LocalSolver

mathematical programming by local search

Quick Start Guide
クイックスタートガイド

＜日本語版＞

MSI 株式会社

更新：2013 年 10 月

「誤訳、不適切な訳があるかもしれませんので、

英文と併読なさるようにしてください。」

Quick Start Guide（クイック　スタートガイド）

LocalSolverは大規模組合せ問題を解く、世界初、実用的にカプセル化したローカルサーチ・ソルバーです。高性能ローカル・サーチを用いて、ビジネス問題をモデリングすることができます。LocalSolverは複雑なパラメーター・チューニングを行うことなく問題規模の拡張ができます。既存の数理計画ソルバーと同様、求まった解に保証はありませんが、LocalSolverは現実世界の大規模アプリケーションに対し、短時間で高品質な解を求めることができます。すでに多くの大手企業で導入され、LocalSolverは企業の信頼性とロバスト性を改善すると同時に、最適化ソリューションの開発及び維持費の大幅な削減に成功しています。

2通りの使用法でLocalSolverをご使用頂けます。モデラーとして使用する場合、短時間で複数の最適化アプリケーションのプロトタイプが可能です。このように、LocalSolverのモデラーは特にオペレーション・リサーチや経営のコンサルト、分析者のニーズに適しています。localSolverの呼出し可能なライブラリ（C++、Java、.NET）を使用することで、企業内の情報システムと最適化アプリケーションを自由に統合すること可能です。

このクイックスタートガイドを使用し、数分間でファーストモデルのモデリングをが可能です。このマニュアルで提供するコーディング・例題は、LocalSolver directoryの　フォルダexsampleで入手頂けます。その他、LocalSolverを使用した例題は、LocalSolver例題集を参照ください（日本語サイト：http://www.msi-jp.com/localsolver/）

目次

Getting Started （初めに）

LocalSolver は C++言語で実装されています。Standalne executable また callable library（C++、Java、.NET)として使用することが可能です。
バイナリファイルは 3 つのプラットホーム：Windows、Mac OS、Linux および 2 つのアーキテクチャ：×86、×64 で利用可能です。 また Java および.NET のラッパーが提供されています。

LocalSolver バイナリファイルが使用されています：C および C++ライブラリを除く第三者ライブラリは使用されていません。
使用するプラットホームに LocalSolver をインストールするステップを、各該当ページでご確認ください。

Starting on Windows
Starting on Linux
Starting on Mac Os
目次に戻る

Starting on Windows （Windows で起動させる）

ここでは、Localsolverのインストール、及びライセンスの取得手順を解説します。
LocalSolverはC++言語で実装されています。当ソフトウエアはスタンドアロンの実行形式（実行プログラム）、または呼び出し可能なライブラリ（C++、Java、.NET）として利用頂けます。
ダウンロードページで解説されている通りに、ソフトウエアを正常に実行するために最小限必要なシステム構成が満たされていることをご確認ください。

必要システム構成

- アーキテクチャ：x86、x64.　注：x64バージョンは完全な64bit環境を提供。
- オペレーティング・システム：WindowsXP(またはこれより上位、Vista、windows7、8、Server2003、Server2008)
- ネイティブC++ライブラリへの接続：Visual Studio 2010(またはこれより上位)
- Javaライブラリへの接続：J2SEランタイム環境5.0のJava開発キット5.0
- NETライブラリへの接続：.NETフレームワーク2.0のVisual Studio2015

インストール

LocalSolverをインストールするにはコンピュータ上で使用するライセンスを取得する必要があります。その後、LocalSolver_XXX.exeを実行し、表示された手順に従って操作を続けてください。

ライセンス

LocalSolverディレクトリ　内の、license.datファイルはLocalSolverの使用用途により容量が制限されています。（最大で100の意思決定変数及び1000の表現式）
ビジネス・ライセンスまたはアカデミック・ライセンスを取得されている場合、license.datをアカウント内の利用可能なライセンスファイルで上書きします

デスクトップ：アカデミック・デスクトップライセンス、サーバーライセンス、またはトライアルライセンス

1 アカウントに入り、ライセンスファイルをダウンロードします。
2 アカウントからダウンロードしたファイルでC:¥localsolver_XXX¥にある「license.dat」ファイルを上書きします。

フローティング・ライセンス、またはアカデミック・ネットワーク・ライセンス

トークン・サーバーをセットアップします。その後、下記の手順で各クライアント・マシンにライセンス与えます。

トークン・サーバーの手順

1 アカウントへ入り、トークンサーバーのライセンスファイルをダウンロードします。
2 アカウントからダウンロードしたトークン・サーバーライセンスでC:¥localsolver_XXX¥にある「license.dat」ファイルを上書きします。

3 Services panelを開きます（Start > Control Panel > Administrative tools)。"LocalSolver token server" serviceをクリックし、起動させます。トークン・サーバーを自動的に起動させるには、serviceを右クリックし、starting typeをautomaticに変更します。

各クライアントマシンの手順

新しくフォルダを作成し、トークン・サーバーIPを持つ、下記の行を記述します。

TOKEN_SERVER = <HOST NAME OR IP ADDRESS OF THE TOKEN SERVER>

C:¥localsolver_XXX¥license.dat.に上記の行をセーブします。

テスト

folder "examples¥toy"でターミナルを開き、下記の行を入力します。

localsolver.exe toy.lsp lsTimeLimit=1

エラー表示がない場合、LocalSolverを使用し最初のモデルを解く準備が完了しました。

LocalSolverをアンインストールにする

LocalSolverをアンインストールにするには、コンピュータのadministrator rightが必要です。

LocalSolverフォルダで、Uninstall.exeを起動します。

特例

クライアント・マシンとして、トークンサーバーを使用する場合、単にトークン・サーバーファイルをこのマシンのトークン・クライアントファイルと結合させてください。

例:

```
SERVER_KEY = <LICENSE KEY OF MACHINE>
```

TOKEN_SERVER = <HOST NAME OR IP ADDRESS OF THE TOKEN SERVER>

目次に戻る

Starting on Linux (Linux で起動させる)

ここでは、Localsolverのインストール及びライセンス取得に関する主要な手順を解説しています。
LocalSolverはC++言語で実装されています。スタンドアロンの実行形式(実行プログラム)、または呼び出し可能なライブラリ(C++、Java)としてご使用頂けます。
ダウンロードページに解説されている通りに、ソフトウエアを正常に実行するための最小限必要なシステム構成が満たされていることをご確認ください。

必要システム構成

- アーキテクチャ:x86、x64. 注:x64バージョンは完全な64bit環境を提供
- オペレーティング・システム:libc2.3.2(またはこれより上位)、libstdc++3.4.9(またはこれより上位)の全Linuxの標準システム
- ネイティブC++ライブラリへの接続:GCC4.2(またはこれより上位)
- Javaライブラリへの接続:J2SEランタイム環境5.0のJava開発キット5.0

インストール

root権限(管理者モード)としてコンピュータにLocalSolverをセットアップする必要があります。
LocalSolverのインストールは下記の2つのステップで行います。

1. アーカイブLocalSolver_XXX.runで、rootとしてターミナルを開きます。
2. 下記のコマンド行を実行します。

   bash LocalSolver_XXX.run

   LocalSolverは/opt directoryにインストールされます。

ライセンス取得

LocalSolverディレクトリ 内、license.datファイルはLocalSolverの使用用途で容量制限されています。(最大で100の意思決定変数及び1000の表現式)
ビジネス・ライセンスまたはアカデミック・ライセンスを取得されている場合、license.datをアカウント内の利用可能なライセンスファイルで上書きします。

デスクトップ、アカデミック・デスクトップまたはトライアル・ライセンス

1. アカウントへ入り、ライセンス・ファイルをダウンロードします。
2. アカウントでダウンロードしたファイルで/opt/localsolver_XXX/にある"license.dat" fileを上書きします。

フローティング・ライセンス、サーバー・ライセンスまたはアカデミック・ネットワーク・ライセンス

トークンサーバーをセットアップし、各クライアントマシンにライセンスを下記の手順に従って与えます。

トークンサーバー

1. アカウントへ入り、トークン・サーバーのライセンスファイルをダウンロードします。
2. アカウントでダウンロードしたサーバー・ライセンスファイルで/opt/localsolver_XXX/にある"license.dat" fileを上書きします。
3. ターミナルを開き、実行ファイル"lstokenserver"を実行します。注:このプログラムは、バックグラウンドで実行します。マシンがシャットダウンするまで、実行状態です。安全にターミナルを終了することができます。

マシンを再起動する時、自動的にトークン・サーバーを実行させるには、init scriptを作成します。または、etc/rc.localに下記の行を追加します。

/opt/localsolver_XXX/bin/lstokenserver

各クライアントマシンの設定

新しいファイルを作成し、トークン・サーバーIPを持つ下記の行を記述します。

TOKEN_SERVER = <HOST NAME OR IP ADDRESS OF THE TOKEN SERVER>

/opt/localsolver_XXX/license.datに上記の行をセーブします。

テスト

フォルダ/opt/localsolver_XXX/examples/toyでターミナルを開き、下記の行を入力します。

$ localsolver toy.lsp lsTimeLimit=1

エラー表示がない場合、localSolverを使用し、最初のモデルを解く準備が完了しました。

LocalSolverをアンインストールにする

LocalSolverをアンインストールにするには、コンピュータのroot権限が必要です。

2ステップでLocalSolverをアンインストールにします。

1. rootとしてターミナルを開き、/opt/localsolver_XXX/ に入る
2. 下記のコマンド行を実行します。

"bash uninstall.sh"

特例

クライアントマシンとして、トークンサーバーを使用する場合、単にトークンサーバーファイルをこのマシンのトークンクライアントファイルと結合させてください。

(例) SERVER_KEY = <LICENSE KEY OF MACHINE> TOKEN_SERVER = <HOST NAME OR IP ADDRESS OF THE TOKEN SERVER>

上記は、推奨通りに、LocalSolver フォルダが、/opt/localsolver_XXX にあることを前提としています。推奨されていないフォルダに LocalSolver がセットアップされている場合、LocalSolver に、ライセンスファイルがどこに格納されているかを示す必要があります。LocalSolver license.dat の格納場所に LS_LICENSE_PATH 環境変数を作成し、設定してください。

目次に戻る

Starting on Mac OS(Mac Os で起動させる)

ここではLocalSolverのインストールおよび、ライセンス取得に関する主要な手順について解説しています。
LocalSolverはC++言語で、実装されています。当ソフトウエアは、スタンドアロン実行形式または呼び出し可能ライブラリの（C++、Java）としてご使用頂けます。

ダウンロードページに記載されているように、ソフトウエアを正常に実行するため、最小限必要なシステム構成が満たされていることをご確認ください。

必要システム構成

Mac Os上でLocalSolverを使用するための要件は、可能な限り簡潔に構築しています。
システムとライブラリには、LocalSolverが必要とする最小限のバージョンのみ与えられます。
例えば、「Mac OS X 10.6」とは、Mac OS X 10.6またはこれより上位バージョンが必要なことを意味します。

- アーキテクチャ：x86, x64　注：x64バージョンは、完全な64-bit環境を提供
- オペレーティング　システム：Snow Leopard（またはこれより上位）
- ネイティブC++ライブラリへの接続：GCC 4.2（またはこれより上位）
- Javaライブラリへの接続：J2SE　ランタイム環境5.0のJava　開発キット　5.0

インストール

LocalSolverをセットアップするには、コンピュータのroot権限が必要です。LocalSolver_XXX.pkgを実行し、手順に従ってください。LocalSolverは、/opt directoryにインストールされます。

ライセンス取得

LocalSolverディレクトリにあるlicense.dat fileはLocalSolverの使用用途で容量制限されています。（最大で100の意思決定変数及び1000の表現式）
ビジネス・ライセンスまたはアカデミック・ライセンスを取得されている場合、license.datをアカウント内の利用可能なライセンスファイルで上書きする必要があります。

デスクトップ、アカデミック・デスクトップまたはトライアル・ライセンス

1. アカウントに入り、ライセンスファイルをダウンロードします。
2. アカウントでダウンロードしたファイルで/opt/localsolver_XXX/にある"license.dat" fileを上書きします。

フローティング・ライセンス、サーバー・ライセンスまたはアカデミック・ネットワークライセンス

トークンサーバーをセットアップし、各クライアントマシンにライセンスを下記の手順に従って与えます。

トークンサーバーの手順

1. アカウントへ入り、トークンサーバー・ライセンスファイルをダウンロードします。
2. アカウントでダウンロードしたサーバー・ライセンスファイルで/opt/localsolver_XXX/にある"license.dat" fileを上書きします。
3. ターミナルを開き、実行ファイル"bin/lstokenserver"を実行します。注:このプログラムは、バックグラウンドで実行します。マシンがシャットダウンするまで、実行状態です。安全にターミナルを終了することができます。
   マシンを再起動する時、自動的にトークン・サーバーを実行させるには、init script を作成します。または、etc/rc.localに下記の行を追加します。

/opt/localsolver_XXX/bin/lstokenserver

各クライアントマシンの手順

新しいファイルを作成し、トークン・サーバーIPを持つ下記の行を記述します。

TOKEN_SERVER = <HOST NAME OR IP ADDRESS OF THE TOKEN SERVER>

/opt/localsolver_XXX/license.datに上記をセーブします。

テスト

folder /opt/localsolver_XXX/examples/toyでターミナルを開き、下記の行を入力します。

$ localsolver toy.lsp lsTimeLimit=1

エラー表示がない場合、localSolverを使用し、最初のモデルを解く準備が完了しました。

LocalSolverをアンインストールにする

LocalSolverをアンインストールにするには、コンピュータのroot権限が必要です。

2ステップでLocalSolverをアンインストールにします。

1. rootとしてターミナルを開き、/opt/localsolver_XXX/ に入る
2. 下記のコマンド行を実行します。

   "bash uninstall.sh"

特例

クライアントマシンとして、トークンサーバーを使用する場合、単にトークンサーバーファイルをこのマシンのトークン・クライアントファイルと結合させてください。

(例)SERVER_KEY = <LICENSE KEY OF MACHINE> TOKEN_SERVER = <HOST NAME OR IP ADDRESS OF THE TOKEN SERVER>

上記は、推奨通りに、LocalSolver フォルダが、/opt/localsolver_XXX にあることを前提としています。推奨されていないフォルダに LocalSolver がセットアップされている場合、LocalSolver に、ライセンスファイルがどこに格納されているかを示す必要があります。LocalSolver license.dat の格納場所に LS_LICENSE_PATH 環境変数を作成し、設定してください。

目次に戻る

Quick tour of LocalSolver’s modeler
(LocalSolver モデラー・クイックガイド)

LocalSolverのインストールおよびライセンス取得終了後、次のステップとして、LocalSolverのモデラーの主要機能を確認します。 LocalSolver Programming language(LSP)はLocalSolverに実装されているローカルサーチ法に基づく革新的なモデリング言語です。LSPは各種数学演算子やゴール・プログラミングを提供します。また上記以外に、このインタプリタ言語は先進的なスクリプト言語を提供します。:ダイナミックかつ暗黙的な変数宣言、簡潔なシンタックスで記述できるループ宣言等、LocalSolverの関数はモデリングだけではなくC++、C#、JAVAを使用した開発も容易に行えるため、 LSPを使用し記述したLSPモデルは他のモデリング言語で記述したモデルよりも、理解し易く読み易い内容になります。 但し、LSP言語の関数はmain 関数を提供しません。実際には、LSP言語は5つの定義済み関数input()、model()、display()、output()、param()のモデリング・フレームワークを使用しモデリングを行います。 LocalSolver・実行ファイルを使用し上記の関数は下記のプログラムを呼び出します。

- input ： ファイルからデータを宣言、または読み込みを行う
- model ： 最適化モデルを宣言する
- param ： 実行前にローカルサーチソルバーにパラメータ値を与える
- display ： 解法中、コンソールまたは複数のファイル内に情報を表示する
- output ： 一度解き終わった問題の解をコンソール内またはファイル内に書き出す

model()を除く全てのbuilt-in変数、built-in関数はデフォルト変数を持ちます。(特にparam()関数)

LSP言語システム開発の目的はプロトタイプ作業の負担を出来る限り減少させることです。

LocalSolverモデラーのクイックガイド主要トピックは下記の通りです。(数理モデリングの基礎、プログラミング形式、全built-in関数、built-in変数のリスト)詳細情報は、リファレンスマニュアルを参照ください。LocalSolverを使用したモデリング法を解説するために、まずオペレーションズ・リサーチでは馴染み深いナップザック問題を解いていきます。

Solving your first model
Mathematical modeling features
Programming style
Built-in variables and functions
目次に戻る

Solving your first model（ファースト・モデルを解いてみよう）

このセクションでは、ファースト・モデル（基本的なナップザック問題）をどのようにモデリングし、解くのかについて解説しています。ナップザック問題を下記のように定義します。
重量と価値をそれぞれ持つアイテム集合があります。これらの全重量が定められた制限を超えずに可能な限り全体の価値が最大になるように部分集合を決定します。ナップザック問題は理論的に解き難い問題です。

ナップザック トイ モデル

下記はナップザック・トイ例題（examples/toy）のLocalSolverプログラム（LSP）です。：合計重量は102キロ以下とし、各8アイテムの重量は10、60、30、40、30、20、20、2で各8アイテムの価値は1、10、15、40、60、0、100、15
モデルの作成法は、整数計画と全く同様である点に留意ください：各アイテムにおいて、0-1意思決定変数はアイテムがナップザックに取られる場合、1となり、取られない場合は0と定義されます。

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

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;
}
```

表現式と呼ばれるモデルの全変数は左矢印<-を使って宣言されています。bool変数（0-1意思決定変数）は算術演算子boolを使用することで導入されます。内部変数は他の演算子を使用し、これらの意思決定変数を生成することが可能です。product (*), sum (+), lower than or equal to (<=)その他多くの各種数学記号が利用頂けるため、大変理解しやすい形で非線形計算が必要な制約を表現することができ、非線形組合せ最適化問題の生成及び、求解が可能です。キーワードconstraintまたはmaximizeは表現式に制約、または最大化の宣言を追加するために使用しています。
このモデルを解くために、アーギュメントとしてLSPファイルを使用しLocalSolverを呼び出します。さらに、built-in変数（プログラミング）lsTimeLimitの値はLocal-searchソルバーの実行時間を1秒に制限するために入力します。
その後、コンソールに下記のトレースが出力します。

```
localsolver_2_0/examples/toy$ localsolver toy.lsp lsTimeLimit=1
LocalSolver 2.0 (build 20120213)
Copyright (C) 2012 Bouygues SA, Aix-Marseille University, CNRS.
All rights reserved (see End-User License Agreement for details).

Run model...
Run solver...

Model:
  expressions = 50, operands = 62
  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:

[1 sec, 24115 itr]: obj = (280), mov = 49405, inf = 75.6%, acc = 2.7%, imp = 13

24115 itr, 49405 mov performed in 1 sec
Feasible solution: obj = (280)
```

Time limit を設定していない場合、Ctrl+C をクリックして、プログラムの停止を強制するまで、探索が継続します。

コンソール内のトレースはまず、モデのル主要数値を表示します。(表現式の数、引数の数、意思決定の数、制約およびオブジェクトの数)その後、他のデフォルトパラメータを表示します:デフォルト・パラメータ、(スレッド数、シュミレーティドアニーリング・レベル)マルチ・オブジェクトの最適化を実行する場合のオブジェクト及びフェーズに関するトレース方法は、当ガイドの巻末を参照ください。

LocalSolver は各二番目に、経過時間、イタレーション数、現時点での最も良いオブジェクト値を表示します。*inf*, *acc* および *imp* の演算子は実行不可能解につながるムーブのパーセンテージ、許可されたムーブのパーセンテージ、およびそれぞれの改善したムーブ総数を表示します。探索が終了すると、イタレーション数とムーブ総数が、発見された最適解のステータスと同様に表示されます。(注;複数のスレットを使用した場合、多くのムーブが実行できます。)解のステータスは、実行不可能解、実行可能解または最適解となります。

目次に戻る

基本的　ナップザックモデル

ここでは、入力データが、フォーマット済みファイルで指定された任意のナップザック例題をどのように解くかを解説します。
LSPファイルは、下記のように(examples/knapsack)簡単に修正することが可能です。

```
/********** knapsack.lsp **********/

function input() {
  usage = "\nUsage: localsolver knapsack.lsp "
    + "inFileName=inputFile [lsTimeLimit=timeLimit]\n";

  if (inFileName == nil) error(usage);

  inFile = openRead(inFileName);
  nbItems = readInt(inFile);
  weights[i in 0..nbItems-1] = readInt(inFile);
  values[i in 0..nbItems-1] = readInt(inFile);
  knapsackBound = readInt(inFile);
}

function model() {
  // 0-1 decisions
  x[i in 0..nbItems-1] <- bool();

  // weight constraint
  knapsackWeight <- sum[i in 0..nbItems-1](weights[i] * x[i]);
  constraint knapsackWeight <= knapsackBound;

  // maximize value
  knapsackValue <- sum[i in 0..nbItems-1](values[i] * x[i]);
  maximize knapsackValue;
}

function param() {
  if (lsTimeLimit == nil) lsTimeLimit = 10;
  lsNbThreads = 1;
}
```

```
function display() {
  println("knapsackWeight = " + getValue(knapsackWeight)
    + ", knapsackValue = " + getValue(knapsackValue));
}

function output() {
  println("Write solution into file 'sol.txt'");
  solFile = openWrite("sol.txt");
  for [i in 0..nbItems-1 : getValue(x[i]) == 1]
    println(solFile, i);
}
```

下記のコマンド行でプログラムを実行すると、前回のLSPファイルに類似したアウトプットを得ます。注:より柔軟性が増すように、コマンド行でいくつかの変数を定義しました。(*inFile*Name, lsTimeLimit)

localsolver_2_0/examples/knapsack$ localsolver knapsack.lsp inFileName=instances/toy.in lsTimeLimit=1

全ての算術演算子を、モデリング及びプログラミング作業の双方で、使用することができます。
例えば、ステートメントc <- a * bは表現cがモデリング表現aとbの積に一致することを宣言しています。一方、c = a * bは、

プログラミングの変数a とbの積の結果をプログラミングの変数cに割り当てることを意味します。n-配列演算子は、任意の引数で表現式を宣言するために、n配列関数が使用できます。(例えば、sum(a, b, c))
オペランドの数が固定される場合、バイナリのinfix形式(例えばa+b+c)として使用することも可能です。
例えば、

```
knapsackWeight <- sum[i in 0..nbItems-1](weights[i] * x[i]);
```

は、0 to nbItems-1.による全てのアイテムに、表現式weights[i]とx[i]の積の合計として表現式knapsackWeightを宣言したステートメントです。LSP言語の特徴は、main 関数を提供しないことです。実際、LSP言語は5つの定義済み関数input(), model(), param(), display(), output().から構成されるモデリング形式を使用します。
これらの関数はLocalSolverの実行ファイルで下記のプログラムを呼び出します。

- input: **データの宣言または、ファイルからデータを読み込む**
- model: **最適化モデルを宣言する**
- param: **実行前にローカルサーチソルバーをパラメータ化する**
- display: **求解中にコンソール内、または複数のファイル内で、様々な情報を表示する**
- output: **一度、解が得られたとき、コンソール内、またはファイル内にその結果を記述する**

注;model()を除き、全ての built-in 変数および built-in 関数はデフォルト値を持ちます。
目次に戻る

Mathematical modeling features(数理モデリング機能)

LocalSolverは非線形0-1問題を解くことが可能です。全ての意思決定変数はバイナリ変数(0または1)のみです。意思決定変数を使用し、値が許す変数:モデル内で宣言された全変数の値を計算ができ、特に制約式や、オブジェクトで生じる変数を計算します。多くの組合せ最適問題は0-1モデルとして表現することが可能です。
ナップザック問題は例題の1つです。:中間変数(ナップザックの重量、ナップザックの価値)は整数、意思決定変数はバイナリ変数です。注:正確な精度の10進法を必要とする多くの問題を扱うことができ、このような線形表現は-2^63 toから2^63-1 (> 10^18)までの整数値を取ることが可能です。
ブール意思決定変数は、演算子 boolを使用し、宣言されます。またモデリング表現と呼ばれる中間変数は下記の表で解説されている算術演算子を使用し宣言させることが可能です。
いかなる表現もキーワードconstraint, maximize, またはminimize.を使用し、制約式またはオブジェクトとしてタグ付けすることが可能です。注:複雑な非線形モデルを宣言するために、これら全ての演算子が使用できます。
従って、LocalSolverは制約式においてだけではなくオブジェクトにおいても非線形、非凸表現式もサポートします。
いかなるブール表現もキーワードconstraint.を表現式の前に付けることで、制約式として表現することができます。

```
constraint knapsackWeight <= 102;
```

全ての制約が1の値を取り、コーディングが満たされている場合、意思決定変数のインスタンス化が可能です。
LocalSolverは、(および、一般的なローカル・サーチ)厳しい制約を持つ問題の求解に適さないため、出来る限り厳しい制約を排除してください。特に、現実の現場で確実に満たせない制約は排除する必要があります。理想は、組合せの制約式のみが、設定されることです。(すなわち問題に組合せ構造を帰納するもののみ)
その他、全ての制約は、「緩く」満たす(ゴールプログラミング)ために、主要なオブジェクトを緩和します。
LocalSolverはこれらの作業を行い易くするために、予約語オブジェクトを提供しています。

少なくとも、1つのオブジェクトをキーワードminimize または maximizeを使用して定義します。
複数のオブジェクトを定義する場合、すべての表現をオブジェクトとして使用でき、それらは予約語の目的関数として、変換されます。予約語オーダーはオブジェクトが宣言されたオーダーに帰納します。
下記のような表現は、数理プログラミングで頻繁に出てきます。

```
maximize 10000 revenues - 100 resources + desiderata;
```

先ずrevenuesを最大化、その後resourceを最小化、そして最終的にdesirataの最大化を行うために厳しい制約は排除します。下記を直接記述することが可能です。

```
maximize revenues; minimize resources; maximize desiderata;
```

	Function	Description	Arguments type	Result type	Arity	Symb
Decisional	bool	Boolean decision variable with domain {0,1}.	*none*	bool	0	
Arithmetic	sum	Sum of all operands.	bool, int, double	int, double	$n > 0$	+
	prod	Product of all operands.	int, double	int, double	$n > 0$	*
	min	Minimum of all operands.	int, double	int, double	$n > 0$	
	max	Maximum of all operands.	int, double	int, double	$n > 0$	
	div	Division of the first operand by the second one.	int, double	int, double	2	/
	mod	Modulo: mod(a,b) = r such that $a = q*b + r$ with q, r integers and $\|r\| < b$.	int	int	2	%
	abs	Asolute value: abs(e) = e if e >= 0, $-e$ otherwise.	int, double	int, double	1	
	dist	Distance: dist(a,b) = abs(a-b).	int, double	int, double	2	
	sqrt	Square root.	int, double	double	1	
	cos	Cosine.	int, double	double	1	
	sin	Sine.	int, double	double	1	
	tan	Tangent.	int, double	double	1	
	log	Natural logarithm.	int, double	double	1	
	exp	Exponential function.	int, double	double	1	
	pow	Power: pow(a,b) is equal to the value of a raised to the power of b.	int, double	double	2	
	ceil	Ceil: round to the smallest following integer.	int, double	int	1	
	floor	Floor: round to the largest previous integer.	int, double	int	1	
	round	Round to the nearest integer: round(x) = floor (x+0.5).	int, double	int	1	
	exp	Exponential function.	int, double	double	1	
	scalar	Scalar product between 2 arrays.	array	int, double	2	
Logical	not	Not: not(e) = 1 - e.	bool	bool	1	!
	and	And: equal to 1 if all operands are 1, and 0 otherwise.	bool	bool	$n > 0$	&&
	or	Or: equal to 0 if all operands are 0, and 1 otherwise.	bool	bool	$n > 0$	||
	xor	Exclusive or: equal to 0 if the number of operands with value 1 is even, and 1 otherwise.	bool	bool	$n > 0$	
Relational	eq	Equal to: eq(a,b) = 1 if a = b, and 0 otherwise.	int, double	bool	2	==
	neq	Not equal to: neq(a,b) = 1 if a != b, and 0 otherwise.	int, double	bool	2	!=
	geq	Greater than or equal to: geq(a,b) = 1 if a >= b, and 0 otherwise.	int, double	bool	2	>=
	leq	Lower than or equal to: leq(a,b) = 1 if a <= b, and 0 otherwise.	int, double	bool	2	<=
	gt	Strictly greater than : gt(a,b) = 1 if a > b, and 0 otherwise.	int, double	bool	2	>
	lt	Strictly lower than: lt(a,b) = 1 if a < b, and 0 otherwise.	int, double	bool	2	<
Conditional	iif	Ternary conditional operator: iif(a,b,c) = b if a is equal to 1, and c otherwise.	bool, int, double	bool, int, double	3	? :
	at	$T[i]$ returns the ith value in array T.	array, int	bool, int, double	2	[ ]

目次に戻る

Programming style（記述形式）

算術演算子に加えて、簡単、迅速にモデルを構築するために、LocalSolverのモデラーは必須のプログラミング言語を複数提供します。この言語（LSP言語）は、スクリプト言語の全特徴を持ちますが、問題のモデリングと求解が特徴です。
注：LSP言語は大文字、小文字を区別して使用します。

基本形式と条件文

変数宣言が即座に行われます。ステートメントx=vは値vを変数xに割当てます。変数が型を持たない場合、値は厳しい型となります。：変数は単にコンテナーです。2つのタイプが明示的に定義されます： integers (x = 12)とstrings (x = "foo")。ブール型はそれぞれ偽と真としてコーディングし、整数0と1で、暗黙的に定義されます。値vがモデリング表現と一致している場合、左側の矢印は＝の代わりに<-を使用する必要があります。このようにして、下記のスタートメントが有効になります。

```
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
```

マップ

マップとはキーに関連付けて値を格納するデータ構造です。キーは文字列であり整数（必ずしも連続的ではない）です。値は任意の型で設定できます。キーに関連付けした値は指定するかまたはカッコ表記を使用し返されます。: a[9] = "abc", a["abc"] = 9. マップはその最初の要素を代入する時、暗黙的に宣言されます。または、空のマップは下記のように、map()関数または{}を使用し定義します。

```
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
a[-3] = "xyz";   // a[0] = "z", a[1] = 9, a["a"] = abc, a[-3] = "xyz"
```

map a のkey[k]に値が割当てられていない場合、a[k]はnillを返します。 既存のキーに値を設定するには、前に関連付した値に上書きします。

条件文

if (C) S_true; else S_false; は条件文です。：cが偽（すなわちCが0ならば）、ステートメントS_falseが実行され、条件Cが真（すなわちCが1）ならばステートメントS_trueが実行されます。注：else 条件分岐はオプションです。
{}を使用し、ブロック・ステートメントを宣言することで、複数のステートメントを実行できます。条件付き三項演算子？を使用すると、コンパクトに記述できます。

```
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;
}
```

ループ文

while (C) S;　条件Cが真ならば（すなわちCが1ならば）、ステートメントSを繰り返し実行します。do S; while (C);は条件cに関係なく、必ず一度ステートメントSを実行し、その後、条件を後判定し、条件が真ならば、Sを繰り返し実行します。
for [v in V] SはステートメントSはVにすべての値を取るVで反復処理を行います。そこでは、Vはレンジまたはマップとなることがあります。from.toで宣言したレンジは、最小値と最大値が含まれます。
マップを得た場合、イタレーションはキーの増加率に従いマップの値で実行されます。
マップのペア（キー、値）は、for [k,v in M].文で、繰り返します。

```
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 [v in V : C]はフィルした反復処理です。変数 v は条件 C を満たしている V の値のみ取ります。注：ループ処理を入れ子構造として構築すると下記のように簡潔に記述することができます。

```
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;
```

すべてのループ制御文に、{}を使用し、ブロック・ステートメントを宣言すると、複数のステートメントを実行できます。

```
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;
}
```

反復条件

LSP言語が提供する特性は「反復条件」です。反復条件文を使用することで簡潔で理解しやすいモデルを記述することができます。コードfor [v in V] a[v] = f(v);を使用し、簡潔に記述することが可能です。a[v in V] = f(v); forと同様、イタレーションをネスト化し、フィルタリングが可能です。

```
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; // very compact!
```

反復条件文を使用し、モデリング表現式を宣言することができます。

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

関数

関数はキーワードfunctionを使用し、宣言します。パラメータ値は括弧で囲い、そのコードは中括弧で囲いカプセル化します。

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

値はreturnステートメントを使用し、あらゆるポイントから呼び出し側の関数に値を返すことが可能です。デフォルトを使用し、すべての変数の参照有効範囲（変数のスコープ）を広範囲に設定することができ、プログラム内のどこからでも、それらのブロックにローカル関数および反復変数（forステートメントに導入された）のパラメータとして特例的に渡すことが可能です。キーワード local はそのブロック内で、変数をローカルに定義します。注：モデリング表現式はローカルに定義することができません。またローカル変数を使用することで、別の場所で宣言されている同名の変数により、副次的問題が生じないことを確認できます。

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

目次に戻る

Built-in variables and functions（組込み変数・関数）

Variables

Modeling & Solving

- 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,..., 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

- 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

- 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

- 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

- 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).

目次に戻る

Quick tour of LocalSolver's APIs
(LocalSolver APIs クイック・ツアー)

LocalSolver APIsクイック・ツアーでは、情報システムにLocalSolverを基礎とする最適化アプリケーションの自由な統合を望む実務者向けの内容を記載しています。LocalSolverはC++言語で実装されています。そのため、フルポータブルなC++ライブラリにはオブジェクト指向プログラミングインターフェースを提供します。JAVA 5.0向けラッパーおよび.NET 2.0のフレームワークも提供しています。LocalSolverAPIsのオブジェクト・モデルは処理するクラスが、軽量化されています。LocalSolver モデラーのクイック・ツアーではナップザック問題の解き方を解説し、LocalSolverが提供するプログラミング・インターフェースを使用し、C++、JAVA、C#でナップザック問題を解くことができます。

Solving your first model in C++

Solving your first model in Java

Solving your first model in C#

目次に戻る

Solving your first model in C++
（C++で最初のモデルを解いてみよう）

LocalSolverはC++言語で実装されています。移植性の高いC＋＋言語をオブジェクト指向アプリケーション・プログラミングインターフェース（ユーザーのビジネス・アプリケーション内に完全な統合が可能です。）で提供します。（APIs）
LocalSolverAPIsは処理するクラスが、軽量化されています。LocalSolverがカプセル化した数理計画ソルバーであることを前述しました。ソルバーを実行するために、モデルをインスタンス化し、追加のコードを記述する必要はありません。

下記は、quick tour of LocalSolver's modeler.で紹介したナップザック・トイ例題を解くため、C++コードで記述しています。
対応するソース・ファイルはexamples/toyで入手頂けます。

```
/********** toy.cpp **********/

#include "localsolver.h"

using namespace localsolver;

int main()
{
  int weights[] = {10, 60, 30, 40, 30, 20, 20, 2};
  int values[] = {1, 10, 15, 40, 60, 90, 100, 15};

  LocalSolver localsolver;
  LSModel* model = localsolver.getModel();

  // 0-1 decisions
  LSExpression* x[8];
  for (int i = 0; i < 8; i++)
    x[i] = model->createExpression(O_Bool);

  // knapsackWeight <- 10*x0 + 60*x1 + 30*x2 + 40*x3 + 30*x4 + 20*x5 + 20*x6 + 2*x7;
  LSExpression* knapsackWeight = model->createExpression(O_Sum);
  for (int i = 0; i < 8; i++)
    knapsackWeight->addOperand(model->createExpression(O_Prod, weights[i], x[i]));

  // knapsackWeight <= 102;
  model->addConstraint(model->createExpression(O_Leq, knapsackWeight, 102));

  // knapsackValue <- 1*x0 + 10*x1 + 15*x2 + 40*x3 + 60*x4 + 90*x5 + 100*x6 + 15*x7;
  LSExpression* knapsackValue = model->createExpression(O_Sum);
  for (int i = 0; i < 8; i++)
    knapsackValue->addOperand(model->createExpression(O_Prod, values[i], x[i]));
```

```
  // maximize knapsackValue;
  model->addObjective(knapsackValue, OD_Maximize);

  // close model, then solve
  model->close();
  LSPhase* phase = localsolver.createPhase();
  phase->setTimeLimit(1);
  localsolver.solve();

  return 0;
}
```

C++プログラムの構造に対応するLSP構造になっていることがわかります。まず最初に、入力データの読み込みを行います。その後、いくつかの表現式を作成し、ナップザック・モデルを宣言します。

モデルを終了させると、Windows上のソルバーを開始する前に探索をパラメータ化ができます。上記のプログラムがVisual Studio Command Prompt(ユーザーのLocalSolverのバージョンに従い、×86または×64)の中で、コンパイルされ、下記の行で実行します。

```
cl /EHsc toy.cpp -I%LS_HOME%\include /link %LS_HOME%\bin\localsolver.dll.lib
toy
```

Linux上では 下記の行を用いて同様のプログラムをコンパイルし、実行します。

```
g++ toy.cpp -I/opt/localsolver_3_1/include -llocalsolver -lpthread -o toy
./toy
```

プログラムのコンパイル及び実行で、トラブルが発生した場合、必要システム構成およびインストール手順を参照ください。また APIs の詳細情報は C++ API Reference を参照ください。

目次に戻る

Solving your first model in Java
（Java で最初のモデルを解いてみよう）

LocalSolverはC++言語で実装されています。オブジェクト指向アプリケーション・プログラミングインターフェースはJava5.0（または上位）向けに提供され、ユーザーのJavaビジネス・アプリケーション内に完全な統合が可能です。
LocalSolverのAPIsは処理するクラスが、軽量化されています。LocalSolverがカプセル化した数理計画ソルバーであることを前述しました。ソルバーを実行するため、モデルをインスタンス化し、追加のコードを記述する必要はありません。

下記は、quick tour of LocalSolver's modelerで紹介したナップザック・トイ例題を解くため、Javaコードで記述しています。
対応するソース・ファイルはexamples/toyで入手頂けます。

```
/********** Toy.java **********/

import localsolver.*;

public class Toy {

  public static void main(String [] args) {
    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, knapsackWeight, 102));

    // 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 model, then solve
    model.close();
    LSPhase phase = localsolver.createPhase();
    phase.setTimeLimit(1);
    localsolver.solve();
  }
}
```

Javaプログラムの構造に対応するLSP構造になっていることがわかります。まず、入力データの読み込みを行います。その後、いくつかの表現式を生成し、ナップザック・モデルを宣言します。モデルを終了させると、Windows上のソルバーを開始する前に探索をパラメータ化できます。コンパイルするために、Java Development kit5.0(または上位)をユーザーのコンピュータにインストールされている必要があります。Windows上では、上記のプログラムでコンパイルし下記の行で、実行します。

```
javac Toy.java -cp %LS_HOME%\bin\localsolver.jar
java -cp %LS_HOME%\bin\localsolver.jar;. -Djava.library.path=%LS_HOME%\bin\ Toy
```

Linux上では、上記のプログラムがコンパイルされ、下記の行で実行します。

```
javac Toy.java -cp /opt/localsolver_2_0/bin/localsolver.jar
java -cp /opt/localsolver_2_0/bin/localsolver.jar:. -Djava.library.path=/opt/localsolver_2_0/bin/ Toy
```

プログラムのコンパイル及び実行で、トラブルが発生した場合、必要システム構成およびインストール手順を参照ください。また APIs に関する詳細情報は C++ API Reference を参照ください。

目次に戻る

Solving your first model in C#
(C#で最初のモデルを解いてみよう)

LocalSolverはC++言語で実装されています。オブジェクト指向アプリケーション・プログラミングインターフェースは.NET2.0（または上位）向けに提供され、ユーザーの.NETビジネス・アプリケーション内に完全な統合が可能です。LocalSolverのAPIsは処理するクラスが、軽量化されています。LocalSolverがカプセル化した数理計画ソルバーであることを前述しました。ソルバーを実行するために、モデルをインスタンス化し、追加のコードを記述する必要はありません。

下記は、**quick tour of LocalSolver's modeler**で紹介したナップザック・トイ例題を解くために、C#コードで記述しています。対応するソース・ファイルはexamples/toyで入手頂けます。

```
/********** Toy.cs **********/

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, knapsackWeight, 102));

    // 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();

  }
}
```

Javaプログラムの構造に対応するLSP構造になっていることがわかります。まず、入力データの読み込みを行います。その後、いくつかの表現式を作成し、ナップザック・モデルを宣言します。モデルが終了すると、ソルバーを開始する前に探索をパラメータ化できます。

Windows上でVisual Studio Command Prompt (ユーザーのLocalSolverのバージョンに従い、x86 または x64)内で、上記のプログラムがコンパイルされ、下記の行で実行します。注:ユーザーのプログラムを構築するため、直接Visual studio IDEを使用する場合、ユーザーのプラットフォーム・ターゲット(ユーザーのLocalSolverバージョンにより×86または×64)をユーザーのvisual Studioプロジェクトのプロパティに指定してください。

```
copy %LS_HOME%\bin\*net.dll .
csc Toy.cs /reference:localsolvernet.dll
Toy
```

プログラムのコンパイル及び実行で、トラブルが発生した場合、必要システム構成およびインストール手順を参照ください。また APIs に関する詳細情報は.NETAPI Reference を参照ください。

目次に戻る

Step by step （ステップ ）

LocalSolver のモデリング言語およびソルバーC++、JAVA および.NET プログラミング・インターフェースを紹介してきました。ここでは LocalSolver を使用した、実践的な最適化問題の解き方を考察します。まずユーザーが最善のモデルをデザインし、LocalSolver を最大限活用するために必要な基本概念を紹介します。制約式が正しく設定されていることを前提とし、適切な定義済み目的関数（予約語）を使用し、意思決定変数を選択します。例題集から引用した複数の例題で上記の概念を解説致します。
車両製造工場の製造ラインに関する組合せ最適化問題として有名な車両割当て問題を解くためにこの概念を適用します。このガイドでは、入力データの読み込みからはじめ、最良の解を記述する作業までを段階的に進めていきます。

最終的に、利用可能な非線形演算子を使用し、意思決定変数の集合を指定し、その後、少し表現式の書き換え作業を行います。線形問題から LocalSolver モデルへの移行を考察します。
目次に戻る

Modeling Principles（モデリング基本概念）

LocalSolverは、効率的に、ユーザー定義の目的関数という観点から最適な解を探索し、ユーザーの最適化モデルで定義した探索領域を調べることができます。これは、全てのオペレーションズ・リサーチのアプローチと同様、「問題のモデル化」が重要な作業となります。
「明確な目的関数、制約の正しい設定、適切な意思決定変数の選択」の重要性を強調しています。

意思決定変数と内部変数を区別する

モデリング作業の最初のステップは、意思決定変数の集合を選択します。LocalSolverは、意思決定変数はバイナリ変数でなければなりません。バイナリ変数はキーワードboolを使用します。この変数は、取られる意思決定を表現しています。自身の意思決定変数を定義したら、これらの変数が、他の変数から推測できなかったか考察してください。

例えば、車両割当て問題（car sequencing problem）を考察します。

```
// A BAD SET OF DECISION VARIABLES FOR THE CAR SEQUENCING PROBLEM

// cp[c][p] = 1 if class c is at position p, and 0 otherwise
cp[1..nbClasses][1..nbPositions] <- bool();

// op[o][p] = 1 if option o appears at position p, and 0 otherwise
op[o in 1..nbOptions][p in 1..nbPositions] <- bool();

// constraints linking variables op[o][p] to cp[c][p]
for[o in 1..nbOptions][p in 1..nbPositions]
  constraint op[o][p] = or[c in 1..nbClasses : options[c][o]](cp[c][p]);
```

上記のモデルの中で、ユーザーは、制約の集合でリンクした、意思決定変数の2つの族を代入しました。このモデルは、このモデルの解が車両割当て問題の解であるという意味で有効です。しかし、あまりにも意思決定変数が多いので、このモデルは狭い探索領域を定義しています：明確にcp[c][p]変数の値を知るには、十分に解の定義を行うことで、他のすべての変数をこれら変数の値から予測することができます。従って、内部表現として、op[o][p]を代入した下記のモデルがより良い表現となります。

```
// 0-1 decisions:
// cp[c][p] = 1 if class c is at position p, and 0 otherwise
cp[1..nbClasses][1..nbPositions] <- bool();

// expressions:
// op[o][p] = 1 if option o appears at position p, and 0 otherwise
op[o in 1..nbOptions][p in 1..nbPositions] <- or[c in 1..nbClasses : options[c][o]](cp[c][p]);
```

完全主義の読者は、nbPositions意思決定変数が、まだ、cp[0][p] <- 1 - sum[i in 2..nbClasses]cp[i][p]を定義することで排除できることに気づかれたと思います。 さらによいものが良いものの敵となるため、過度な記述は控えてください。
より正確に、変数を個体としてではなく、変数の族として考えてください。この車両割当て問題がとても簡単な問題に見えてくることでしょう。
この問題よりも難解なケースは製鉄所のスラブ・デザインです。
この問題では、オーダーが様々なサイズのスラブに割り当てられます。目的は、使用される原材料(少なくとも1つのオーダーを持つスラブのサイズ)を最小化することです。この問題の簡単なモデルはオーダーoがスラブsに割り当てられるな

らば、様々なサイズのスラブを十分に定義し、x[o][s]変数が1に等しいことを完全に定義することです。このモデル、スラブの使用法及び、(内部表現)、目的関数内で直接使用されているこがわかります。

```
// slab s is used if at least one order is assigned to this slab
use[s in 1..nbSlabs] <- or[o in 1..nbOrders](x[o][s]);

// objective function
minimize sum[s in 1..nbSlabs](use[s] * size[s]);
```

このモデルは、CSPLib　インスタンス上で正確かつ、非常に適切に実行します。しかし、その内容からスラブのサイズが予測可能ならば、よりいっそう強靭なモデル構築ができます。各スラブ内容に対する各ムーブ後に実行し、最小かつ十分な容量のスラブに移動します。注:この問題に対する制約プログラミングモデルも同じアプローチを用います。下記の行はこのモデルの重要部分のみ抜粋しています。完全なモデルは例題集で入手頂けます。

```
for [j in 1..nbOrders] {
  slabContent[j] <- sum[i in 1..nbOrders](orders[i] * x[i][j]);
  constraint slabContent[j] <= maxSize;
  // threshold detection variables
  t[j][1] <- slabContent[j] > 0;
  t[j][k in 2..nbSlabSizes] <- slabContent[j] > slabSizes[k-1];
}

/* ... */

obj <- sum[j in 1..nbOrders](slabSizes[1] * t[j][1]
  + sum[k in 2..nbSlabSizes]((slabSizes[k] - slabSizes[k-1]) * t[j][k]))
  - sumSizeOrders;
```

制約式と最優先の目的を区別する

自意思決定変数の集合の選択後、制約、すなわち、実行可能だと見なした解を満たすために、論理式を選択します。これらの表現式にキーワードconstraintを行頭に付けます。ローカル・サーチは最適化問題に適しているので、良い解を発見することが難しいと感じるユーザー問題でも、比較的安易に実行可能解を発見できます。そして、ユーザー問題内にあるいくつかの「制約」が実際、最優先目的であるため、制約条件が問題の構造属性として保たれるべきであることを意味します。

http://www.csplib.orgに掲載されている車両割当て問題(car sequencing problem)とその定義を考察してください。
ある工場では、膨大な数の車両製造を行っています。:様々なオプションはベーシックモデルの変型として使用しているためオプションは同一ではありません。組み立てラインには、様々なオプションの取り付けを行う作業場が複数箇所あります。(エアコンの取り付け、サンルーフの取り付け等)これらの作業場では、組み立てラインに沿って通過している車両に対して一定のパーセンテージのみ処理するよう設計されています。(...) 各作業場の処理能力を越えないように、且つ車両を連続的に配置します。例えば、特定の作業場ではラインに沿って通過している車両の半分までの処理能力しか持ちません。オプションは1台の車両に最大2つのオプションを要求するよう生成します。
上記の定義を慎重に読んでいただくと、これが純粋な充足問題として表現されていることがわかります。
次に現実問題を考察します。:将来的に、製造される車両の過半数以上に、制約「1台の車両に最大2つのオプションを要求する」と設定されたオプションが必要な場合、自動車メーカーは何をすればよいのでしょうか？「解が見つからない」と返答する最適化システムは全然役に立たないことでしょう。この現実問題におけるこの例題では、他の制約(ビジネス制約と呼ぶ場合もある)が満たされるべきプロパティであり、制約のうち、いくつかが問題の構造を定義しています。実際、

それらすべてのプロパティを充足することが可能かどうか未知なため、自発的に、「should」という用語を使い、「must」という用語は使いません。ここで、唯一の構造制約は、組み立てラインの各作業場が少なくとも1台の車両を含んでいることです。ここで、ユーザーはいくつかの最適化モデルを定義して構いません。例えば、作業場の制約(任意Qにおける最大P車両の)は構造であり、最小化される目的関数が組み立てライン上の使用されていない作業場の数であると考えることができます。すなわち、組み立てラインの制約に左右される生産された車両数の最大化を行います。
この車両割当て問題は、通常、逆の観点で考えられています:すべての車両を生産しなければならず、作業場の制約の違反を計算し違反を最小化します。この計算はWindows上では単に、過剰設備の合計です。:

```
// expressions: compute the number of violations in each window
nbViolationsWindows[o in 1..nbOptions][p in 1..nbPositions-ratioDenoms[o]+1]
  <- max(nbCarsWindows[o][p]-ratioNums[o], 0);

// objective: minimize the sum of violations for all options and all windows
obj <- sum[o in 1..nbOptions]
  [p in 1..nbPositions-ratioDenoms[o]+1](nbViolationsWindows[o][p]);

minimize obj;
```

上記の通り、制約cを目的に変えることは必ずしも最大化cになるわけではありません。実際に、制約がしばし比較の形式を取るので、多くの場合、max(0、a−b)の最小化に、または<= bの充足a == bは充足dist(a、b)の最小化に変化します。ここで、純粋な充足モデルは最適化モデルに変り、それゆえ、目的関数は1つしかありません。そして、自身のオリジナル問題が、表現costを最小化する必要がある最適化問題の場合、そのいくつかの「制約」が識別できるなら、最優先目的(minimize violations)として生成すべきであり、その後、予約語で最適化される複合目的関数を持つようになります。:

```
minimize violations;
minimize cost;
```

このような場合、複数の指定制限時間を処理するのに、LocalSolverの能力が有効です。一般に、問題解決の時間が、合計60秒ある場合、設定しているlsTimeLimit={50、10}は、50秒がviolations (コストへの影響は考えない)の最小化に割かれる一方、残りの10秒で双方の目的を考慮(予約語)するために割かれます。最適解(ここでは0)が1秒で見つかった場合、当然のことながら、LocalSolverには、2番目の目的を最適化するために59秒残っていることになります。つまり、制約集合の選択時、下記の2つの点に配慮する必要があります。
・制約を追加しすぎると、LocalSolverが、解(戻された解のステータスは実行不可能)を全く発見できない、または実行可能解から別の解への移動が非常に困難となり、実行不可能解を導くムーブ率が高くなります。
・制約を目的に変えすぎると、構造体のないモデルになります。LocalSolverの強みは、実行可能性、従って「意味のある」ムーブを維持しつつ実行することです。

ユーザー定義の目的関数

LocalSolverの演算子により、条件(演算子iif)および高性能な非線形表現(product, max, square root,等)で、簡単に目的関数を記述できます。ごくまれに、オリジナル関数(ユーザー定義関数)をわずかに修正することでよりいっそう良い解を得られる場合もあります。例えば、Googleマシンの配置展問題(google machine reassignment problem)　で、目的関数の最初のデジタル化に重点を置くという多様性はとても参考になります。

```
// Classical technique for favoring diversification of the search when the
// objective value contains numerous digits. We iteratively optimize the most
// significant digits, passing from millions to hundred of thousands, etc.
minimize obj / 1000000;
minimize obj / 100000;
minimize obj / 10000;
minimize obj / 1000;
minimize obj;
```

注:この例題で、obj を直接最小化した結果、とても良い解を得ることができ、上記の簡単な改良で、数%の増加を実現させました。

目次に戻る

Solving your first business problem
（最初のビジネス問題を解いてみよう）

車両割当問題は自動車メーカーの生産ライン会社に関する有名な組合せ最適化問題です。クラスの中には様々なオプションのセットにより分類される車両の組をスケジューリングする問題が含まれています。組み立てラインは、様々なオプション(エアコン、サンルーフの取り付けなど)をチームによって設置する複数の作業場があります。そのような各作業場ではパラメータPとQによって与えられた一定の仕事量を処理する能力を持ちます：作業場は、Q車両の各連続でオプションを持つP車両を処理することができます。問題のゴールは、処理能力が最小化される個々のクラスとそれに与えられた要求を満たし、一連の車両製造を行うことです。すなわち、各オプションP/Q、および全長Qの各ウィンドウ、違反max(0、N-P)がカウントされ、Nは実際にこのウィンドウの内でスケジューリングされたオプション数です。このセクションで、問題向けに簡単なLSPモデルを提供致します。その中で、ファイルからどのようにデータの読み込みを行うのかを説明し、明確な意思決定、制約、および問題の目的を定義し、どうのように車両配列のモデリングを行うのかを解説します。またローカル・サーチをコントロールするために使用することができる複数のパラメータを導入します。この問題向けの完全なプログラムはC++、Java、およびC#バージョンを例題集から入手頂けます。

入力データを読み込む

この問題に関する複数のデータファイルは、folder localsolver/examples/carsequencingで入手頂けます。
こちらからダウンロード頂けます。
上記のファイル・フォーマット：
1行目：車両数、オプション数、クラス数
2行目：各オプションに対する、ブロックで（パラメータP）このオプションを使う車両の最大数
3行目：各オプションに対する、最大数を参照するための（パラメータQ）ブロックサイズ
そして、各クラスに対する、クラスのインデックス：このクラスにおける車両数：各オプションに対する。（このクラスがそれを（0または1）必要としているかに関わらず）下記は入力データを読み込むinput()関数です。
注：不確定な変数はnilと等しく、これは既存の変数の確認が可能です。ここで、inFileNameまたはsolfileNameが定義されていない場合、エラーが表示されます。単にファイルに（改行に関係なく）次々と整数を読み込みました。

```
/* Reads instance data. */
function input() {

  usage = "\nUsage: localsolver car_sequencing.lsp "
    + "inFileName=inputFile solFileName=outputFile [lsTimeLimit=timeLimit]\n";

  if (inFileName == nil) error(usage);
  if (solFileName == nil) error(usage);

  inFile = openRead(inFileName);
  nbPositions = readInt(inFile);
  nbOptions = readInt(inFile);
  nbClasses = readInt(inFile);

  ratioNums[1..nbOptions] = readInt(inFile);
  ratioDenoms[1..nbOptions] = readInt(inFile);

  for [c in 1..nbClasses] {
    readInt(inFile); // Note: index of class is read but not used
    nbCars[c] = readInt(inFile);
    options[c][1..nbOptions] = readInt(inFile);
  }
}
```
}

問題のモデリング

LSP言語で最適化問題をモデリングする上で、最も重要なステップの1つは、取りうる意思決定を定義することです。これらの意思決定は、それらをインスタンス化することで、モデル(特に制約と目的)のすべての他の表現を評価することが可能です。LocalSolverは、所与の目的と制約に対し最もよい実行可能な意思決定の発見を目指します。ここで、生産ラインにおける車両のオーダーを決定したいと思います。0-1モデリングで、ポジションpの車両がクラスcと0またはその他に属する時、cp[c][p]が1と等しくなるように、nbClasses*nbPositions変数cp[c][p]を定義しています。

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

ユーザー・モデルの制約定義は、これとは別の重要なモデリング作業です。
一般的に、制約は必要とされる要求のみにすべきです。車両割当問題で、同じ位置に2台の車両を与えることは物理的に不可能なため、すべてのP/Q比率を常に満たすことは不可能です。(目的とここで定義した理由です)。それが、探索スペースと1の解から別の解に移行する機能を定義するため、制約の使用を制限することはローカル・サーチのモデリングで特に重要です。ここで、私達は制約の2つの族に割り当てる要求を表現します。:
1つのポジションあたり、確実に1台の車両を与え、各クラスに対する車両の需要を満たします。

```
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;
```

注:目的を記述するために、複数の中間表現を導入します。まず、ポジションpの車両がオプションoを持ち0ではない場合、1と等しいことを表現するop[o][p]を宣言します。

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

実際、このポジションがこのオプション取るためにクラスの1つを取った場合、オプションoはポジションpで表します。この

ようなクラスの表はiteration [c in 1..nbClasses : options[c][o]]を使用し指定します。中間表現を定義するこの機能は、モデルをより読みやすく、より効率的にします。(その他、複数の表現で、この中間表現が使用される場合)
同様に、タイムオプションoの数は下記のように定義することで、ポジションjとポジションj+Q[o]-1間を表します。

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

最終的に、過剰設備のペナルティ関数は、上記の問題仕様書で定義された通りに記述することができます。

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

注:ここでは合計を表現するために、異なる方法が使われています。実際に、演算子sumを使用し合計を定義することができ、また算術演算子+、-も使用することが可能です。最後のステップは、目的関数(全ての違反の合計)を定義することです。

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

下記は車両割当問題に対応するLSPモデルの関数model()を解くコードです。

```
function model() {
  // 0-1 decisions:
  // cp[c][p] = 1 if class c is at position p, and 0 otherwise
  cp[1..nbClasses][1..nbPositions] <- bool();

  // constraints:
  // for each class c, no more than nbCars[c] assigned to positions
  for [c in 1..nbClasses]
    constraint sum[p in 1..nbPositions](cp[c][p]) == nbCars[c];

  // constraints: one car assigned to each position p
  for [p in 1..nbPositions]
    constraint sum[c in 1..nbClasses](cp[c][p]) == 1;

  // expressions:
  // op[o][p] = 1 if option o appears at position p, and 0 otherwise
  op[o in 1..nbOptions][p in 1..nbPositions] <-
    or[c in 1..nbClasses : options[c][o]](cp[c][p]);

  // expressions: compute the number of cars in each window
  nbCarsWindows[o in 1..nbOptions][p in 1..nbPositions-ratioDenoms[o]+1]
    <- sum[k in 1..ratioDenoms[o]](op[o][p+k-1]);

  // expressions: compute the number of violations in each window
  nbViolationsWindows[o in 1..nbOptions][p in 1..nbPositions-ratioDenoms[o]+1]
    <- max(nbCarsWindows[o][p]-ratioNums[o], 0);

  // objective: minimize the sum of violations for all options and all windows
  obj <- sum[o in 1..nbOptions]
    [p in 1..nbPositions-ratioDenoms[o]+1](nbViolationsWindows[o][p]);
  minimize obj;
}
```

ソルバーをパラメータ化する

いくつかのパラメータはローカル・サーチをコントロールする目的で関数param()を使用し定義します。

```
function param() {
  if (lsTimeLimit == nil) lsTimeLimit = 60;
  lsTimeBetweenDisplays = 3; // the number of seconds between each display
  lsNbThreads = 4;  // number of parallel searches launched
}
```

この最後の関数で、モデルが完成します。複数のコントロール・パラメータは、LSP ファイルで定義する代わりに、コマンド行で設定します。例えば、localsolver/examples/carsequencing/instances/carseq_500_8_20_08.の例題で、(localSolver ライセンスがサイズで制限されている場合、問題サイズが小さい例題 free_trial_cs を使用ください) 下記のコマンド行により、300 秒間で解に到達します。

```
> localsolver car_sequencing.lsp inFileName=instances/carseq_500_8_20_08.in solFileName=sol.txt
lsTimeLimit=30
```

全てのコントロール・パラメーターはセクションBuilt-in variables and functionsを、参照ください。
上記のコマンド行を使用し、LocalSolverをスタートさせると、下記のアウトプットを得ます。

```
LocalSolver 2.0 (build 20120213)
Copyright (C) 2012 Bouygues SA, Aix-Marseille University, CNRS.
All rights reserved (see Terms and Conditions for details).

Run input...
Run model...
Run param...
Run solver...

Model:
  expressions = 27001, operands = 75873
  decisions = 10000, constraints = 520, objectives = 1

Param:
  time limit = 30 sec, no iteration limit
  seed = 0, nb threads = 4, annealing level = 1

Objectives:
  Obj 0: minimize, no bound

Phases:
  Phase 0: time limit = 30 sec, no iteration limit, optimized objective = 0

Phase 0:

[3 sec, 308456 itr]: obj = (68), mov = 1248013, inf = 83.4%, acc = 0.8%, imp = 1380

[6 sec, 588195 itr]: obj = (52), mov = 2394957, inf = 79.9%, acc = 0.7%, imp = 1455

[9 sec, 874330 itr]: obj = (45), mov = 3616826, inf = 79.4%, acc = 0.7%, imp = 1482

[13 sec, 1155344 itr]: obj = (38), mov = 4768613, inf = 78.4%, acc = 0.7%, imp = 1509

[16 sec, 1428567 itr]: obj = (30), mov = 5891541, inf = 77.4%, acc = 0.7%, imp = 1529

[19 sec, 1721793 itr]: obj = (27), mov = 7036828, inf = 76.8%, acc = 0.7%, imp = 1540

[22 sec, 2011834 itr]: obj = (23), mov = 8198412, inf = 76.7%, acc = 0.7%, imp = 1550

[25 sec, 2270191 itr]: obj = (23), mov = 9295094, inf = 76.5%, acc = 0.7%, imp = 1553

[28 sec, 2567075 itr]: obj = (22), mov = 10405504, inf = 76.2%, acc = 0.7%, imp = 1557

[30 sec, 2730517 itr]: obj = (22), mov = 11071294, inf = 76.2%, acc = 0.7%, imp = 1562

2730517 iterations, 11071294 moves performed in 30 seconds
Feasible solution: obj = (22)

Run output...
```

3秒毎に、(パラメータIsTimeBetweenDisplaysに応じた)探索状況の要約リポートが表示されます:
経過した秒数と実行されたイタレーション数、目的関数の値は、予約語(セパレートを使用して)で表示されます。

・movは実行されたムーブの総数です。注:複数の探索が並行的に実行されるので、イタレーション数より多くなります。

・infは実行不可能なムーブ率です。(実行不可能解を導くムーブ)

・accは取られたムーブ率です。(シミュレーティドアニーリング手法で取られたコストを持つ実行可能解を導くムーブ)

・impは確実に改善するムーブの総数です。注:これは並行探索のすべてを蓄積するため、最善の目的関数が変更されない間は、この数の増加を確認することが可能です。IsVerbosity to 0.の設定で、全ディスプレイの解除が行えます。

解を記述する

探索の最後で、LocalSolverにより呼び出される関数output()を定義します。例えば、ここでいくつかの変数に割り当てられるべき名前を持つファイルに問題の解を記述するために、関数の定義を選択します。(例えば、コマンド行の中で)
下記のフォーマットで解を記述します。

1. 1 行目＝目的関数
2. 2行目＝nbポジションにポジション1のクラス

ファイルに記述するには、最初のパラメータとして出力ファイルを取り、通常、printとprintln関数を使用します。

```
function output() {
  solFile = openWrite(solFileName);
  println(solFile, getValue(obj));
  for [p in 1..nbPositions][c in 1..nbClasses : getValue(cp[c][p])]
    print(solFile, c-1, " ");
  println(solFile);
}
```

目次に戻る

How to migrate from MIP to LSP（MIP からLSP への移行方法）

LocalSolver は算術演算子、sum、product 及び演算比較を提供しているため、いかなる線形計画も、整数変数はバイナリ変数の加重和として記述されていれば、LocalSolverモデリング言語で直接、記述することができます。しかし、このようなモデルは、最大パフォーマンスを目的に自身のモデルで Local search および単一演算子を実行したい場合、適さない場合があります。

step by step example に掲載されている車両投入問題(car sequencing problem)を熟考してみましょう。ここではこの問題を代表的な線形計画として、LocalSolver のシンタックスを使用し記述しています。全ての変数はすでに定義されていることを前提としています。

```
// 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;
```

意思決定及び内部変数表現

モデリング概念の解説として、最も重要なモデリングの意思決定は意思決定変数の集合を選択することです。ここでは、他の変数、すべてが、cp[c][p]の変数によるものと推測されるため、cp[c][p]の集合が最も最良な意思決定変数です。ここで、内部変数は次のように、記述します。例えば、制約式 nbVehicles[o][j] == sum[k in 1..Q[o]](op[o][j+k-1])は条件 nbVehicles[o][j] <- sum[k in 1..Q[o]](op[o][j+k-1]).を定義する表現式に変わります。

線形化の代わりに非線形演算子を使用する

ここでは、このモデル内のいくつかの等式は実際に演算または論理表現の線形化であることを確認することができます。例えば制約式 op[o][p] >= sum[c in 1..nbClasses : options[c][o]](cp[c][p])は単に op[o][p]は、与えられた条件の一つが 1 と等しいとき、1 と等しいということを述べているだけです。もっとも自然な表現として、

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

最後に、violations[o][j]上の制約式の組がまさに、特定の数の変数の最大化を定義するための線形形式です。この表現を LocalSolver で直接記述すると下記のようになります。

```
violations[o][j] <- max( nbVehicles[o][j] -P[o], 0)
```

同様に、全ての最大化に関する線形化は、条件または論理積は演算子 max, iif や and を使用し直接 localSolver 形式に変換されます。区分的に、線形関数は単純に、同じように記述することが可能です。MIP が区分的線形関数（バイナリ変数と関連付け可能な）を含む場合、全ての間隔[c[i-1],c[i]] we have Y = a[i] * X + b[i] with i in (1..3)において、Y は f(X)に等しいとすることができます。その後すぐに、Y を下記のように定義します。

```
Y <- X < c[1] ? a[1]*X+b[1] : (X < c[2] ? a[2]*X+b[2] : a[3]*X+b[3]);
```

これらの変換後、下記のモデルを得ます。

```
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;
}
```

不必要な制約式を削除する

モデルに有効な不等式が含まれている場合、それらを削除することを推奨します。LocalSolver はモデルの緩和に頼らないため、これらの不等式はモデルに重みを付けるだけです。同様に制約を阻害する対称性を省く必要があります：LocalSolver はツリー探索に頼らないため、対称性を壊すことは実行可能解から他の解への移行をより困難にし、通常、LocalSolver が近傍のより良い解を探索する際に障害になります。

目次に戻る