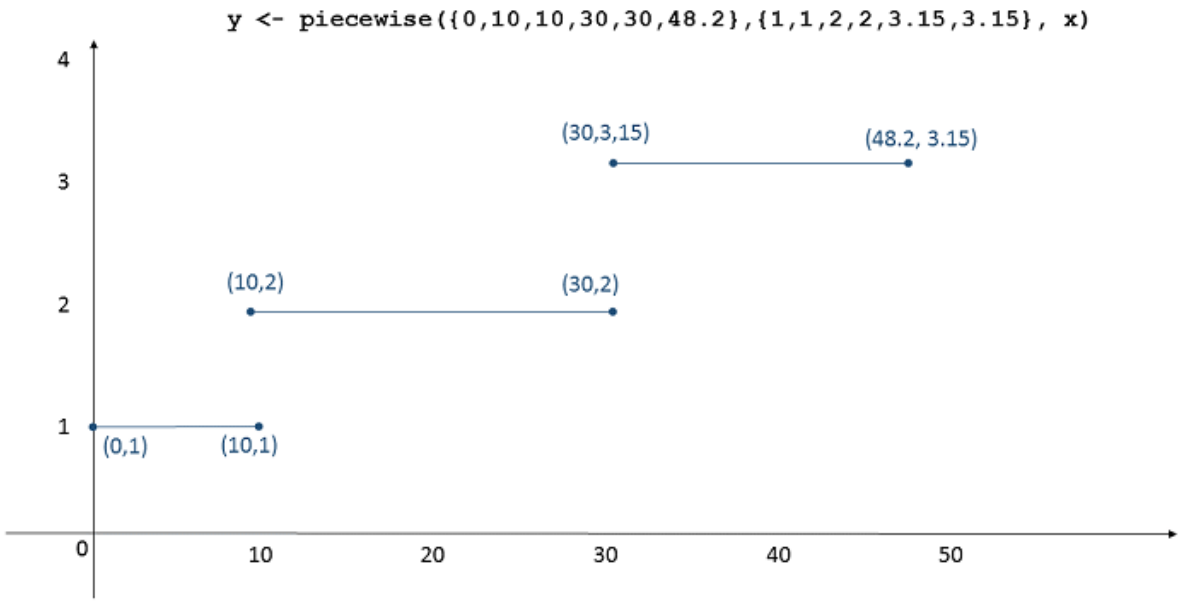
y <- piecewise({0,10,15,48.2},{1,2,-0.5,3.15}, x).

0 <= x <=48.2.



例題2.

y <- piecewise({0,10,10,30,30,48.2},{1,1,2,2,3.15,3.15}, x).



Piecewise関数を使った時間軸付TSPの例

- 都市数: nbCities =17; (0, 1, 2,・・・,16)
- 計画期間: nbMonths = 3; (0, 1, 2)
- 各都市の作業量
  citywork ={10,23,5.5,10,10,21,17,16,10,23,5.5,13,10,19,18,16,9};
- 各都市の作業終了期限
  cityduedate = {0,0,1,2,0,2,2,2,1,1,0,1,0,2,1,1,1};
- 各月の作業可能容量: monthCapa = {90,110,50};
- Piecewise関数の定義
  xlist = {0,90,90,200,200,250};
  ylist = {0,0,1,1,2,2};

TIMETSP.lsp(時間軸付TSP)の例（その１）

```
/********** tsp.lsp **********/
/* The input files follow the TSPLib "explicit" format.*/
function input() {
    usage = "¥nUsage: localsolver tsp.lsp "
        + "inFileName=inputFile [lsTimeLimit=timeLimit]¥n";
    if (inFileName == nil) error(usage);
    inFile = openRead(inFileName);
    stop = false;
    while (!stop) {
        str = readln(inFile);
        if (startsWith(str,"DIMENSION:")) {
            dim = trim(split(str,":")[1]);
            nbCities = toInt(dim);
            println("Number of cities = "+nbCities);
        } else if (startsWith(str,"EDGE_WEIGHT_SECTION")) {
            stop = true;
        }
    }
// distance from i to j is distance[i + nbCities*j]
    distanceWeight[0..nbCities*nbCities-1] <- readInt(inFile);
                citywork ={10,23,5.5,10,10,21,17,16,10,23,5.5,13,10,19,18,16,9};
                cityduedate = {0,0,1,2,0,2,2,2,1,1,0,1,0,2,1,1,1};
                nbMonths = 3;
                monthCapa = {90,110,50};
                xlist = {0,90,90,200,200,250};
                ylist = {0,0,1,1,2,2};
}
```

TIMETSP.lsp(時間軸付TSP) の例（その2）

```
/* This LocalSolver model is based on the assignment of a position to each city.
The length of the tour is computed as the sum of the distance between consecutive cities,
using the at operator.*/
function model() {

    // x[i][j] equal to 1 if city j is ith visited city in the tour
    x[0..nbCities-1][0..nbCities-1] <- bool();

    // one city per position i
    for [i in 0..nbCities-1] constraint sum[j in 0..nbCities-1](x[i][j]) == 1;
    // one position per city j
    for [j in 0..nbCities-1] constraint sum[i in 0..nbCities-1](x[i][j]) == 1;

    //city[i] is the city at position i in the tour
    city[i in 0..nbCities-1] <- min(sum[j in 0..nbCities-1](j*x[i][j]),nbCities-1);

    local maxCode = nbCities*nbCities-1;
    distanceCode[i in 1..nbCities-1] <- min(maxCode,city[i-1] + nbCities * city[i]);
    distanceCode[0] <- min(maxCode,city[nbCities-1] + nbCities * city[0]);

    // the distance of the arc reaching the ith city
    distance[i in 0..nbCities-1] <- distanceWeight[distanceCode[i]];
```

TIMETSP.lsp(時間軸付TSP) の例（その3）

```
// the work to be performed at the ith city
work[i in 0..nbCities-1] <- citywork[city[i]];
// the due month of the ith city
duedate[i in 0..nbCities-1] <- cityduedate[city[i]];
// the cumulated work for the ith city + previous cities
cumulatedWork[0] <- work[0];
cumulatedWork[i in 1..nbCities-1] <- work[i]+cumulatedWork[i-1];
// the completion month for the ith city (stepwise function)

month[i in 0..nbCities-1] <- piecewise(xlist,ylist,cumulatedWork[i]);

// detect if the ith city is visited too late

isLate[i in 0..nbCities-1] <- month[i] > duedate[i];
// the total traveled distance
totalDistance <-  sum[i in 0..nbCities-1](distance[i]);
// the number of cities visited too late
nbLate <-  sum[i in 0..nbCities-1](isLate[i]);
// a mix of the two criteria

minimize totalDistance + 3 * nbLate;
}
```

TIMETSP.lsp(時間軸付TSP) の例（その4）

```
function param() {
    if (lsTimeLimit == nil) lsTimeLimit = 10;
}

function output() {
                println("Distance = "+getValue(totalDistance)+", nbLate =
"+getValue(nbLate));
    for[i in 0..nbCities-1] print(getValue(city[i])+ " ");
                println();
                for[i in 0..nbCities-1] print(getValue(cumulatedWork[i])+ ",  ");

    println();
}
```

TIMETSP.atspデータの例

```
NAME:  TIMETSP↓
TYPE: ATSP↓
COMMENT: 17 city problem (Repetto)↓
DIMENSION:  17↓
EDGE_WEIGHT_TYPE: EXPLICIT↓
EDGE_WEIGHT_FORMAT: FULL_MATRIX ↓
EDGE_WEIGHT_SECTION↓
 9999    3    5   48   48    8    8    5    5    3    3    0    3    5    8    8    5↓
    3 9999    3   48   48    8    8    5    5    0    0    3    0    3    8    8    5↓
    5    3 9999   72   72   48   48   24   24    3    3    5    3    0   48   48   24↓
   48   48   74 9999    0    6    6   12   12   48   48   48   48   74    6    6   12↓
   48   48   74    0 9999    6    6   12   12   48   48   48   48   74    6    6   12↓
    8    8   50    6    6 9999    0    8    8    8    8    8    8   50    0    0    8↓
    8    8   50    6    6    0 9999    8    8    8    8    8    8   50    0    0    8↓
    5    5   26   12   12    8    8 9999    0    5    5    5    5   26    8    8    0↓
    5    5   26   12   12    8    8    0 9999    5    5    5    5   26    8    8    0↓
    3    0    3   48   48    8    8    5    5 9999    0    3    0    3    8    8    5↓
    3    0    3   48   48    8    8    5    5    0 9999    3    0    3    8    8    5↓
    0    3    5   48   48    8    8    5    5    3    3 9999    3    5    8    8    5↓
    3    0    3   48   48    8    8    5    5    0    0    3 9999    3    8    8    5↓
    5    3    0   72   72   48   48   24   24    3    3    5    3 9999   48   48   24↓
    8    8   50    6    6    0    0    8    8    8    8    8    8   50 9999    0    8↓
    8    8   50    6    6    0    0    8    8    8    8    8    8   50    0 9999    8↓
    5    5   26   12   12    8    8    0    0    5    5    5    5   26    8    8 9999↓
EOF↓
↓
```

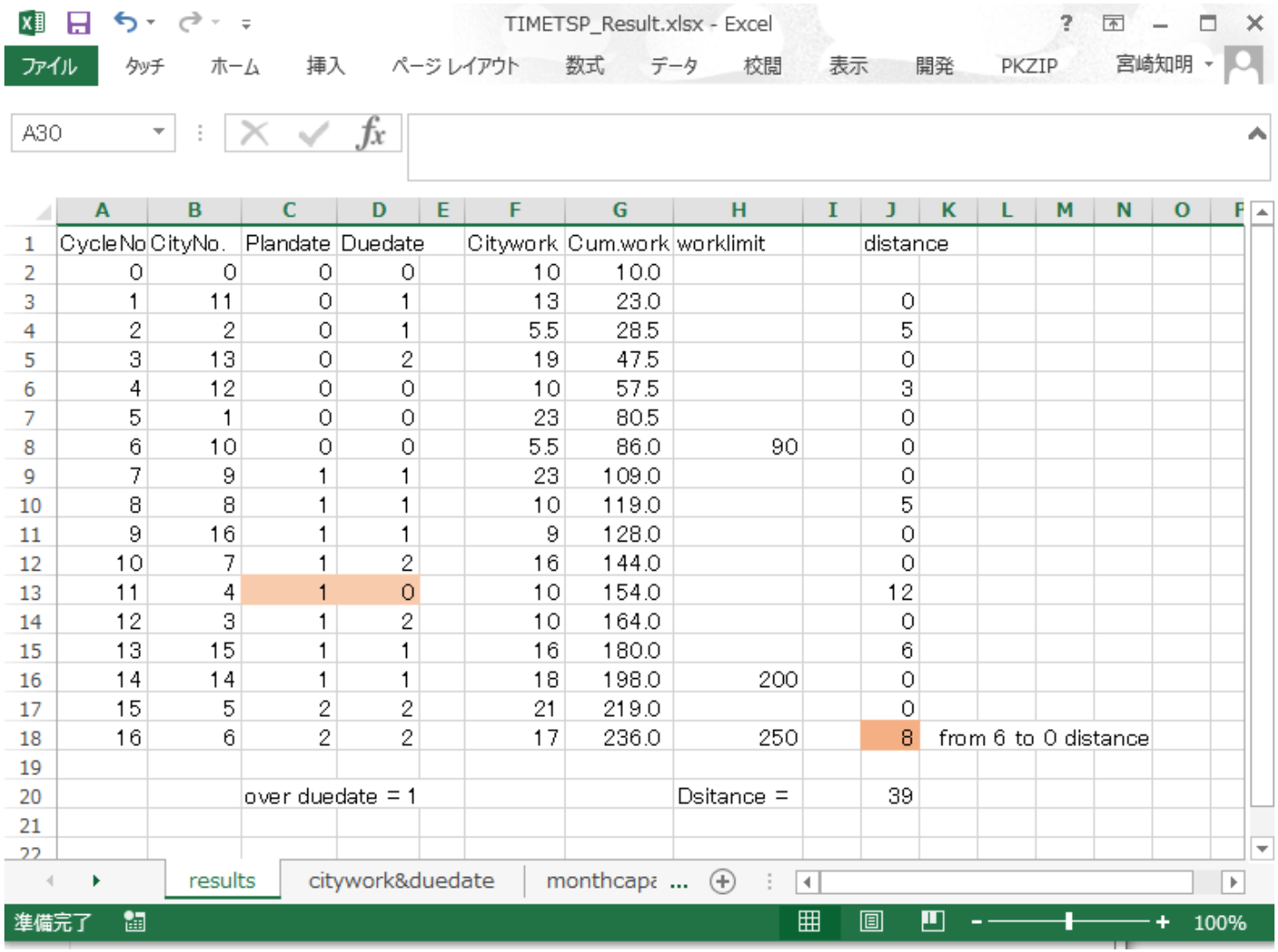
TIMETSP.lsp 実行結果（その1）

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
C:¥Users¥Miyazaki-MSI>cd C:¥localsolver_5_0¥examples¥TIMETSP

C:¥localsolver_5_0¥examples¥TIMETSP>localsolver TIMETSP.lsp
inFileName=TIMETSP.A TSP lsTimeLimit=1

LocalSolver 5.0 (Win64, build 20150119)
Copyright (C) 2015 Innovation 24, Aix-Marseille University, CNRS.
All rights reserved. See LocalSolver Terms and Conditions for details.

Load TIMETSP.lsp...
Run input...
Number of cities = 17
Run model...
Preprocess model 100% ...
Close model 100% ...
Preprocessing transformed 136 expressions.
Run param...
Run solver...
Initialize threads 100% ...
Push initial solutions 100% ...
```

TIMETSP.lsp　実行結果（その2）

```
Model:
  expressions = 858, operands = 2207
  decisions = 289 (bools = 289,  ints = 0, floats =  0),
  constraints = 34, objectives = 1, constants  = 34
Param:
  time limit = 1 sec, no iteration limit
  seed = 0, nb threads = 2, annealing level = 1
Objectives:
  Obj 0: minimize, bound = 0
Phases:
  Phase 0: time limit = 1 sec, no iteration limit, optimized objective = 0
Phase 0:
[  0 sec,       0 itr] : infeas :   68 violated expressions
[  1 sec,   30244 itr] : obj =          42
[  1 sec,   30244 itr] : obj =          42
30244 iterations, 60489 moves performed in 1 seconds
Feasible solution: obj =          42
Run output...
Distance = 39, nbLate = 1
0 11 2 13 12 1 10 9 8 16 7 4 3 15 14 5 6
10,  23,  28.5,  47.5,  57.5,  80.5,  86,  109,  119,  128,  144,  154,  164,  1
80,  198,  219,  236,
C:¥localsolver_5_0¥examples¥TIMETSP>
```

TIMETSP.lsp　実行結果（まとめ）



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	CycleNo	CityNo.	Plandate	Duedate		Citywork	Cum.work	worklimit		distance					
2	0	0	0	0		10	10.0								
3	1	11	0	1		13	23.0			0					
4	2	2	0	1		5.5	28.5			5					
5	3	13	0	2		19	47.5			0					
6	4	12	0	0		10	57.5			3					
7	5	1	0	0		23	80.5			0					
8	6	10	0	0		5.5	86.0	90		0					
9	7	9	1	1		23	109.0			0					
10	8	8	1	1		10	119.0			5					
11	9	16	1	1		9	128.0			0					
12	10	7	1	2		16	144.0			0					
13	11	4	1	0		10	154.0			12					
14	12	3	1	2		10	164.0			0					
15	13	15	1	1		16	180.0			6					
16	14	14	1	1		18	198.0	200		0					
17	15	5	2	2		21	219.0			0					
18	16	6	2	2		17	236.0	250		8	from 6 to 0 distance				
19															
20			over duedate = 1					Dsitance =		39					
21															

results　citywork&duedate　monthcapa ...

準備完了　100%

４．List意思決定変数セット（Set-based modeling)の概要

Set-based modeling(List意思決定変数セットとは)

Structured decisional operator **list(n)**

- listセットは {0, …, n-1} の値のサブセットである。
- 一つのセット内は異なる値を持つ。

List(n)に対するオペレータ:

- **count(u)** : number of values selected in the list
- **get(u,i) or u[i]** : value at index i in the list
- **indexOf(u,v)** : index of value v in the list
- **contains(u,v)** : "indexOf(u,v)!= -1" と同じ意味
- **disjoint(u1, u2, …, uk)** : list を切り離す
- **partition(u1, u2, …, uk)**:listを一つの物として見る

List意思決定変数セットを使ったTIMETSPの例

- **List**は意思決定変数を定義する新しい変数セット定義であり、制約論理プログラミング（CSP)のノウハウの一つでもあった。list意思 変数セットとして定義された意思決定変数セットは、0‐n までの整数値を順番に持つ。
- 都市City群を list 変数セットとして定義する。

```
x <- list(nbCities);
city[i in 0..nbCities-1] <- x[i];
```

- Bool変数でTSPを定式化する場合、from-to を指定するｎ×ｎのbool変数を定義する必要があるがperputation変数では、定義される変数をセットとして考慮するため、大幅な意思決定変数の削減と制約条件の計算を省くことが可能となる。
- TSPのように、N個の箱のなかに決められた変数を入れる場合に最適である。

permutation意思決定変数を使ったTIMETSP_new.lspの例

```
/********** tsp.lsp **********/
/* The input files follow the TSPLib "explicit" format.*/
function input() {
    usage = "¥nUsage: localsolver tsp.lsp "
        + "inFileName=inputFile [lsTimeLimit=timeLimit]¥n";
    if (inFileName == nil) error(usage);
    inFile = openRead(inFileName);
    stop = false;
    while (!stop) {
        str = readln(inFile);
        if (startsWith(str,"DIMENSION:")) {
            dim = trim(split(str,":")[1]);
            nbCities = toInt(dim);
            println("Number of cities = "+nbCities);
        } else if (startsWith(str,"EDGE_WEIGHT_SECTION")) {
            stop = true;
        }
    }
// distance from i to j is distance[i + nbCities*j]
    distanceWeight[0..nbCities*nbCities-1] <- readInt(inFile);
                    citywork ={10,23,5.5,10,10,21,17,16,10,23,5.5,13,10,19,18,16,9};
                    cityduedate = {0,0,1,2,0,2,2,2,1,1,0,1,0,2,1,1,1};
                    nbMonths = 3;
                    monthCapa = {90,110,50};
                    xlist = {0,90,90,200,200,250};
                    ylist = {0,0,1,1,2,2};
}
```

List意思決定変数セットを使ったTIMETSP_new.lspの例

```
/* This LocalSolver model is based on the assignment of a position to each city.
The length of the tour is computed as the sum of the distance between consecutive cities, using the at operator.*/
function model() {
```

削除部分

```
// x[i][j] equal to 1 if city j is ith visited city in the tour
 x[0..nbCities-1][0..nbCities-1] <- bool();
// one city per position i
for [i in 0..nbCities-1] constraint sum[j in 0..nbCities-1](x[i][j]) == 1;
// one position per city j
for [j in 0..nbCities-1] constraint sum[i in 0..nbCities-1](x[i][j]) == 1;
//city[i] is the city at position i in the tour
city[i in 0..nbCities-1] <- min(sum[j in 0..nbCities-1](j*x[i][j]),nbCities-1);
```

変更追加部分

```
x <- ulist(nbCities);
city[i in 0..nbCities-1] <- x[i];
```

```
local maxCode = nbCities*nbCities-1;
distanceCode[i in 1..nbCities-1] <- min(maxCode,city[i-1] + nbCities * city[i]);
distanceCode[0] <- min(maxCode,city[nbCities-1] + nbCities * city[0]);

// the distance of the arc reaching the ith city
distance[i in 0..nbCities-1] <- distanceWeight[distanceCode[i]];
```

以下同じ：TIMETSP_new.lspの例（その3）

```
    // the work to be performed at the ith city
    work[i in 0..nbCities-1] <- citywork[city[i]];
    // the due month of the ith city
    duedate[i in 0..nbCities-1] <- cityduedate[city[i]];
    // the cumulated work for the ith city + previous cities
    cumulatedWork[0] <- work[0];
    // the completion month for the ith city (stepwise function)

    month[i in 0..nbCities-1] <- piecewise(xlist,ylist,cumulatedWork[i]);

    // detect if the ith city is visited too late

    isLate[i in 0..nbCities-1] <- month[i] > duedate[i];
    // the total traveled distance
    totalDistance <-  sum[i in 0..nbCities-1](distance[i]);
    // the number of cities visited too late
    nbLate <-  sum[i in 0..nbCities-1](isLate[i]);
    // a mix of the two criteria

    minimize totalDistance + 3 * nbLate;
}
```

以下同じ：TIMETSP_new.lspの例（その4）

```
function param() {
    if (lsTimeLimit == nil) lsTimeLimit = 10;
}

function output() {
                println("Distance = "+getValue(totalDistance)+", nbLate =
"+getValue(nbLate));
    for[i in 0..nbCities-1] print(getValue(city[i])+ " ");
                println();
                for[i in 0..nbCities-1] print(getValue(cumulatedWork[i])+ ",  ");

    println();
}
```

TIMETSP.atspデータの例(同じデータ)

```
NAME:  TIMETSP↓
TYPE: ATSP↓
COMMENT: 17 city problem (Repetto)↓
DIMENSION:  17↓
EDGE_WEIGHT_TYPE: EXPLICIT↓
EDGE_WEIGHT_FORMAT: FULL_MATRIX ↓
EDGE_WEIGHT_SECTION↓
 9999    3    5   48   48    8    8    5    5    3    3    0    3    5    8    8    5↓
    3 9999    3   48   48    8    8    5    5    0    0    3    0    3    8    8    5↓
    5    3 9999   72   72   48   48   24   24    3    3    5    3    0   48   48   24↓
   48   48   74 9999    0    6    6   12   12   48   48   48   48   74    6    6   12↓
   48   48   74    0 9999    6    6   12   12   48   48   48   48   74    6    6   12↓
    8    8   50    6    6 9999    0    8    8    8    8    8    8   50    0    0    8↓
    8    8   50    6    6    0 9999    8    8    8    8    8    8   50    0    0    8↓
    5    5   26   12   12    8    8 9999    0    5    5    5    5   26    8    8    0↓
    5    5   26   12   12    8    8    0 9999    5    5    5    5   26    8    8    0↓
    3    0    3   48   48    8    8    5    5 9999    0    3    0    3    8    8    5↓
    3    0    3   48   48    8    8    5    5    0 9999    3    0    3    8    8    5↓
    0    3    5   48   48    8    8    5    5    3    3 9999    3    5    8    8    5↓
    3    0    3   48   48    8    8    5    5    0    0    3 9999    3    8    8    5↓
    5    3    0   72   72   48   48   24   24    3    3    5    3 9999   48   48   24↓
    8    8   50    6    6    0    0    8    8    8    8    8    8   50 9999    0    8↓
    8    8   50    6    6    0    0    8    8    8    8    8    8   50    0 9999    8↓
    5    5   26   12   12    8    8    0    0    5    5    5    5   26    8    8 9999↓
EOF↓
↓
```

TIMETSP_new.lsp　実行結果（その1）

```
Microsoft Windows [Version 6.1.7601] Copyright (c)
2009 Microsoft Corporation.  All rights reserved.
C:¥Users¥Miyazaki-MSI>cd C:¥localsolver_5_1¥bin

C:¥localsolver_5_1¥bin>localsolver TIMETSP_new.lsp inFileName=TIMETSP.atsp
lsTimeLimit=1

LocalSolver 5.0 (Win64, build 20150213)
Copyright (C) 2015 Innovation 24, Aix-Marseille University, CNRS.
All rights reserved. See LocalSolver Terms and Conditions for details.

Load TIMETSP_new.lsp...
Run input...
Number of cities = 17
Run model...
Preprocess model 100% ...
Close model 100% ...
Run param...
Run solver...
Initialize threads 100% ...
Push initial solutions 100% ...
```

TIMETSP_new.lsp 実行結果（その2）

Model:
expressions = 213, operands = 729
decisions = 1 (bool = 0, int = 0, float = 0),
constraints = 0, objectives = 1, constants = 34
Preprocessing transformed 51 expressions
Param:
time limit = 1 sec, no iteration limit
seed = 0, nb threads = 2, annealing level = 1
Objectives:
Obj 0: minimize, bound = 0
Phases:
Phase 0: time limit = 1 sec, no iteration limit, optimized objective = 0
Phase 0:
[0 sec, 0 itr] : obj = 186
[1 sec, 45711 itr] : obj = 42
[1 sec, 45711 itr] : obj = 42
45711 iterations, 92909 moves performed in 1 seconds
Feasible solution: obj = 42
Run output...
Distance = 39, nbLate = 1
11 0 13 2 12 10 1 9 7 8 16 4 3 14 15 5 6
13, 23, 42, 47.5, 57.5, 63, 86, 109, 125, 135, 144, 154, 164, 182, 198, 219, 236,
C:¥localsolver_5_1¥bin>

５．LSP言語の概要

5-1. LSP言語(特徴)

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。従来比、1/10から1/3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(マルチスレッド)

- 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。
- 目的計画法のように目的を順番に複数設定することができる
  (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)

- モデル作成(修正)←→実行が同時にできる
  (エディタ、DOSコマンドプロンプトの二つのウインドウを交互に利用)

モデル定義（LSP言語）

予約語	機能	摘要
bool()	意思決定変数（0-1整数変数）	
float（下限、上限）	意思決定変数（実数）	
constraint	制約条件（実行可能性の判定に使用）	
Minimize	目的関数（最小）	定義順に最適化が適用
maximize	目的関数（最大）	定義順に最適化が適用
<-	内部変数の明示的宣言	モデルの見易さ
//	注釈	

```
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;
}
```

バイナリ意志決定変数

内部変数（整数）

ユーザーはLSP言語でモデルを記述するだけ！！

出力制御

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

LSP言語(概要)

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。

従来型の言語では実現できないコンパイラによる様々な言語プログラムによる自動化とチェックが可能になる点にある。

生産性の高いモデルの記述（LSP言語）

インタプリタ型言語が搭載されており、トライ＆エラー方式で高い生産性でモデルの記述と検証が可能である。モデルは、テキストエディタを使用してLSPファイル（xxx.lsp)として作成する。
モデルは、以下の五つの関数セクションからなる。

セクション	記述	機能
入力制御	function input() { }	外部ファイルからデータを取り込み。それをもとにモデルに投入する変数の値を制御
モデル定義	function model() { }	最適化モデルを定義
出力制御	function output() { }	最適化の結果を業務システムなどとの連携を目的として外部ファイルに出力
表示制御	function display() { }	最適化走行中に標準出力に変数の値などを表示
パラメタ制御	function param() { }	local-search の実行前にモデルをパラメタ化

```
FUNCTION MODEL(){
// O-1 decisions
   X_1 <-BOOL(); X_2 <-BOOL(); X_3<-BOOL(); X_4<-
BOOL(); X_5 <-BOOL(); X_6 <-BOOL(); X_7<-BOOL(); X_8
<-BOOL();
//weight constraint
  KnapsackValues <-10*x_1+ 60*x_2+ 30*x_3+ 40*x_4+
30*x_5+ 20*x_6+ 20*x_7+ 2*x_8;
  Constraint kNAPSACKwEIGHT<=102;
//maximize  Value
  KnapsackValues <-1*x_1+ 10*x_2+ 15*x_3+ 40*x_4+
60*x_5+ 90*x_6+ 100*x_7+ 15*x_8;
  Maximize KaapsackValue;
}
```

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

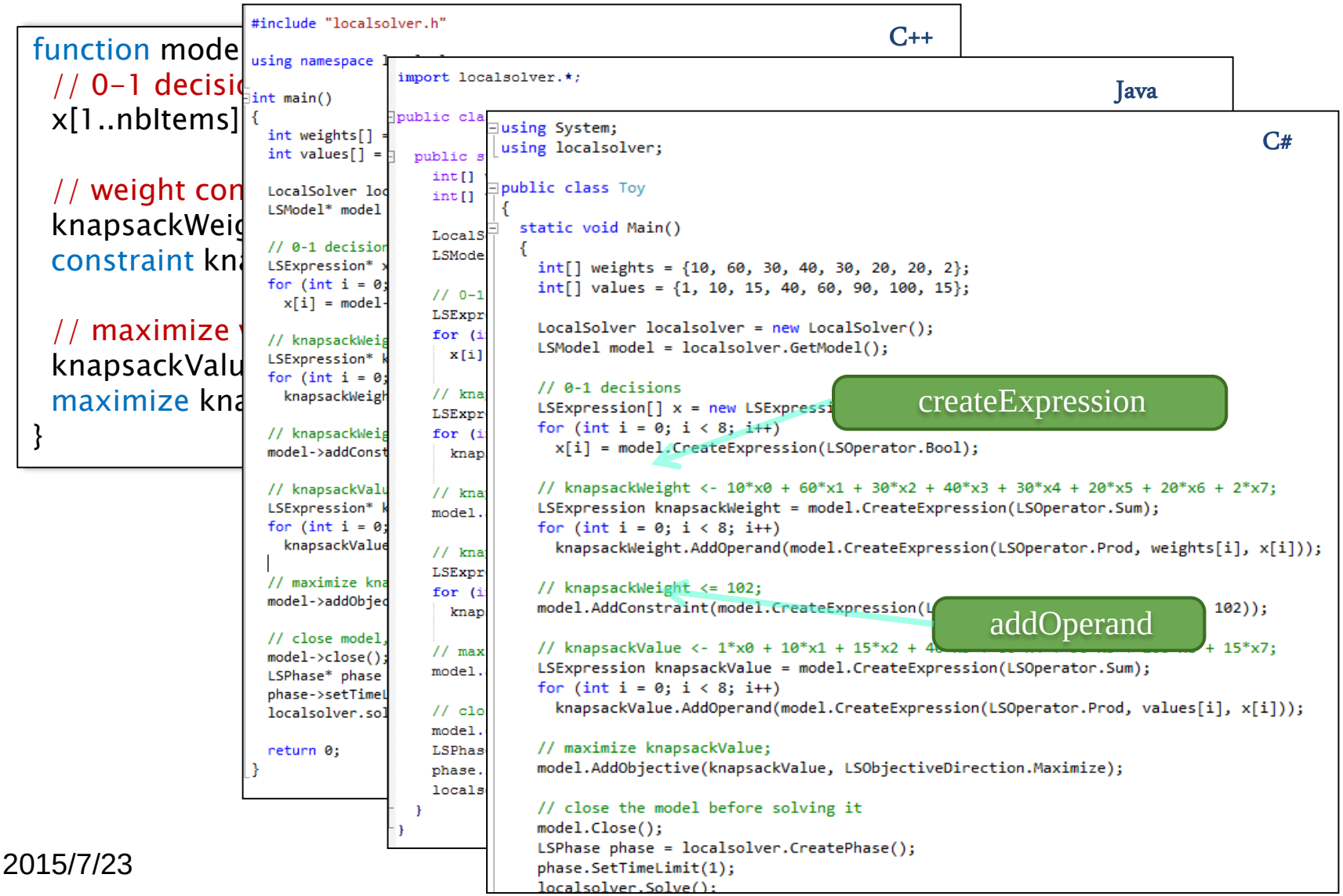
LSP言語の演算子

モデリングで利用可能なおもな演算子

（目的関数、制約式を演算子で表現可能）

算術演算子		論理演算子	関係演算子	複合演算子
sum	prod	not	==	if
min	max	and	!=	array + at
div	mod	or	<=	
abs	sqrt	xor	>=	
cos	log		<	
ceil	round		>	

LSP言語のModeling API（C#、Java、C++）



開発言語比較

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
knasack	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
比較(平均)	**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

5-2　LSPによるモデリング

- 意志決定変数の定義
  - ・**０－１整数変数の型を持つ意志決定変数**を定義
  - ・この意志決定変数の組合せが一つの解となる
  - ・探索は意志決定変数の組合せで行われる

- 制約式、目的関数の定義
  - ・**意志決定変数を使って制約式、目的関数を表現**する。
  - ・式の表現に、算術演算子、論理演算子等が使用でき、**非線形表現**で記述可能である。

- 外部入出力及び、表示
  - ・数値データの外部入力、結果の外部出力機能により、**データと問題定義を完全に分離**できる。このため、どんなに大きなモデルでも、モデル本体を簡潔に表現可能である。

モデルの記述例（ナップザック問題）

積める重量が102Kgの袋に、8個のアイテムを積める場合にアイテムの価値の合計価値を最大化させる組み合わせを求める。

番号	アイテム	価値	重量
1	X_0	1	10
2	X_1	10	60
3	X_2	15	30
4	X_3	40	40
5	X_4	60	30
6	X_5	90	20
7	X_6	100	20
8	X_7	15	2

LSP言語モデル事例（Knapsack問題）

有名なナップザック問題です。
　ある袋に8つのアイテムを積める：アイテムの合計価値を最大化させる。
　但し、合計重量は102Kg以下とする。

```
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;
}
```

バイナリ意志決定変数

作業変数（整数変数）

ユーザーはLSP言語でストレートにモデルを記述するだけ！！

LSP言語モデル事例(Knapsack問題実行結果)

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:¥Documents and Settings¥miyazaki>cd C:¥localsolver_3_0¥examples¥toy
C:¥localsolver_3_0¥examples¥toy>localsolver toy.lsp lsTimeLimit=1
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 = 1 sec, no iteration limit
  seed = 0, nb threads = 2, annealing level = 1
Objectives:
  Obj 0: maximize, no bound
Phases:
  Phase 0: time limit = 1 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, 386111 itr]: obj = (280), mov = 820062, inf = 39.5%, acc = 43.8%, imp =14
386111 iterations, 820062 moves performed in 1 seconds
Feasible solution: obj = (280)
C:¥localsolver_3_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。

モデルの実行

モデルは、DOSコマンドプロンプトを立ち上げ、LSPファイルの格納されているフォルダにカレントディレクトリを設定して実行する。

Cd C:¥localsolver_3_0¥example¥model¥
Localsolver model.lsp lsTimeLimit=1

model.lsp　　：　実行するLSPファイル
lsTimeLimit=1　：　実行時間を1秒に設定

モデルの実行結果（1）

モデルが実行され結果がログとして表示される。

```
    :
[1sec,38611 itr386111 ] : obj(280),mov=820062,inf=39.55,acc=43.5%,imp=14
386111 iterations,820062 moves performed in 1 seconds
Feasible solution: obj(280)
```

Obj(280)　　：目的関数の値（解）
mov=820062 ：探索回数
inf=39.55　　：探索回数のうち、実行不可能解の占める割合
acc=43.5%　：探索回数のうち、実行可能解の占める割合
imp=14　　：探索回数のうち、解が完全された回数

時間指定（この例では1秒を設定）などの設定により探索が打ち切られた場合、最適解の検出、実行不可能解の検出の状態を表示します。
Infeasible：実行不可能解
feasible　：実行可能解
Optimal　：最適解

価格の合計は、実行可能解として280円が検出される。

モデルの実行結果（2）

出力制御部の記述により、実行可能解などの状態や結果の内訳などを表示することができる。

```
Run output...
Selected Products:
#2 (15)
#4 (60)
#5 (90)
#6 (100)
#7 (15)
```

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

価格の合計の実行可能解として検出した280円の内訳は、
X_2(30,15), X_4(30,60), X_5(20,90), X_7(2,15)の五個で
その重さは102Kgであることが分かりる。

出力制御部の記述により、状態や結果の内訳などをファイルに出力（ファイル連携）したり、アプリケーションプログラムに値を通知することもできる。

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:¥Documents and Settings¥miyazaki>cd C:¥localsolver_3_0¥examples¥toy
C:¥localsolver_3_0¥examples¥toy>localsolver toy.lsp lsTimeLimit=1
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 = 1 sec, no iteration limit
  seed = 0, nb threads = 2, annealing level = 1
Objectives:
  Obj 0: maximize, no bound
Phases:
  Phase 0: time limit = 1 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, 386111 itr]: obj = (280), mov = 820062, inf = 39.5%, acc = 43.8%, imp =14
386111 iterations, 820062 moves performed in 1 seconds
Feasible solution: obj = (280)
C:¥localsolver_3_0¥examples¥toy>
```

LSP言語事例（MultiObjective Knapsack問題）

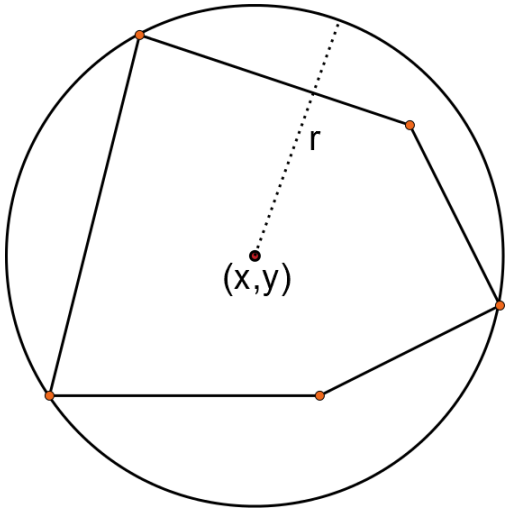
```
function model() {
  // 0-1 decisions
  x[0..7] <- bool();

 // 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];
  maximize knapsackValue;

  // secondary objective: minimize product of minimum and maximum values
  knapsackMinValue <- min[i in 0..7](x[i] ? values[i]  : 1000);
  knapsackMaxValue <- max[i in 0..7](x[i] ? values[i]  : 0);
  knapsackProduct <- knapsackMinValue * knapsackMaxValue;
  minimize knapsackProduct;
}
```

算術演算子：min、max、and、or、sin、cos、（非線形演算子）　if-then-else…

予約語

実数問題の例(最小半径決定問題)

(x、y)を中心とし、n個の点をカバーする最小半径を求める問題（Smallest circles)



実数型意思決定変数(float)

二次計算式

```
x <- float(minX, maxX);
y <- float(minY, maxY);

r2 <- max[i in 1..n](pow(x-coordX[i],2) + pow(y-coordY[i],2));

minimize sqrt(r2);
```

６．日本の事例

6-1　SCMモデル:超大規模モデル

１．問題概要

　顧客要求に合わせて、予め決められた工場ー顧客間の配送便に間に合うように、いつどこ向けになにを生産するかを決める問題（複数工場の生産スケジュール）

２．モデル概要

　ー種類の0-1整数の意思決定変数（Bool変数）で、すべての制約を表現した。また、複数の目的関数を設定し、現実的な解法を実現した。

SCMモデルの実行結果比較

実行結果（1） 既存システム

全国規模を一つのモデルにすると、実用時間内で解くことができなかった（実行CPU時間を5分程度にするのに、全国規模のモデルを **20分割**にする必要があった)。

実行結果（2） LocalSolver

LocalSolverでは、**5～6分で全国規模**を一つのモデルで解くことができた。

- Bool変数: **800万以上**
- 複数の目的関数を重要な順番で設定し、順番に最適化計算を実行。

SCMモデルの実行結果

Phase 0:
[0 sec, 0 itr]: obj = (75593, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), mov = 0, inf < 0.1%, acc < 0.1%, imp = 0

[1 sec, 27226 itr]: obj = (64967, 0, 6534, 0, 0, 0, 29280, 2838.45, 0, 519, 1059, 28.5621, 10132.2),
mov = 48265, inf = 55%, acc = 45%, imp = 17073

[2 sec, 65843 itr]: obj = (46593, 0, 23231, 0, 0, 0, 39820, 3763.47, 0, 1287, 1145, 30.6949, 27794.2),
mov = 124782, inf = 50.3%, acc = 49.6%, imp = 51706

[7 sec, 237135 itr]: obj = (11, 0, 51402, 0, 22, 0, 46130, 4122.05, 0, 2702, 1161, 31.1616, 73535.1),
mov = 477135, inf = 60.2%, acc = 38.9%, imp = 146374

[8 sec, 280000 itr]: obj = (11, 0, 51402, 0, 22, 0, 46130, 4122.05, 0, 2702, 1161, 31.1616, 73535.1),
mov = 560000, inf = 64.6%, acc = 34.4%, imp = 146374

[9 sec, 320000 itr]: obj = (10, 0, 52006, 0, 25, 0, 46150, 4125.65, 0, 2713, 1160, 31.1471, 73522.5),
mov = 640000, inf = 67.9%, acc = 31.1%, imp = 146375

[10 sec, 360000 itr]: obj = (10, 0, 52006, 0, 25, 0, 46150, 4125.65, 0, 2713, 1160, 31.1471, 73522.5),
mov = 720000, inf = 70.4%, acc = 28.5%, imp = 146376
360000 iterations, 720000 moves performed in 10 seconds
Feasible solution: obj = (10, 0, 52006, 0, 25, 0, 46150, 4125.65, 0, 2713, 1160, 31.1471, 73522.5)

6-2 人員配置（アルバイト配置計画）

1. 意志決定変数の定義（Bool変数）

X(人員、日付、時刻、業務) ＝ {0, 1}

ここまでは装置のスケジューリングとあまり変わらないが・・・

2. 人のスケジューリング特有の制約

・最大勤務時間

・最小勤務時間←**あまり短いとアルバイトは出てこない**

・１日１シフト←**勤務の途中に何もしない時間を入れない**

を導入。

人員配置計画の実行結果比較

実行結果（1） 小モデル

人員：20名、1週間、1時間毎の16タイムスロット、3業務

(意志決定変数： 20 x 7 x 16 x 3 = 9520)

LocalSolver 3.1 ：最適解まで約20秒

既存MIP ：最適解まで約30秒

実行結果（2） 業務モデル

人員：83名、1週間、15分毎の54タイムスロット、5業務

(意志決定変数： 83 x 7 x 54 x 5 = 156870)

LocalSolver3.1：最適解まで約210秒

既存MIP ：600秒以上

（制約内容等は異なっているが・・・）

6-3 裁断計画

5000mmの幅をもつフィルム等のロールから、色々な幅を持つ複数製品のロールを要求本数にできるだけ近くなるよう、裁断パターンとパターンの使用回数を求める問題である。

目的関数（複合目的関数）

1. パターン数最小（＊1000）
2. 要求数量とのギャップ最小（＊100）
3. ロス部分(4600mmからの差)。

Valid Roll width		3600 ～		4600
Products	width	Request	minimum	maximum
1	600	10	8	12
2	740	15	12	18
3	920	20	16	24
4	1060	5	4	6
5	1200	10	8	12

Pattern No	1	2	3	4	5	6	16	17	18	19	26	27	28	51	52	
Product	volume count in pattern(i)							9本				2本				
600	0	0	1	0	0	0	1	0	1	0	1	4	0	6	7	8本
740	0	1	0	0	0	0	1	2	1	3	1	0	1	1	0	18本
920	1	0	0	0	1	2	1	2	2	1	1	0	3	0	0	18本
1060	0	0	0	2	1	0	1	0	0	0	2	2	1	0	0	8本
1200	3	3	3	2	2	2	1	1	1	1	0	0	0	0	0	9本
Total	4520	4340	4200	4520	4380	4240	4520	4520	4380	4340	4380	4520	4560	4340	4200	
rest	80	260	400	80	220	360	80	80	220	260	220	80	40	260	400	

裁断計画実行結果比較

実行結果（1） 通常モデル

製品種類数：10、総製品数量：153、切断パターン候補数：12898

(意志決定変数： 75302)

LocalSolver 3.1 ：11秒で4パターン、要求差＝3

既存MIP ：400秒すぎても9パターン、要求差＝10

実行結果（2） 大規模モデル

製品種類数：18、総製品数量：504、切断パターン候補数：70921

(意志決定変数： 399372)

LocalSolver 3.1 ：31秒で8パターン、要求差＝17

既存MIP ：4060秒で10パターン、要求差＝22

フランスの事例（Car sequencing in enault's plants）

車両投入計画（塗装及びアセンブル）

- 従来はラインに沿って部品を配置する計画
- 同一色塗装の連続制約
- 二次的目的として色の変更を最小限にしたい

大規模な最適化問題

- **1300台の投入順**を決める → **400 000** バイナリ意志決定変数が必要
- MIP or CP ソルバーでは数時間の実行でも実行可能解さえ見つからない
- LocalSolver は**数秒**で実用的な精度を持つ解を Renault に提供

Car Sequencing at Renault（比較結果）

色に関する自動車配列問題の結果を示す。LP緩和解（上限は3001（万））。

LocalSolver V3.0（式数：80850, オペランド数：295212、意志決定変数：44184, 制約条件：636, 目的関数：3）

- 1秒で目的関数：16122の解を見つける
- 6秒で3478に達する
- 1分後に、3125に達する（この例で最も有名なものは3109である）

Guroubi　V5.1（式数: 29845, 列数：57331, 非零要素数：442093、連続変数：15777, バイナリ変数：41554）

- LP緩和解を求めるのに、87561イタレーション, 312 秒かかる
- B&Bで最初の実行可能解をみつけるのに647秒かかる
- 1766秒後に解27720を見つける
- 60分後でもコストはまだ25000を越えている

大規模組合せ最適化問題に関しては、Branch&Boundに基づく探索では、探索が本質的に制限されるため、解くのが難しいのが実体である。
たとえヒューリスティクティクサーチ等の発見的探索法を使用するとしても限界がある。
※かなり良い線形緩和の問題（従来のMIP用ベンチマークデータ）では、従来型でもある程度実用的な時間で解を見つけることは可能である。

7．おわりに

　30年前には殆ど実現出来なかった大規模組合せ最適化問題に対して、実践的なアプローチが実現できる時代になったと考える。
　また、大規模最適化問題の定式化技術が進歩し、簡単に大規模問題の作成が可能となっている。

「実学に役立つOR」として、人間と機械の調和を実現して日本の産業界の再生の一助となれば幸いである。

参照文献

(1)T. Benoist, B. Estellon, F. Gardi, R. Megel, K. Nouioua (2011).
・「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. Springer.

(2)MSI株式会社
・「http://msi-jp.com/localsolver/」ホームページ

(3) Frédéric Gardi(2013)
・「Habilitation Thesis in Computer Science:Toward a mathematical programming solver based on local search」、Université Pierre et Marie Curie (Paris 6) - Faculté d'Ingénierie (UFR 919) École Doctorale Informatique, Télécommunications et Électronique de Paris (EDITE).

(4) Frédéric Gardi(2013)
・「Toward a mathematical programming solver based on local search」、13th December 2013 ORO Workshop, Nantes.

有難う御座いました。

LocalSolverについてのお問い合わせ

MSI株式会社（日本配給元）
〒261-7102千葉市美浜区中瀬2-6 WBGマリブウエスト2階
Tel:043-297-8841 Fax:043-297-8836
Eメール：localsolver@msi-jp.com