Permutations Problems and how to solve them

Tony Hürlimann
info@matmod.ch

November 30, 2022

Abstract

This paper presents a class of problems that can be modeled as *permutation problems*. The goal is to show how these problems can be modeled in a mathematical modeling language and then be solved by a solver. In the introduction, a definition is given and the syntax is specified. In a second part, various implemented models in the modeling language LPL are given. The unique solver, I know of, that can solve these kind of models is LocalSolver, a very powerful and universal solver. The LPL modeling language contains an (experimental) interface to that solver, so all the models in this paper can directly be solved by LocalSolver. All models in this paper are also implemented in the LSP modeling language offered by LocalSolver.

Permutations problems is a class of problems in which a particular sequence or a partition into subsequences is to be found. Many interesting real applications result from this class, especially routing and scheduling problems. The class of practical problems that can be casted into this paradigm is astonishingly large. This paper gives an introduction and a (first) overview of several kinds with full examples in this class. The "proof of concept" that this works and can be formulated in a mathematical modeling language are the presented models in this paper.

A third part of this paper contains additional models coded in LPL that can be sent directly to LocalSolver: non-linear, MIP models or models with logical constraints, it shows that LocalSolver is a very powerful solver to solve all kind of models.

A ZIP file of all models can be found HERE

Contents

1 Introduction

The concept of *permutation problems* is introduced by an example. The traveling salesman problem (TSP) is an example for a problem that can be viewed as a permutation problem: Given a set of locations, we are looking for a tour –visiting all locations in a sequence– that minimizes the total distance traveled (see for a MIP implementation for this problem at tsp-1[1]). That is, given a set of locations $i, j \in I$ and a distance matrix $c_{i,j}$ between the locations, find a permutation (a sequence) that minimizes the total tour distance.
Example: Suppose there are $n = 9$ locations ($I = \{1, \ldots, n\}$) and a given 9×9 distance matrix c with

$$c = \begin{pmatrix} 0 & 31 & 24 & 30 & 40 & 39 & 50 & 51 & 66 \\ 31 & 0 & 50 & 38 & 20 & 57 & 53 & 41 & 76 \\ 24 & 50 & 0 & 26 & 52 & 20 & 42 & 52 & 50 \\ 30 & 38 & 26 & 0 & 30 & 20 & 20 & 26 & 39 \\ 40 & 20 & 52 & 30 & 0 & 50 & 38 & 22 & 62 \\ 39 & 57 & 20 & 20 & 50 & 0 & 26 & 42 & 30 \\ 50 & 53 & 42 & 20 & 38 & 26 & 0 & 20 & 24 \\ 51 & 41 & 52 & 26 & 22 & 42 & 20 & 0 & 43 \\ 66 & 76 & 50 & 39 & 62 & 30 & 24 & 43 & 0 \end{pmatrix}$$

then *any* permutation π of the locations defines a tour. For example, the permutation $\pi = [1, 2, 3, 4, 5, 6, 7, 8, 9]$ defines the following tour: start at location 1, go to location 2, then to 3, and so on until location 9 and back to location 1. The length of this tour is (where $j = i \mod n + 1$) :

$$c_{1,2} + c_{2,3} + c_{3,4} + \ldots + c_{8,9} + c_{9,1} = \sum_{i \in I} c_{\pi_i, \pi_j} = 342$$

With the permutation $\pi = [1, 3, 6, 9, 4, 7, 8, 5, 2]$, start at location 1, then go to location 3, then to 6 and so on, the length of this tour is much shorter (is it the shortest possible? – find out!) :

$$c_{1,3} + c_{3,6} + \ldots + c_{5,2} + c_{2,1} = 226$$

Let Π be the set of all permutations, then we can formulate the problem with the following model

$$\min_{\pi \in \Pi} \sum_{i \in I} c_{\pi_i, \pi_j} \quad , (\text{ where } j = i \mod n + 1)$$

How to solve such a model? A trivial way is to systematically generate all permutations and calculate the length of its tour, and then take the shortest one. This works only for problems with a few locations since the number of permutations grows exponentially with the number of locations. There are more powerful methods to generate permutations in a more "intelligent" way. Many heuristic or meta-heuristic methods can be used to find near-optimal solutions. To test the interface of my modeling language LPL to such a solver method, I implemented a primitive Tabu-Search solver already in 2001 (LPL version 4.41). It is still part of LPL and can be used for small instances. For commercial applications, LPL's recent version 6.91 (2022) contains an (experimental) interface to the powerful solver LocalSolver.

[1] https://lpl.matmod.ch/lpl/Solver.jsp?name=/tsp-1

Various problems can be formulated as permutation problems: the quadratic assignment problem (QAP) (see qap-1[2]), the linear ordering problem (LOP) (see lop[3]), even the sorting problem (see sorting[4]) could be formulated this way.[5]

The class of permutation problems, however, is much broader then just minimizing a simple permutation: We may look for a subset of a permutation, or we may have additional variables and constraints beside the minimization function, or we may looking for a partition of the permutation, etc. A systematical overview of the different variants is now given. All of them can contains additional variables and linear or non-linear constraints.

The following notation is use to specify a variable that denominates a permutation:

$$x_i \text{ alldiff}, \quad \text{forall } i \in I, \text{ and } I = \{1, \ldots, n\}, \ n > 0$$

This notation is a shortcut for

$$x_i \in I \text{ and } x_i \neq x_j \text{ forall } i, j \in I, i < j$$

that is, a permutation variable is a vector of integer in the range $[1, n]$ and all variables are different from each others. Note that the set I is an index-set, that is, the elements are ordered. In the modeling language LPL a similar notation is used (the keyword "variable" can even be missing) :

```
set i:=1..n;
alldiff variable x{i};
```

Let's go though various kind of permutation problems now:

1. **Minimizing a single permutation**: The permutation variable is one-dimensional and we are looking for a permutation x_i that minimize an expression (as seen above) :

   ```
   set i:=1..n;
   alldiff x{i};
   ```

   Problems examples are the TSP, the QAP (see below), or the LOP problem (see lop[6] and many others. These kind of problems – if they are small ($n < 200$)– can be "solved" with LPL's internal Tabu-Search solver, otherwise the commercial solver LocalSolver should be used.

2. **Minimizing single permutation with additional variables and constraints**: An example is the model aircraftLanding1 below.

3. **Minimizing a partial permutation**: The problem is to find a subset of the permutation. Let $m < n$ then only the variables $x_1, \ldots, x_m$ are assigned with a value in $[1, n]$, the rest has no value (in LPL they are zero). An example is the TSP with price collection, where only a part of the locations is visited (see below). In LPL the notation is (LPL uses the Quote Attribute to specify the different permutation problem classes) :

[2] https://lpl.matmod.ch/lpl/Solver.jsp?name=/qap-1
[3] https://lpl.matmod.ch/lpl/Solver.jsp?name=/lop
[4] https://lpl.matmod.ch/lpl/Solver.jsp?name=/sorting
[5] Sorting is an easy problem and many efficient algorithms have been developed to sort an array. Therefore, one should not use this permutation method to sort in a serious application.
[6] https://lpl.matmod.ch/lpl/Solver.jsp?name=/lop

```
set i:=1..n;
alldiff x{i} 'notall';
```

4. **Minimizing multiple permutations**: Suppose there are several machines on which the jobs must be processed in a particular order, on each in a different order, then we have to find multiple permutations. Let $k \in K$ with $K = \{1, \ldots, m\}$, $m > 0$ a set of machines, for instance, then we may define a 2-dimensional variable $x_{k,i}$ that maps the various permutations. For example with $n = 4$ and $m = 3$ we may have :

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

This matrix then models the situation that the 4 jobs on machine 1 must be ordered as $[1, 2, 3, 4]$, and on machine 2 as $[2, 4, 3, 1]$, etc. In LPL, the syntax is (note that the order in the indexes is important, the last (the second) must be the permutation index) :

```
set i:=1..n;
set k:=1..m;
alldiff x{k,i};
```

A model is the jobshop problem below.

5. **Minimizing a partitioned permutation**: Given a single permutation that is to be partitioned into m subsets $k \in K = \{1, \ldots, m\}$. An example is the capacitated vehicle routing problem: several trucks ($k \in K$) visit a subset of customers in a given sequence. This may be modeled again with a 2-dimensional matrix $x_{k,i}$. For example with a permutation of 9 items and $K = \{1, \ldots, 3\}$, we may have the following partition :

$$x = \begin{pmatrix} 1 & 5 & 6 & - & - & - & - & - & - \\ 9 & 4 & 3 & 1 & - & - & - & - & - \\ 7 & 8 & 2 & - & - & - & - & - & - \end{pmatrix}$$

The first subset (tour) of the partition is $[1, 5, 6]$, the second is $[9, 4, 3, 1]$, etc. Note that the order within the subset is important, note also that the remaining right part of the matrix is empty – the matrix only contains $n = 9$ entries). In LPL this is modeled as :

```
set i:=1..n;
set k:=1..m;
alldiff x{k,i} 'partition';
```

An application is given in model "capacitated vehicle routing problem" CVRP and the CVRPTW below.

6. **Minimizing a partitioned permutation without ordering**: A similar case as before is bin packing, with the exception that the ordering in the subset is not important: a list of items must be packed into a set of bags without exceeding their capacity. In LPL this permutation can be modeled as follows :

```
set i:=1..n  "items";
set k:=1..m  "bags";
alldiff x{k,i} 'set␣partition';
```

The binpacking model below shows a complete model. The models capaFacilityLocation1, assemblyLineBalancing1, and kmeans1 are also examples.

7. **Minimizing a covered permutation**: This is the same as the "partitioned permutation" of the CVRP model, with the exception that items can be repeated in the different subsets. This is the case in the *split delivery vehicle routing problem*: Several trucks visit a subset of customers, but two trucks can also visit the same customer whose demanded delivery is then split between the two trucks. Note that the ordering of the subset is important in this case. In LPL this permutation is modeled as :

```
set i:=1..n;
set k:=1..m;
alldiff x{k,i} 'cover';
```

The model below is the "Split Delivery Vehicle Routing Problem" SDVRP.

8. **Minimizing a disjoint permutation**: This is the same as the "partitioned permutation" of the CVRP model, with the exception that items must be disjoint pairwise in the different subsets. This is the case in the *capacitated arc routing problem*: Several trucks visit a subset of arcs. Note that the ordering of the subset is important in this case. In LPL this permutation is modeled as :

```
set i:=1..n;
set k:=1..m;
alldiff x{k,i} 'disjoint';
```

The model below is the "Capacitated Arc Routing Problem" CARP.

To illustrate the use of permutation variables in concrete modeling, I implemented several models that can be run directly in LPL (you need a licence from the LocalSolver solver to solve these models also). LocalSolver no only is a solver but it also contains a modeling language that is close to LPL. All the models below are also implemented in that language (see LocalSolver Example Tour. In this way, the reader may compare them one to one.

2 Traveling Salesperson Problem (tsp1)

Problem: Given a set of locations, find a roundtrip visiting all locations using a minimal distance. The problem has been explained in examp-tsp[7]. MIP formulations of the TSP can be found in my book Case Studies I.

All data instances used here are from TSBLib and the model is solved using the commercial solver LocalSolver.

Modeling Steps

The problem is formulated as a permutation problem as follows:

1. Let Π be the set of all permutations, and let π be a single permutation ($\pi \in \Pi$). For example, $\pi = [1, 2, 3, \ldots, n]$ is a single permutation. Let π_i be the i-th element in π.

[7] https://lpl.matmod.ch/lpl/Solver.jsp?name=/examp-tsp

2. Then the distance of a roundtrip of the permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n, \pi_1)$ can be formulated as :

$$\sum_{i=1}^{n} c_{\pi_i,\pi_j} \ , \qquad \text{where } j = i \mod n+1$$

j is the next element of i in the permutation π, and the next to the last is the first element in π, that is:

$$j = \begin{cases} i+1 & : \quad i < n \\ 1 & : \quad i = n \end{cases}$$

Another to formulate this (without an extra j) is :

$$\sum_{i=1}^{n-1} c_{\pi_i,\pi_{i+1}} + c_{\pi_n,\pi_1}$$

3. In other words: Let π be a permutation $[\pi_1, \pi_2, \pi_3, \ldots, \pi_{n-1}, \pi_n]$. Then c_{π_i,π_j} with $1 \leq i, j \leq n$ is the travel cost from the node π_i to node π_j. The summation, therefore, expresses the costs of a round trip: $c_{\pi_1,\pi_2} + c_{\pi_2,\pi_3} + \ldots + c_{\pi_{n-1},\pi_n} + c_{\pi_n,\pi_1}$. Each possible round trip can be generated by a permutation. Minimizing this sum over all permutations means to look for the shortest round trip. Hence, the traveling salesman problem can be formulated as following:

$$\begin{array}{ll} \min & \displaystyle\sum_{i=2}^{n} c_{\pi_{i-1},\pi_i} + c_{\pi_n,\pi_1} \\ \text{subject to} & \pi \in \Pi \end{array}$$

Note the syntax characteristic for this kind of models is, that (integer) variable are used as indexes: π_i is a variable. For example, $\pi_2 = 7$ means that 7 is at the second position (from left to right) within the permutation. And c_{π_i,π_j} is the distance value of $c_{h,k}$ where $h = \pi_i$ and $k = \pi_j$. This notation extension allows the modeler to formulated the problem directly in the modeling language LPL:

Listing 1: The Main Model implemented in LPL [2]

```
model tsp "Traveling Salesperson Problem";
  set i,j          "the node set";
  parameter c{i,j} "distance matrix";
  alldiff variable z{i} [1..#i] "a permutation";
  minimize obj: sum{i in 2..#i} c[z[i-1],z[i]] + c[z[#i],z[1]];
end
```

Further Comments: The LPL code is an one-to-one formulation of the model above. The variable definition `alldiff` defines a permutation of numbers starting with 1. The minimization function is also close to the mathematical notation. An alternative formulation of the objective function in LPL if to use the construct with `if` :

```
minimize obj: sum{i} c[if(i=1,z[#i],z[i-1]),z[i]];
```

This is a very compact formulation, but how is the problem solved? As already mentioned, there exist an experimental interface to the commercial solver LocalSolver. LPL generates source code of the lsp-language of LocalSolver. (Small instances could also be solved with the internal tabu-search solver of LPL).

Solution: The LPL model contains data instances from the symmetric and from the asymmetric tsp library (tsplib). The default data set is the symmetric one:

Listing 2: The Data Model

```
model data "data for symmetric tsps";
  string parameter File:='tsp-instances/fl417.tsp'; --opt=11861 ?
  --string parameter File:='tsp-instances/fl1400.tsp'; --opt=20127 ?
  --string parameter File:='tsp-instances/fl3795.tsp'; --opt=28772?
  parameter n;
  Read(File&',%1;-1:DIMENSION',n,n,n);
  i:=1..n;
  parameter X{i}; Y{i}; dum;
  Read{i}('%1:NODE_COORD_SECTION',dum,X,Y);
  c{i,j}:=Sqrt((X[i]-X[j])^2 + (Y[i]-Y[j])^2);
end
```

For an asymmetric data set use the following data model :

```
model data1 "data for asymmetric tsps";
  string parameter File:='tsp-instances/br17.atsp';  --opt=39
  --string parameter File:='tsp-instances/rbg443.atsp'; -opt=2720
  parameter n;
  Read(File&',%1;-1:DIMENSION',n,n);
  i:=1..n;
  Read('#%1:EDGE_WEIGHT_SECTION',{i,j} c);
end
```

For the three data set `fl417.tsp`, `fl1400.tsp`, `fl3795.tsp`, LocalSolver founds near optimal solutions within 20secs.

For the curious, LPL generates the following LSP code for LocalSolver. If the data are too large, then LPL export them to a file and the generate lsp-code to read it, but for smaller model the data are directly generated within the code. The code generated for the `fl417.tsp` instance is as follows :

```
// LSP file (LocalSolver.com) :: automatically generated by LPL
use io;
function input() {
  I = 417;
  c = { ...distance data here... };
}

function model() {
  z <- list(I);
  constraint count(z) == I;
  obj <-  sum (1..I-1, i => c[z[i-1]][z[i]])+c[z[I-1]][z[0]];
  minimize obj;
}

function param() {
  if (lsTimeLimit == nil) lsTimeLimit = 100;
  lsVerbosity = 2;
}

function output() {
  if (solFileName == nil) return;
  local solFile = io.openWrite(solFileName);
  solFile.println(getSolutionStatus());
  solFile.println(obj.value);
  solFile.print("z:␣");
  solFile.println(z.value);
}
```

3 Quadratic Assigment Problem (qap1)

Problem: The problem is explained in qap-1[8]. MIP formulations of the QAP problem can be found in my book Case Studies II.

All data instances used here are from QAPLIB.

Modeling Steps

The problem is formulated as a permutation problem as follows:

1. Given a set of locations $i, j \in I$, the distance matrix $A_{i,j}$ and the quantity of items that flows between two location as $B_{i,j}$
2. A permutation variable p_i is given over the set Π of all permutations, the model is:

$$\begin{array}{ll} \min & \sum_{i,j \in I} A_{i,j} \cdot B_{p_i,p_j} \\ \text{subject to} & p \in \Pi \end{array}$$

In LPL code this model is

Listing 3: The Main Model implemented in LPL [2]

```
model qap "Quadratic Assigment Problem";
  set i,j "items/locations";
  parameter A{i,j}; B{i,j};
  alldiff p{i};
  minimize obj: sum{i,j} A[i,j] * B[p[i],p[j]];
end
```

Solution: The data model is:

Listing 4: The Data Model

```
model data;
  string parameter File:='qap-instances/kra32.dat'; --opt=88900
  parameter n; dum;
  Read(File,n,dum);
  i:=1..n;
  Read{i}('@%1;1',{j} A);
  Read{i}('@%1',{j} B);
end
```

LocalSolver returns a solution of 89500 after 5secs (the optimum is 88900).

[8] https://lpl.matmod.ch/lpl/Solver.jsp?name=/qap-1

4 Aircraft Landing (aircraftLanding1)

Problem: The landing times of a set of planes have to be scheduled. Each plane can land in a predetermined time window around a target time. The objective is to minimize the deviation due to landings before or after target times. A separation time has to be respected between two successive planes. This problem is from LocalSolver. All data instances used here are from Beasley, OR-Library.

Modeling Steps

The problem is formulated as a permutation problem as follows:

1. Given a set of planes $p, q \in P$. For each plane the earliest time eT_p, the target time tT_p, the latest time lT_p, the earliness cost eC_p, and the tardiness cost tC_p is given. Furthermore, a separation time $sT_{p,q}$ between the two planes p and q must be observed.

2. We are looking for a permutation x_p of the landing order of the planes. In addition, for each plane a preferred time is defined as a variable $y_p \geq 0$, which must be in the interval $[eT_p, tT_p]$.

3. The landing time LT_p of a plane x_p (that is the plane p at position x_p) is the maximum between the prefered time and (if the plane x_p is not at the first position within the order) the landing time of the previous plane and the separation time between them. This can be define as:

$$LT_p = \max\left(pT_{x_p}\ ,\ \text{if}(p > 1, LT_{p-1} + sT_{x_{p-1},x_p})\right) \quad \text{forall } p \in P$$

4. The earliness and tardiness cost CC_p of a plane at position p is as follows: if the landing time is smaller then the target time then it is the earliness cost, else the tardiness cost multipled by the distance from the target time:

$$CC_p = \text{if}(LT_p < tT_{x_p}, eC_x, tC_{x_p}) \cdot |LT_p - tT_{x_p}| \quad \text{forall } p \in P$$

5. The first constraint that must hold is: the landing time of a plane at position p must be equal or larger than the landing time of all planes in an previous position plus the separation time:

$$LT_p \geq \max_{q \in P} (LT_q + sT_{x_q,x_p}) \quad \text{forall } p \in P, p > 1$$

6. A second hard constraint requires that the landing time must be before the latest time:

$$LT_p \leq lT_{x_p} \quad \text{forall } p \in P$$

7. We want to minimize the "costs", that is, the deviation of landing time from the target time:

$$\min \sum_{p \in P} CC_p$$

The model in LPL syntax is as follows:

Listing 5: The Main Model implemented in LPL [2]

```
model landing "Aircraft Landing";
  set p,q "a set of planes";
  parameter nbPlanes; dum;
  parameter earliestTime{p}; targetTime{p}; latestTime{p};
    earlinessCost{p}; tardinessCost{p}; separationTime{p,q};
  alldiff x{p}  "landingOrder";
  integer variable preferredTime{p} [earliestTime..targetTime];
  expression landingTime{p}: Max(preferredTime[x],
    if(p>1, landingTime[p-1] + separationTime[x[p-1],x[p]]));
  constraint C1{p|p>1}: landingTime >= max{q in 1..p}(landingTime[q]
              + separationTime[x[q],x[p]]);
  constraint C2{p}: landingTime <= latestTime[x];
  expression cost{p}: if(landingTime<targetTime[x] , earlinessCost[x] ,
              tardinessCost[x]) * Abs(landingTime - targetTime[x]);
  minimize totalCost: sum{p} cost;
end
```

Solution: With the data file "airland1txt", the optimum is found immediately. With the data "airland9.txt", LocalSolver returns the near optimal solution of 5642 (opt=5611.7) after 100secs.

Listing 6: The Data Model

```
  model data;
    string parameter File:='ai-instances/airland1.txt'; --opt=700
    --string parameter File:='ai-instances/airland9.txt'; --opt=5611.7
    Read(File,nbPlanes);
    p:=1..nbPlanes;
    set k:=1..nbPlanes+6;
    parameter mat{p,k};
    Read('#%1;1',{k,p} mat);
    earliestTime{p}:= mat[p,2];
    targetTime{p}  := mat[p,3];
    latestTime{p}  := mat[p,4];
    earlinessCost{p} := mat[p,5];
    tardinessCost{p} := mat[p,6];
    separationTime{p,q} := mat[p,q+6];
  end
```

Figure 1 show the landing intervals and the effective landing time of the "airland1.txt" instance in a diagram.

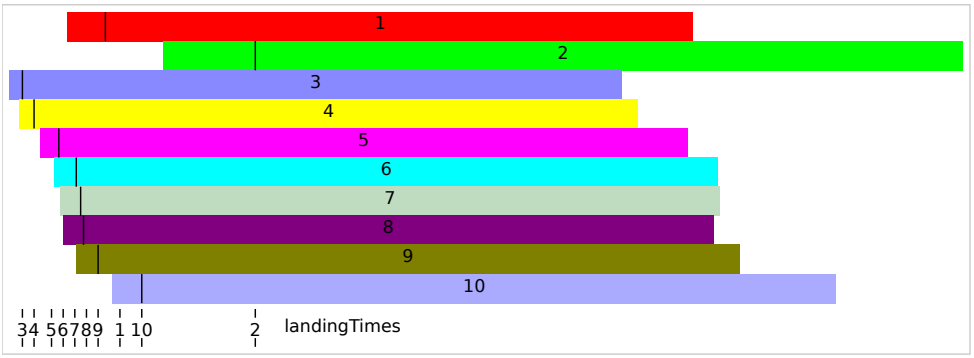

Figure 1: Solution of "airland1.txt"

5 TSP with Price Collection (tsp-pc)

Problem: Given a set of locations, find a roundtrip visiting a subset of locations by maximizing the revenue and minimizing the distance. This model is known as the TSP with price collection: A location is visited if its reward is higher than the cost of the traveled distance to it.

Modeling Steps

The problem is formulated as a permutation problem as follows:

1. Let $i, j \in I$ be a set of locations. The distance between location i and j is given by $c_{i,j}$, and the reward (price) of location i is p_i.
2. We are looking for a subset tour, that is, for a (partial) permutation z_i.
3. Let C be the number of locations that are visited (note that count is an index operator that returns the number of elements in the list):

$$C = \sum_{i \in I | z_i} 1 \quad \text{or} \quad \underset{i}{\text{count}}\ z_i$$

4. The distance traveled D is the sum of the distances traveded from location to location (note that C is a variable and must be calculated for each generated permutation) :

$$D = \text{if}(C > 1, \sum_{i \in 1..C} c_{\text{if}(i=1, z_C, z_{i-1}), z_i})$$

5. The total revenue R collected from the visited location is:

$$R = \sum_{i \in 1..C} p_{z_i}$$

6. The minimizing function could be given as (maximize revenue minus distance) :

$$\max\ R - D$$

Listing 7: The Main Model implemented in LPL [2]

```
model tsp "TSP with Price Collection";
  set i,j          "the node set";
  parameter c{i,j} "distance matrix";
  parameter price{i};
  alldiff variable z{i} 'notall' "a permutation";
  expression cnt: count{i} z;
  expression distance:
    if(cnt>1, sum{i in 1..cnt} c[if(i=1,z[cnt],z[i-1]),z[i]]);
  expression revenue: sum{i in 1..cnt} price[z[i]];
  maximize obj: revenue - distance;
end
```

Solution: With a random data set defined in the data model, the solution is displayed in Figure 2 generatd by the output model. The first number within the circle is the ID and the second is the reward.

Listing 8: The Data Model

```
model data;
  parameter n:=[30] "Problerm Size";
    X{i}; Y{i}; m:=Trunc(Sqrt(n));;
  i:=1..n;
  X{i}:=(i%m+1)*2+Trunc(Rnd(0,2));
  Y{i}:=(i/m+1)*2+Trunc(Rnd(0,2));
  c{i,j}:= 10*Sqrt((X[j]-X[i])^2+(Y[j]-Y[i])^2);
  price{i}:=Trunc(if(Rnd(0,1)>.3,Rnd(0,100)));
end
```

Listing 9: The Output Model

```
model output friend data;
  Draw.Scale(30,30);
  Draw.DefFont('Verdana',8);
  {i|i<=cnt} Draw.Line(X[z],Y[z],X[z[i%cnt+1]],Y[z[i%cnt+1]],3,3);
  {i} Draw.Circle(i&'/'&price,X,Y,.45,1,0);
end;
```

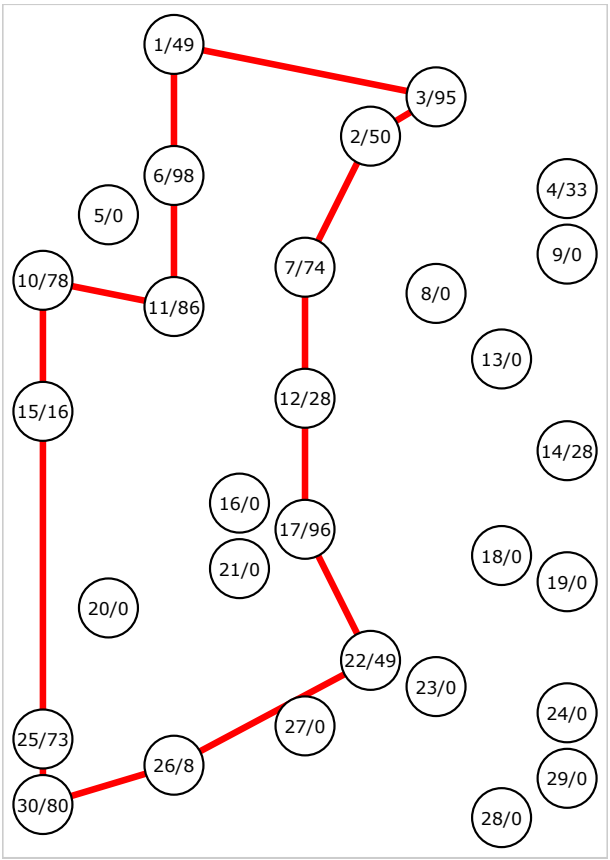

Figure 2: Solution to the model

6 Jobshop Problem (jobshop1)

Problem: The problem is explained in jobshop[9]. Several MIP formulations are explained in my book Case Studies II.

Implement this problem as a permutation problem.

Modeling Steps

Given a set of jobs $i \in I$ and a set of machines $k \in K$. The processing time of each job on each machine is given as $pt_{i,k}$, and the order of the job i on a machine k is $o_{k,i}$. An (trivial) upper bound for the overall processing time is $mS = \sum_{i,k} pt_{i,k}$. Let n be the cardinality of the set I and m be the cardinality of the set K.

1. The two variables are the starting time of a job on a machine: $s_{i,k}$ and the permutation of the jobs on each machine: $x_{k,i}$.

2. The finishing time $F_{i,k}$ of a job i on machine k is given as:

$$F_{i,k} = s_{i,k} + pt_{i,k} \quad \text{forall } k \in K, i \in I$$

3. The precedence constraint says that the starting time of a job i must be at least as large as the finishing time on the previous machine :

$$s_{i,o_{i,k}} \geq F_{i,o_{i,k-1}} \quad k \in K, i \in I, k > 1$$

4. The disjoint constraint says that on each machine k all the starting time of a job at position $x_{k,i+1}$ must be larger or equal the finishing time of a job at position $x_{k,i}$:

$$\bigwedge_{i \in 1..n-1} (s_{x_{k,i+1},k} \geq F_{x_{k,i},k}) \quad \text{forall } k \in K$$

5. The goal is to minimize the makespan:

$$\min \max_{i \in I} F_{i,o_{i,m}}$$

Listing 10: The Main Model implemented in LPL [2]

```
model jobshop "Jobshop Problem";
  set i  "a set of jobs";
      k  "a set of machines";
  parameter processingTime{i,k}; mOrder{i,k}; maxStart;
  integer variable start{i,k} [0..maxStart];
  alldiff variable x{k,i} [1..#i];
  expression finish{i,k}: start + processingTime;
  constraint Prec{i,k|k>1}:
     start[i,mOrder[i,k]] >= finish[i,mOrder[i,k-1]];
  constraint Disj{k}:
     and{i in 1..#i-1} (start[x[k,i+1],k] >= finish[x[k,i],k]);
  minimize makespan: max{i} finish[i,mOrder[i,#k]];
end
```

[9] https://lpl.matmod.ch/lpl/Solver.jsp?name=/jobshop

Solution: The data model is

Listing 11: The Data Model

```
model data;
  string parameter File:='js-instances/la40.txt';     //opt=1222
  --string parameter File:='js-instances/ft20.txt';  //opt=1165
  parameter nJobs; nMach;
  parameter t1{i,k};
  Read(File&',%1;1',nJobs,nMach);
  i:=1..nJobs;
  k:=1..nMach;
  Read{i}('%1:Times',{k} t1);
  Read{i}('%1:Machines',{k} mOrder);
  {i,k} (processingTime[i,mOrder]:=t1);
  maxStart:=sum{i,k} processingTime;
end
```

The LocalSolver returns a solution of 1284 (the optimum is 1222) after 10secs for the "la40.txt" instance. The Gantt diagram generated by the following output model is given in Figure 3.

Listing 12: The Output Model

```
model output;
  Writep(makespan);
  Draw.Scale(1,40);
  {i,k} Draw.Rect(i&'',start,k,processingTime,1,i+2,0);
end;
```

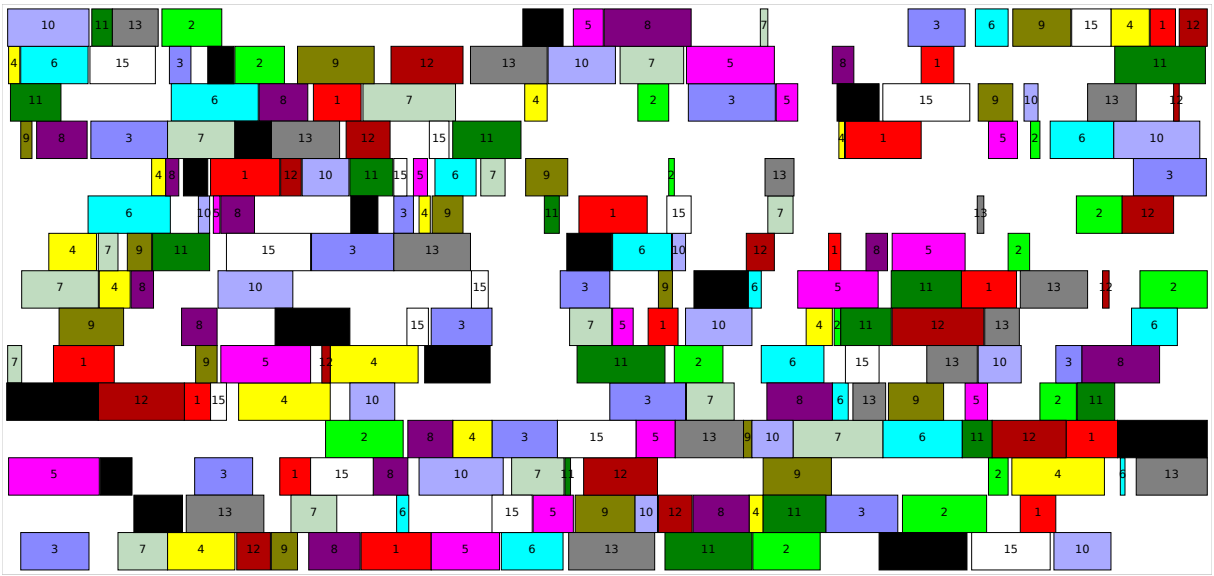

Figure 3: Near Optimal Solution of the instance "la40.txt"

7 Capacitated Vehicule Routing Problem (cvrp1)

Problem: A fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. An concrete application is given in cvrp[10]. The models cvrp-1[11], cvrp-2[12], and cvrp-3[13] give MIP-formulations of the problem. The modeling of these problems is explained in my book Case Studies I.
The data instances are from NEO Research Group

Modeling Steps

Let $i, j \in I = \{1, \ldots, n-1\}$ be a set of customer locations (n is the number of locations including the depot (or the depot)) and let $k \in K = \{1, \ldots, m\}$ be a set of trucks. Let the capacity of each truck be CA, and the demand quantity to deliver to a customer i be dem_i. Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is dw_i.

The customers are enumerated with integers from 1 to $n-1$. If there were one single truck (that must visits all customers), every permutation sequence of the numbers 1 to $n-1$ would define a legal tour. Since we have m trucks, to each truck a sequence of a subset of 1 to $n-1$ must be assigned. Hence, each truck starts at the depot, visiting a (disjoint) subset of customers in a given order and returns to the depot.

The variables can now be formulated as a "partitioned permutation". Example: if we had 3 trucks and 10 customers then the 3 subset sequences:

$$\{\{2,3,4\},\{1,9,8,5\},\{6,7,10\}\}$$

defines a partitioning between the 3 trucks. It means, for example, that truck 1 starts at the depot visiting customers 2, 3, 4 in this order and returns to the depot. In LPL, we can declare these partitioned permutation with a permutation variable $x_{k,i} \in [1 \ldots n-1]$. For the example above, the content would be:

$$x = \begin{pmatrix} 2 & 3 & 4 & - & - & - & - & - & - & - \\ 1 & 9 & 8 & 5 & - & - & - & - & - & - \\ 6 & 7 & 10 & - & - & - & - & - & - & - \end{pmatrix}$$

For example: $x_{1,1} = 2$ means that the customer number 2 is visited by the truck 1 right after the depot, $x_{1,2} = 3$ means that the next customer (after 2) is 3, etc.

1. Let C_k be the (unknown) size of the k subtour (without depot):

$$C_k = \operatorname*{count}_i \; x_{k,i}$$

2. Whether a truck is used or not is defined by uT_k and the (unknown) number of trucks T can be formulated as :

$$uT_k = C_k > 0, \text{ forall } k \in K \quad , \quad T = \sum_k uT_k$$

[10] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp
[11] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp-1
[12] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp-2
[13] https://lpl.matmod.ch/lpl/Solver.jsp?name=/cvrp-3

3. The route distance D_k of truck k is the sum of its partition plus the distance from the depot to the first customer and the distance back to the depot from the last visited customer :

$$D_k = \sum_{i \in 2..C_k} d_{x_{k,i-1},x_{k,i}} + \text{if}(C_k > 0, dw_{x_{k,1}} + dw_{x_{k,C_k}}) \quad \text{forall } k \in K$$

4. The constraint is the capacity of the trucks that cannot be exceeded, that is, each subset sequence must be chosen in such a way that the capacity of the truck is larger than the cumulated demand of the customers that it visits :

$$\sum_{i|x_{k,i}} dem_{x_{k,i}} \leq CA \quad \text{forall } k \in K$$

5. The total distance of all trucks is to be minimized (alternatively) we may first minimize the used trucks and then the distance) :

$$\min \sum_{k} D_k$$

Listing 13: The Main Model implemented in LPL [2]

```
model cvrp "Capacitated Vehicule Routing Problem";
  set i,j  "customers";
      k    "trucks";
  parameter
      d{i,j} "distances";
      dw{i}  "distance from/to the depot";
      dem{i} "demand";
      CA     "truck capacity";
  alldiff x{k,i} 'partition';
  expression
    cc{k}: count{i} x;
    trucksUsed{k}: cc[k] > 0;
    routeDistances{k}: sum{i in 2..cc} d[x[k,i-1],x[k,i]]
       + if(cc[k]>0, dw[x[k,1]] + dw[x[k,cc[k]]]);
    nbTrucksUsed: sum{k} trucksUsed[k];
    totalDistance: sum{k} routeDistances[k];
  constraint CAP{k}: sum{i|x} dem[x[k,i]] <= CA;
  --minimize obj1: nbTrucksUsed;
  minimize obj2: totalDistance;
end
```

Solution: LocalSolver finds the optimal solution of the instance "A-n32-k5.vrp" with 2secs and Figure 4 displays the optimal tours, a graph generated by the output model. The data model in LPL is :

Listing 14: The Data Model

```
model data;
  string parameter File:='cvrp-instances/A-n32-k5.vrp';    //opt=784
  --string parameter File:='cvrp-instances/A-n45-k6.vrp'; //opt=944
  set h; //h is i plus 1, 1 is depot (warehouse)
  parameter de{h}; n; m; X{h}; Y{h}; string typ; dum;
```

```
  Read(File&',%1;-1:DIMENSION',dum,dum,n);
  Read('%1;-1:EDGE_WEIGHT_TYPE',dum,dum,typ);
  if typ<>'EUC_2D' then Write('Only␣EUC_2D␣supported\n'); return 0;
     end;
  Read('%1;-1:CAPACITY',dum,dum,CA);
  Read{h}('%1:NODE_COORD_SECTION:DEMAND_SECTION', dum,X,Y);
  Read{h}('%1:DEMAND_SECTION:DEPOT_SECTION', dum,de);
  m:=Ceil(sum{h} de/CA);
  k:=1..m;    i:=1..n-1;
  d{i,j}:=Round(Sqrt((X[i+1]-X[j+1])^2+(Y[i+1]-Y[j+1])^2));
  dem{i}:=de[i+1];
  dw{i}:=Round(Sqrt((X[i+1]-X[1])^2+(Y[i+1]-Y[1])^2));
end
```

Listing 15: The Output Model

```
model output friend data;
  parameter y{k,i};
  {k} (y[k,1]:=1, y[k,cc+2]:=1,{i|i<=cc} (y[k,i+1]:=x+1));
  Draw.Scale(5,5);
  {k,i in 1..cc+1} Draw.Line(X[y],Y[y],X[y[k,i+1]],Y[y[k,i+1]],k+3,3)
     ;
  {k,i in 1..cc+2} Draw.Circle(y&'',X[y],Y[y],2,1,0);
  Write{k}('Tour␣%d:␣%3d␣␣\n',k,{i in 1..cc+2} y[k,i]);
  Write{k}('Tour␣Load␣%d:␣%d␣<=␣%d\n',k,sum{i|x} dem[x[k,i]],CA);
end
```

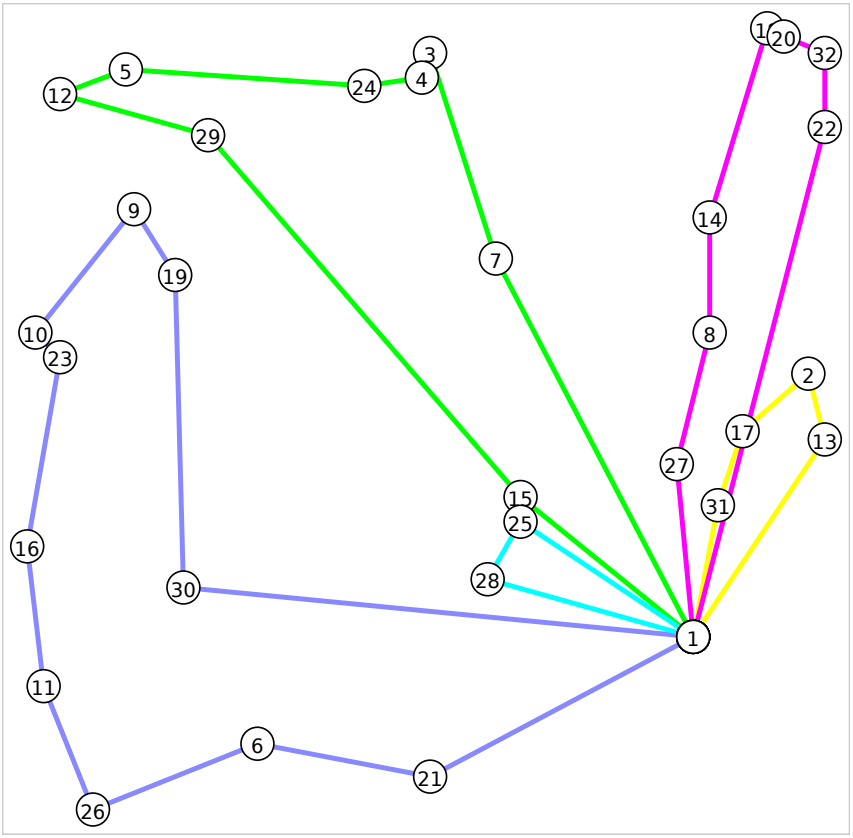

Figure 4: Optimal Solution of "A-n32-k5.vrp"

8 CVRP with Time Windows (cvrptw1)

Problem: The "capacitated vehicle routing problem with time window" (CVRPTW) is similar to the CVRP with the exception that each customer should be served within a given time interval.

The data instances used here are the Solomon instances.

Modeling Steps

Let $i, j \in I = \{1, \ldots, n\}$ be a set of customer locations (n is the number of locations excluding the depot) and let $k \in K = \{1, \ldots, m\}$ be a set of trucks. Let the capacity of each truck be CA, and the demand quantity to deliver to a customer i be dem_i. Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is given by dw_i. In addition, for each customer i, there is a service time sT_i, a earliest starting time eS_i and a latest ending time lE_i. Finally, we specify a maximum time horizon tH (for each truck the same), that is, a time that all trucks should be home if not to be late. As in the CVRP model, we are looking for a "partitioned permutation": assign a (ordered) subset of customers i to each truck k : $x_{k,i}$

1. Let C_k be the (unknown) size of the k subtour (without depot):

$$C_k = \operatorname{count}_i x_{k,i}$$

2. Whether a truck is used or not is defined by uT_k and the (unknown) number of trucks T can be formulated as :

$$uT_k = C_k > 0, \text{ forall } k \in K \quad , \quad T = \sum_k uT_k$$

3. The route distance rD_k of truck k is the sum of its partition plus the distance from the depot to the first customer and the distance back to the depot from the last visited customer :

$$rD_k = \sum_{i \in 2..C_k} d_{x_{k,i-1}, x_{k,i}} + if(C_k > 0, dw_{x_{k,1}} + dw_{x_{k,C_k}}) \quad \text{forall } k \in K$$

4. The unique constraint is the capacity of the trucks that cannot be exceeded, that is, each subset sequence must be chosen in such a way that the capacity of the truck is larger than the cumulated demand of the customers that it visits :

$$\sum_{i | x_{k,i}} dem_{x_{k,i}} \leq CA \quad \text{forall } k \in K$$

5. Till now the model is identical with the CVRP. In addition, the time windows should hold. First, let's define the ending time $eT_{k,i}$ at each customer: it is the arrival time plus the service time. The arrival time is the ending time of the previous customer in a tour k plus the travel time between them, and for the first customer in a tour it is the travel time from the depot. In any case, the arrival time must be at least the earliest starting time (if the truck arrives ealier then it must wait until the opening of the customer store) :

$$eT_{k,i} = \max(eS_{x_{k,i}}, \text{if}(i = 1, dw_{x_{k,1}}, eT_{k,i-1} + d_{x_{k,i-1}, x_{k,i}}) + sT_{x_{k,i}}) \quad \text{forall } k \in K, i \in 1..C_k$$

6. The lateness at home lH_k (arrival time at the depot after the tour) is zero, if the truck k arrives not later than the maximum time horizon tH, otherwise it is the ending time at the last customer plus the travel time back to the depot minus the maximum time horizon:

$$hL_k = \text{if}(tU_k, \max(0, eT_{k,C_k} + dw_{x_{k,C_k}} - mH)) \quad \text{forall } k \in K$$

7. The lateness L_k of a whole tour k is the home lateness plus the sum of all latenesses at a customer :

$$L_k = hL_k + \sum_{i \in 1..C_k} \max(0, eT_{k,i} - lE_{x_{k,i}}) \quad \text{forall } k \in K$$

8. From these previous definitions, three objectives to be minimized can be derived: minimizing the total lateness TL, the number of trucks used TU, and the total distances traveled TD:

$$TL = \sum_k L_k$$

$$TU = \sum_k tU_k$$

$$TD = \sum_k rD_k$$

9. Various versions to combined these objectives can be implemented: In this case, preemptive multi-objective minimization is used: first minimise total lateness, then the number of trucks used, and finally the total distances.

Listing 16: The Main Model implemented in LPL [2]

```
model cvrptw "CVRP with Time Windows";
 set i,j  "customers";
      k    "trucks";
  parameter
      d{i,j} "distances";
      dw{i}  "distance from/to the warehouse";
      dem{i} "demand";
      CA     "truck capacity";
      serviceTime{i}; earliestStart{i}; latestEnd{i}; maxHorizon;
  alldiff x{k,i} 'partition'  "sequences";
  expression cc{k}: count{i} x;
  expression truckUsed{k}: cc[k] > 0;
  expression endTime{k,i in 1..cc}: Max(earliestStart[x[k,i]],
    if(i=1,dw[x[k,1]] , endTime[k,i-1] + d[x[k,i-1],x[k,i]])
                       + serviceTime[x[k,i]]);
  expression homeLateness{k}: if(truckUsed,
                Max(0, endTime[k,cc] + dw[x[k,cc]] - maxHorizon));
  expression routeDistances{k}: sum{i in 2..cc} d[x[k,i-1],x[k,i]]
       + if(cc[k]>0, dw[x[k,1]] + dw[x[k,cc[k]]]);
  expression lateness{k}: homeLateness + sum{i in 1..cc}
      Max(0, endTime[k,i] - latestEnd[x[k,i]]);
  constraint CAP{k}: sum{i|x} dem[x[k,i]] <= CA;
  expression totalDistance: sum{k} routeDistances[k];
  expression totalLateness: sum{k} lateness[k];
```

```
  expression nbTrucksUsed: sum{k} truckUsed[k];
  minimize obj1: totalLateness;
  minimize obj2: nbTrucksUsed;
  minimize obj3: totalDistance;
end
```

Solution: Three minimization functions are defined and they are treated in a preemptive way, that is, 33% of the solver time is used for the first, 33% for the second objective function, etc. The instance data model generates the solution in Figure 5

Listing 17: The Data Model

```
model data;
  parameter nbTrucks;
    wIndex; wX; wY; X{i}; Y{i};
  string parameter File:='cvrptw-instances/C101.25.txt';
  Read(File&',%1;4',nbTrucks,CA);
  k:=1..nbTrucks;
  Read('%1;9',wIndex,wX,wY,maxHorizon=6);
  Read{i}('%1;10',i,X,Y,dem,serviceTime=7,earliestStart=5,latestEnd
      =6);
  latestEnd{i}:=latestEnd+serviceTime;
  d{i,j}:=Sqrt((X[i]-X[j])^2+(Y[i]-Y[j])^2);
  dw{i}:=Sqrt((X[i]-wX)^2+(Y[i]-wY)^2);
end;
```

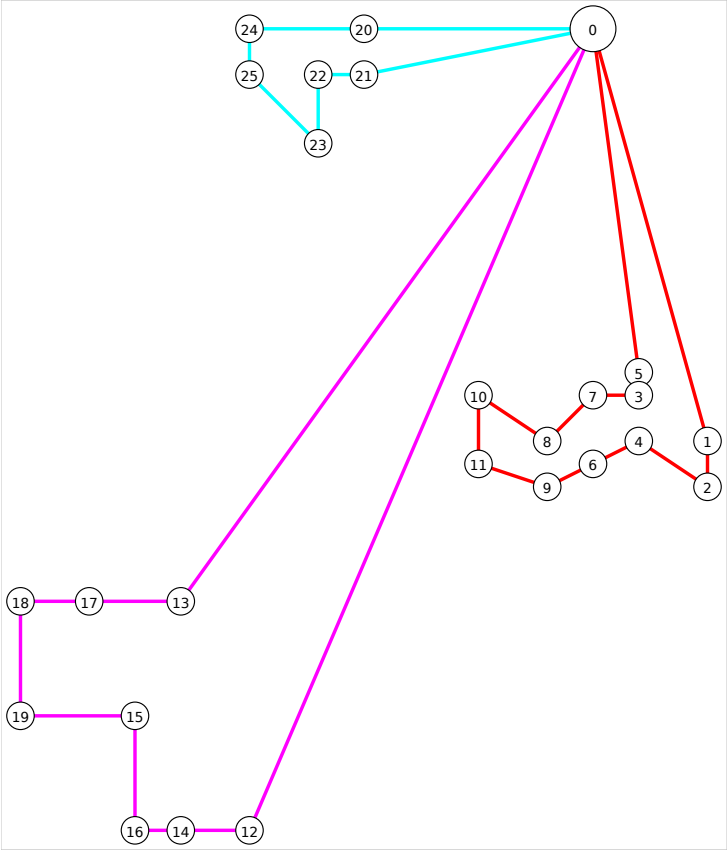

Figure 5: Solution of "C101.25.txt"

9 Bin Packing (binPacking1)

Problem: The bin packing problem (BPP) is a classical model from the operations research (see also binpack[14] for a MIP formulation of the problem). Different items of given weights must be packed into a number of bins of given capacity. How many bins are needed to pack all items?

Modeling Steps

Given a set of items $i \in I$ with a weight w_i and a set of bins $k \in K$ with capacity $Capa$ (we suppose that $w_i \leq Capa$ for all $i \in I$), that is all items can be packed in a bin. We need at most $maxBins$ bins :

$$maxBins = \min(|I|, \sum_i w_i/capa)$$

Hence $K = \{1, \ldots, maxBins\}$.

Let's enumerate all items from 1 to $|I|$. Then we are asked to partition these numbers into maximally $|K|$ subsets (but as few as possible), such that the cumulated weight of the items in a subset do not exceed the capacity $Capa$. The item numbers form a permutation, that must be partitioned into subsets. We can formulated this by introducing a variable $x_{k,i}$ with $k \in K$ and $i \in I$. For example: $x_{1,1} = 16$ means that the first item in bin 1 is the item with number 16, $x_{1,2} = 19$ means that the second item in bin 1 is the item with number 19, etc. $x_{i,k} = 0$ means that there is no k-th item in bin i. Note that the order within the subset does not matter.

The model then can be formulated as follows:

let	$bW_k = \sum_{i \mid x_{k,i} \neq 0} w_{x_{k,i}}$	forall $k \in K$
let	$bU_k = \operatorname{count}_i \ x_{k,i} > 0$	forall $k \in K$
s.t.	$bW_k <= Capa$	forall $k \in K$
min	$\sum_k bU_k$	
set partition	$x_{k,i} \in \{1 \ldots \lvert I \rvert\}$	forall $k \in K, i \in I$

bW_k is the (unknown) weight of a bin k. bU_k is true (1) if the bin k is not empty (it counts the items in a bin k and if there is at least 1 then it is true). The unique constraint makes sure that the capacity of a bin is not exceeded, and the objective function minimizes the number of bins used.

Listing 18: The Main Model implemented in LPL [2]

```
model bin "Bin Packing";
set i,j  "items";
    k    "bins";
parameter Capa; weight{i};
alldiff x{k,i} 'set partition';
expression bWeight{k}: sum{i|x} weight[x];
expression bUsed{k}: count{i} x > 0;
constraint bCapa{k}: bWeight <= Capa;
minimize nrBins: sum{k} bUsed;
```

[14] https://lpl.matmod.ch/lpl/Solver.jsp?name=/binpack

```
end
```

Solution: With the data instance "t60_00.txt" the optimal solution is 21, which is immediately found by LocalSolver. LPL's data model is :

Listing 19: The Data Model

```
model data;
  string parameter File:='bp-instances/t60_00.txt';
  parameter n; m;
  Read(File,n);
  i:=1..n;
  Read('%1;1',Capa);
  Read{i}('%1;2',weight);
  parameter minBins:=Ceil(sum{i} weight/Capa);
            maxBins:= min(#i, 2*minBins);;
  k:=1..maxBins;
end
```

The output is as follows:

```
Bin weight:  979 | Items:  11 37 42
Bin weight:  968 | Items:   7 48 56
Bin weight:  925 | Items:   1 10
Bin weight:  968 | Items:   8 47 55
Bin weight:  893 | Items:  15 51 53
Bin weight:  946 | Items:  26 33 34
Bin weight:  864 | Items:   6 12
Bin weight:  998 | Items:  14 29 36
Bin weight:  967 | Items:  13 38 41
Bin weight:  989 | Items:   2 50 57
Bin weight:  976 | Items:  18 30 31
Bin weight:  961 | Items:  17 28 39
Bin weight:  982 | Items:   9 40 49
Bin weight:  941 | Items:  16 32 45
Bin weight:  973 | Items:  21 24 52
Bin weight:  827 | Items:   4 23
Bin weight:  998 | Items:   3 43 59
Bin weight:  995 | Items:   5 46 54
Bin weight:  980 | Items:  19 20 60
Bin weight:  871 | Items:  27 44 58
Bin weight:  999 | Items:  22 25 35
```

10 Capacitated Facility Location (capaFacilityLocation1)

Problem: The capacitated facility location problem (CFLP) tries to assign customer (demand) sites to distribution center (warehouses or facilities) at the minimal cost.

Modeling Steps

Let $i \in I$ be a set of demand sites served from a warehouse (facility) and let $f, g \in F$ be a potential number of facility locations. Each facility has a capacity c_f and an opening price o_f. The demand at each site is d_i and the price to allocate site i to facility f is $A_{f,i}$. The set of sites must be partionend into subset, that is, we are looking for a partitioned permuation (without ordering), modeled as $x_{f,i}$.

1. Let C_k be the (unknown) number of sites assigned to facility f :

$$C_k = \underset{i}{\text{count}}\ x_{k,i}$$

2. The capacity constraint must be fullfilled, that is, the capacity of facility f should at least be the cumulated demand of all sites allocated to it :

$$\sum_{i|x_{f,i}} d_i <= c_f \quad \text{forall } f \in F$$

3. The cost at an (open) facility is the allocation prices of its sites plus the opening price :

$$Cost_f = \sum_{i|x_{f,i}} A_{f,i} + o_f \cdot (C_f > 0) \quad \text{forall } f \in F$$

4. We like to minimize the overall costs:

$$\min \sum_f Cost_f$$

Listing 20: The Main Model implemented in LPL [2]

```
model capaFac "Capacitated Facility Location";
  set f,g  "set of facilities/warehouses";
  set i    "set of demand sites served from a warehouse";
  parameter nbMaxFacilities; nbSites;
  parameter capacity{f}; openningPrice{f};
  parameter demand{i};
  parameter allocationPrice{f,i};
  alldiff x{f,i} 'set partition';
  expression cnt{f}: count{i} x;
  constraint Dem{f}: sum{i|x} demand[i] <= capacity[f];
  expression cost{f}:
     sum{i|x} allocationPrice[f,i] + openningPrice[f]*(cnt>0);
  minimize totalCost: sum{f} cost;
end
```

Solution: The data are read in the data model :

Listing 21: The Data Model

```
model data;
  string File:='cfl-instances/cap61';
  Read(File, nbMaxFacilities, nbSites);
  f:=1..nbMaxFacilities;
  i:=1..nbSites;
  Read{f}('@%1',capacity, openningPrice);
  Read('@%1',{i} demand);
  Read('@#%1',{f,i} allocationPrice);
end
```

The optimal solution for the instance "cap61" is 932615.75. Running the output model and the results are:

```
Warehouse  1 delivers to sites:   3  6 14 24 30 31 33
Warehouse  2 delivers to sites:   7
Warehouse  3 delivers to sites:   8 34
Warehouse  4 delivers to sites:  11 17 19 21 42
Warehouse  6 delivers to sites:   4 13 37 38 40 49
Warehouse  7 delivers to sites:  15 20 22 44 48
Warehouse  8 delivers to sites:   1  5  9 10 16 39 43 46 47
Warehouse  9 delivers to sites:  18
Warehouse 11 delivers to sites:  12 23 26 28 29 32 41
Warehouse 12 delivers to sites:   2 25 35 36 50
Warehouse 13 delivers to sites:  27 45
```

Listing 22: The Output Model

```
model output;
  Write{f|exist{i}x}('Warehouse␣%2d␣delivers␣to␣sites:␣%3d\n',f,{i|x}
      x);
end
```

11 Assembly Line Balancing (assemblyLineBalancing1)

Problem: In the assembly line balancing problem (SALBP) a set of tasks with a given processing time must by partitioned into subsets. Each subset is assigned to a "station" (work place) and the cycle time (cumulated processing time) on a station is limited. Furthermore, the tasks are partially ordered by precedence relations. Assign the tasks to the stations such that the cycle time is not exceeded and the precedence order holds. The number of stations must be minimized.
The problem and the following Figure is from Armin Scholl. In the Figure, there are 5 stations each containing 2 tasks that do not exceed the cycle time of 66.

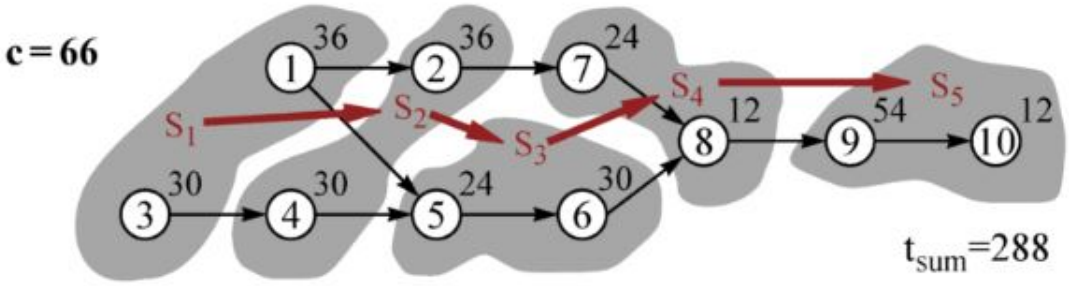

Figure 6: A Solution to the "salbp.txt" Instance

Modeling Steps

Given a set of tasks $i, j \in I$ and a set of stations $s \in S$. The cycle time for each station is cT and the processing time of each task is pT_i. Furthermore, let $p_{i,j} = 1$ if task j is a successor of task i (precedence relation).
A set of tasks must be partioned into subsets: this can be modeled using a partitioned permutation variable $x_{s,i}$, where the order inside a partition is not important.

1. Let tS_i be the station that contains the task i. This can be modeled by the function Find() (see LPL manual).
$$tS_i = \text{Find}(x_{s,i}, i)$$

2. A constraint is required that limits the cycle time within a station, that is, the cumulated processing time of the task in the station cannot exceed the cycle time. :
$$\sum_{i|x_{s,i}} pT_i \leq cT \quad \text{forall } s \in S$$

3. If task j is a successor of task i (i precedes j), then task j cannot be in a previous station then i :
$$tS_i \leq tS_j \quad \text{forall } i, j \in I,\ succ_{i,j} \neq 0$$

4. Finally, the number of station is to be minimized:
$$\min \sum_{s \in S} (\underset{i \in I}{\text{count}}\ x_{s,i} > 0)$$

Listing 23: The Main Model implemented in LPL [2]

```
model assembly "Assembly Line Balancing";
  set i,j "set of tasks";
  set s   "stations";
  set prec{i,j} "j is succ of i";
  parameter nbTasks; maxNbStations; cycleTime;
  parameter processingTime{i}; succ{i,j};
  alldiff station{s,i} 'set␣partition';
  constraint timeInStation{s}:
    sum{i|station} processingTime <= cycleTime;
  expression taskStation{i}: Find(station,i);
  constraint Prec{i,j|succ>0}: taskStation[i] <= taskStation[j];
  minimize nbUsedStations: sum{s} (count{i} station[s] > 0);
end
```

Solution: The data are read with the data model:

Listing 24: The Data Model

```
  model data;
    string parameter File:='n20_1.alb'; --opt=3
    --string parameter File:='alb-instances/n20_99.alb'; --opt=12
    --string parameter File:='alb-instances/salbp.txt'; --opt=5
    Read(File&',%1:<number␣of␣tasks>',nbTasks);
    i := 1..nbTasks;
    s := 1..nbTasks;
    Read('%1:<cycle␣time>',cycleTime);
    Read{i}('%1:<task␣times>:<precedence␣relations>',i,processingTime);
    Read{i,j}('%1:<precedence␣relations>:<end>',i,j,prec);
    succ{i,j}:=if(prec,j-1);
  end
```

The solution of instance "n20_1.alb" is 3, three stations are needed. Running the output model and the results are:

```
On station  1: tasks:   1  2  3  4  5  7  8
On station  2: tasks:   6  9 11 12 14 15 19
On station  3: tasks:  10 13 16 17 18 20
```

Listing 25: The Output Model

```
  model output;
    Write{s|exist{j}station}('On␣station␣%2d:␣tasks:␣%3d\n',s,{j|
       station} station);
  end
```

12 K-Means Clustering (kmeans1)

Problem: Given a sample of observations along some dimensions, the goal is to partition these observations into k clusters. Various different mathematical models can also be found in my book Case Studies I.

Modeling Steps

There is a set of observations $i \in I$ which are to partitioned into a number of clusters $k \in K$. Each observation consists of a set of attributes $d \in D$, called the dimension, and data of an observation i and the attribute d is defined as $c_{d,i}$. Furthermore, for each observation, an initial membership to a cluster is given as iC_i.

The variable is a partitioned permutation without ordering $x_{k,i}$, since all the items i must be partitioned into the k subsets.

1. Let S_k be the (unknown) size of the k cluster :

$$S_k = \operatorname*{count}_i x_{k,i}$$

2. The centroid of a cluster k on attribute d is given by:

$$CE_{k,d} = \text{if}(S_k > 0, \sum_{i|x_{k,i}} d_i)/S_k \quad \text{forall } k \in K, d \in D$$

3. The quadratic deviation of the attribute d in cluster k is :

$$SQ_{k,d} = \sum_{i|x_{k,i}} (d_i - CE_{k,d})^2 \quad \text{forall } k \in K, d \in D$$

4. Hence, the variance of cluster k can be expressed as :

$$VA_k = \sum_d squares_{k,d} \quad \text{forall } k \in K$$

5. The goal is to minimize the overall variances :

$$\min \sum_k VA_k$$

Listing 26: The Main Model implemented in LPL [2]

```
model kmeans "K-Means Clustering";
  set i "obervations";
      d "dimensions";
      k "Clusters";
  parameter coor{d,i};
  string initCluster{i};
  alldiff clusters{k,i} [1..#i] 'set partition';
  expression
    size{k}:= count{i} clusters;
    centroid{k,d}:= if(size>0,sum{i|clusters} coor) / size;
    squares{k,d}:=sum{i|clusters}(coor-centroid)^2;
    variances{k}:=sum{d} squares;
  minimize obj: sum{k}(variances);
end
```

Solution: The data instances are read by the data model:

Listing 27: The Data Model

```
model data;
  --string parameter File:='km-instances/glass.dat';
  string parameter File:='km-instances/ruspini.dat';
  --string parameter File:='km-instances/iris.dat';
  --string parameter File:='km-instances/segment.dat';
  parameter nbObservations; nbDimensions; K:=2;
  Read(File,nbObservations,nbDimensions);
  i:=1..nbObservations;
  d:=1..nbDimensions;
  Read{i}('%1;1', {d} coor, initCluster);
  set kk; {i} Addm(kk,initCluster);
  k:=1..#kk;    //k:=1..K;
end
```

The solution of problems with $|D| = 2$ (two attributes) can be displayed in a diagram, see Figure 7, generated by the output model :

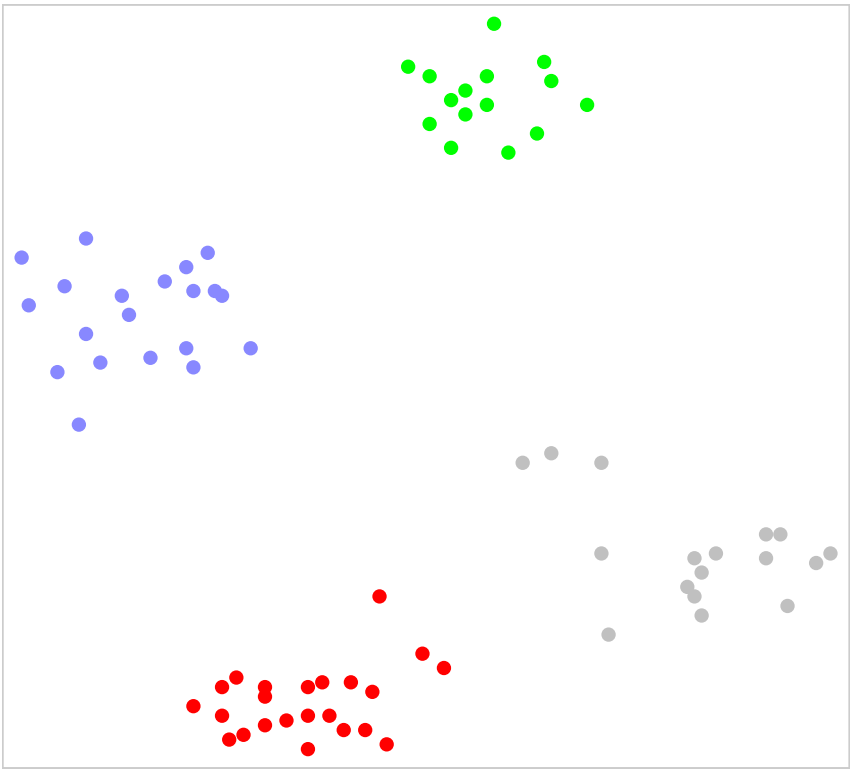

Figure 7: Solution of the instance "ruspini.dat"

Listing 28: The Output Model

```
model output friend data;
  Write{k,i|clusters}('%3d %3d %s\n', k,clusters,initCluster[clusters
     ]);
  if File='km-instances/ruspini.dat' then
    Draw.Scale(3,2);
    {k,i|clusters} Draw.Circle(coor[1,clusters],coor[2,clusters],1,k
       +1);
  end;
end
```

13 Split Delivery Vehicle Routing (sdvrp1)

Problem: The Split Delivery Vehicle Routing Problem is a variant of the capacitated vehicle routing problem for which a customer can be visited by more then one vehicle: The delivery is split between at least two trucks.
All data instances used here are from Belenguer et at., SDVRP LIB.

Modeling Steps

Let $i, j \in I = \{1, \ldots, n\}$ be a set of customer locations (n is the number of locations excluding the depot and let $k \in K = \{1, \ldots, m\}$ be a set of trucks. Let the capacity of each truck be CA, and the demand quantity to deliver to a customer i be dem_i. Furthermore, the distance between two customer i and j is $d_{i,j}$. Finally, the distance from the warehouse – the depot from where the trucks start – to each customer i is dw_i.

We are looking for a partitioned permutation for which repetition in the subset can occur, that is, the same costumer ID can be in several subset. This is modeled in LPL by adding the Quote attribute `'cover'` to the permutation variable $x_{k,i}$. An additional variable $q_{k,i}$ is introduced for the quantity delivered to customer i with truck k.

1. Let C_k be the (unknown) size of the k subtour (without depot):

$$C_k = \underset{i}{\text{count}}\ x_{k,i}$$

2. Whether a truck is used or not is defined by uT_k and the (unknown) number of trucks T can be formulated as :

$$uT_k = C_k > 0 \quad \text{forall } k \in K$$

3. Let rD_k the route distance traveled by the truck k: it is the cumulated distance between the customers plus the distance from the depot to the first customer and the distance to the depot from the last customer:

$$rD_k = \sum_{i \in 2..C_k} d_{x_{k,i-1}, x_{k,i}} + \text{if}(tU_k, dw_{xk,1} + dw_{x_{k,C_k}}) \quad \text{forall } k \in K$$

4. The capacity constraint of a truck k must hold: the cumulated quantity that a truck delivers to its customers must not exceed the truck's capacity:

$$\sum_{i|x_{k,i}} q_{k,i} \leq CA \quad \text{forall } k \in K$$

5. A second constraint requires that the demand for each customer must be fullfilled, that ia, The sum of the delivery of all trucks to customer i must be at least its demand :

$$\sum_{k|x_{k,i}} q_{k,i} \geq dem_i \quad \text{forall } i \in I$$

6. The objective is to minimize the overall distances traveled by the trucks :

$$\min \sum_k rD_k$$

Listing 29: The Main Model implemented in LPL [2]

```
model sdvrp "Split Delivery Vehicle Routing";
  set i,j  "customers";
      k    "trucks";
  parameter demands{i};
    X{i}; Y{i}; wX; wY; --coord
    distance{i,j} "distances between customers";
    distW{i}  "distance from warehouse";
    CA "truck capacity";
  alldiff x{k,i} 'cover' "customerSequences";
  variable qua{k,i} [0..demands] "quantity";
  expression cnt{k}:=count{i} x;
  expression trucksUsed{k}: cnt > 0;
  constraint Capa{k}: sum{i|x} qua <= CA;
  //constraint Capa{k}: sum{i in 1..cnt} qua[k,x] <= CA;
  expression routeDistances{k}: sum{i in 2..cnt}
    distance[x[k,i-1],x[k,i]] + if(trucksUsed,distW[x[k,1]]+distW[x[k,
        cnt]]);
  constraint quantityServed{i}: sum{k|x} qua >= demands;
  minimize totalDistance: sum{k} routeDistances;
end
```

Solution: The data model is

Listing 30: The Data Model

```
  model data;
    string parameter File:='sd-instances/S51D1.sd';     //opt=458
    --string parameter File:='sd-instances/S51D3.sd';   //opt=972
    --string parameter File:='sd-instances/S101D5.sd'; //opt=2915
    parameter nbCustomers;
    Read(File, nbCustomers, CA);
    i:=1..nbCustomers;
    k:=1..nbCustomers;
    Read('%1;1',{i} demands);
    Read('%1;2', wX,wY);
    Read{i}('%1;3', X,Y);
    distance{i,j}:=Floor(Sqrt((X[i]-X[j])^2+(Y[i]-Y[j])^2)+0.5);
    distW{i}:=Floor(Sqrt((X[i]-wX)^2+(Y[i]-wY)^2)+0.5);
  end
```

After 20secs, LocalSolver find the solution 463 of the "S51D1.sd" instance shown in Figure 8 which is close to the optimum of 458.
Figure 9 shows the solution of the instance "S51D3.sd" after 20secs using LocalSolver with the objective value of 981, which is close to the optimum of 972. It shows a solution where the customer 14 is visited by two trucks. Its demand of 74 is split into 16 and 58 units. The graph in Figure 9 is generated by the following output model :

Listing 31: The Output Model

```
  model output;
    Draw.Scale(7,7);
    set k0{k}:=exist{i}x;
    {k0,i in 1..cnt-1} Draw.Line(X[x],Y[x],X[x[k,i+1]],Y[x[k,i+1]],k
        %6+3,3);
    {k0} (Draw.Line(wX,wY,X[x[k,1]],Y[x[k,1]],k%6+3,3),
          Draw.Line(wX,wY,X[x[k,cnt]],Y[x[k,cnt]],k%6+3,3));
    {k0,i in 1..cnt} Draw.Circle(x&'',X[x],Y[x],2,1,0);
```

```
    Draw.Circle('0',wX,wY,2,1,0);
end
```

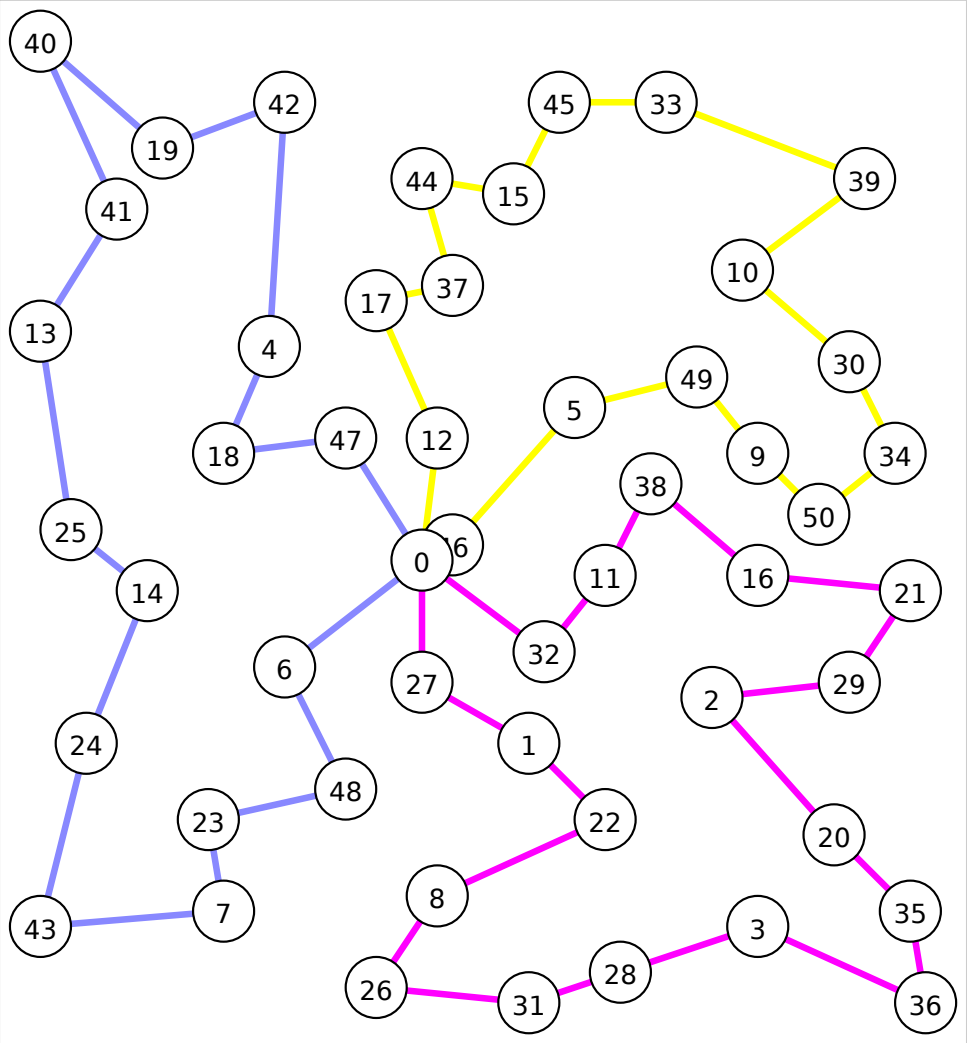

Figure 8: Solution to instance “S51D1.sd”

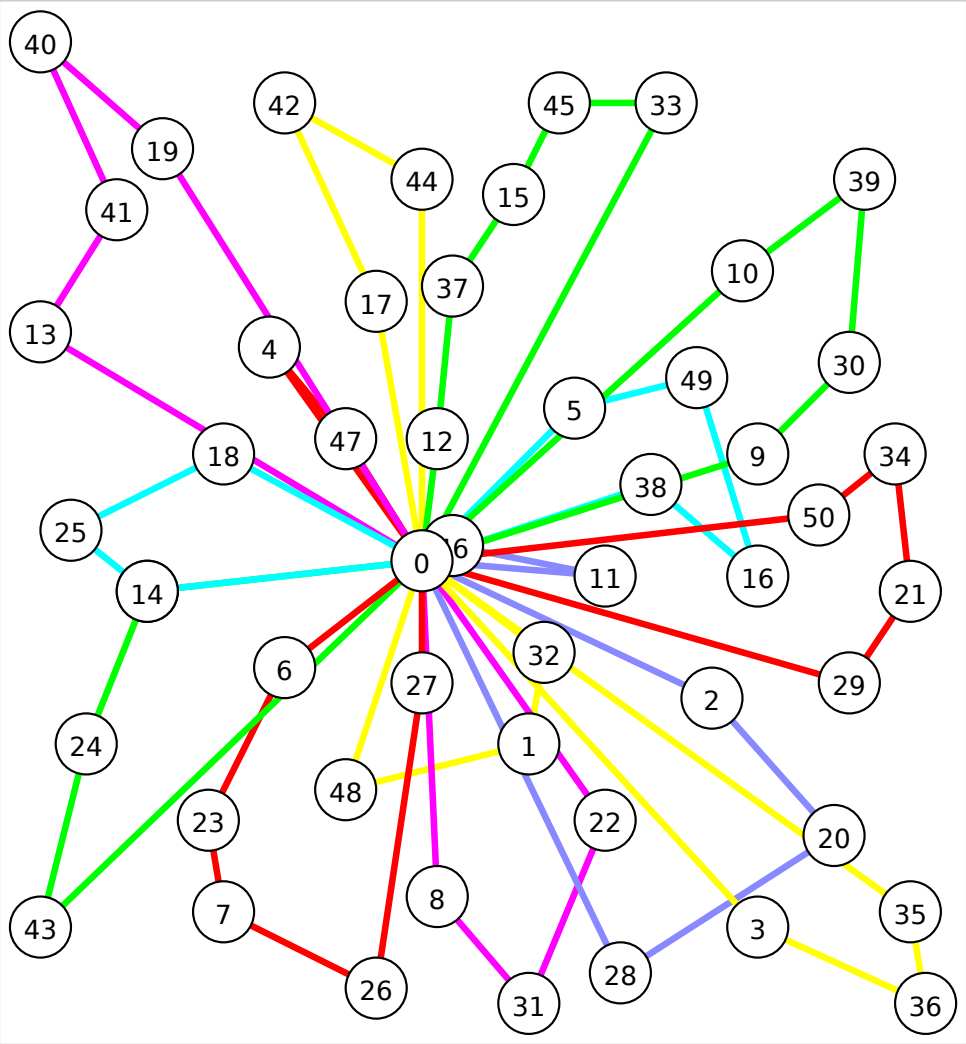

Figure 9: Solution to instance “S51D3.sd”

14 Capacitated Arc Routing (capacitatedArcRouting1)

Problem: The capacitated arc routing problem (CARP) asks for traversing a given number of required arcs (or edges) in a graph. Several vehicles start at the same depot and each of them serve a (disjoint) subset of arcs with a given demand such that the capacity of the vehicle is not exceeded. The total distances traversed by the vehicles should be minimal. To fullfill the tasks also non-required arcs can be traversed. The data instances are from DIMACS.

Modeling Steps

Given a graph $G(V, A \cup A')$ with a given set of nodes $(i, j \in V)$ and a required arcs $(k, l \in A)$ and not required arcs $(nk \in A')$. Note that the required arcs are ordered from 1 to $|A|$ and if k_0 is an arc with starting node i and ending node j, then the next arc k_0+1 in the list is the arc with starting node j and ending node i. Furthermore, let $t \in T$ be a set of vehicles. The following data are given: the capacity of a vehicle is CA; for each required edge, the required edge cost is rC_k; the required edge demand is rD_k; the distance between the edges k and l is $dE_{k,l}$ (calculated by the shortest path algorithm); the distance from the depot is df_k; and finally the distance to depot is dd_k. The variables is a (routing) disjoint permutation over the required edges: $x_{t,k}$.

1. The first constraint says that an required arc must be traversed exactly once, either from a node i to j or from the node j to i. Hence, each edge (containing a forward and backward arc $2k-1$ and $2k$ where $k \in |A|/2$ – which are ordered sequentially as mentioned above) must be contained in a vehicle tour, and only in one:

   $$\text{Contains}(x, 2 * k0 - 1) + \text{Contains}(x, 2 * k0) = 1 \quad \text{forall } k0 \in \{1, \ldots, |A|/2\}$$

   The function contains return 1 if the arc k is part of a tour t (note: $x_{t,k}$ is a disjoint permutation).

2. Let C_t be the (unknown) number of required arcs in the tour t:

   $$C_t = \operatorname{count}_k x_{t,k} \quad \text{forall } t \in T$$

3. A second constraint is the capacity constraint: The demands on the required arcs of a tour t must not exceed the capacity of the vehicle t:

   $$\sum_{i \in \{1,\ldots,C_t\}} rD_{x_{t,i}} \leq CA \quad \text{forall } t \in T$$

4. Let the routing distance of a vehicle t be rD_t. This is the sum of the traversed required arcs plus the distances between the required arcs plus (if the tour is not empty) the distance from the depot to the first edge and the distance from the last edge to the depot:

   $$rD_t = \sum_{i \in 2\ldots C_t} (rE_{x_{t,i}} + dE_{x_{t,i-1},x_{t,i}} + \text{if}(C_t > 0, rC_{x_{t,1}} + df_{x_{t,1}} + dd_{x_{t,C_t}}) \quad \text{forall } t \in T$$

5. Finally we want to minimize the total distance:

$$\min \sum_t rD_t$$

Listing 32: The Main Model implemented in LPL [2]

```
model CapArcR "Capacitated Arc Routing";
  set i,j   "nodes";
      k,l   "required edges";
      t     "trucks";
  parameter CA; reqEdgesCosts{k}; reqEdgesDemands{k};
    distBetweenEdges{k,l}; distFromDepot{k}; distToDepot{k};
  alldiff x{t,k} 'disjoint' "edge sequences";
  constraint A{k in 1..Round(#k/2)}:
    Contains(x,2*k-1) + Contains(x,2*k) = 1;
  expression C{t}: count{k} x;
  constraint B{t}: sum{i in 1..C} reqEdgesDemands[x[t,i]] <= CA;
  expression rD{t}: sum{i in 2..C} (reqEdgesCosts[x[t,i]] +
    distBetweenEdges[x[t,i-1],x[t,i]]) +
    if(C>0, reqEdgesCosts[x[t,1]] + distFromDepot[x[t,1]] + distToDepot
       [x[t,C]]);
  minimize totDist: sum{t} rD;
end
```

Solution: The data instance file "egl-e1-A.dat" is chosen. Line 3 contains the number of nodes, line 4 the number of required arcs, line 5 the number of not required arcs, line 6 the number of vehicles, line 7 the capacity. The next list from line 11 on contains the required arcs with starting node, ending node, cost (distance), and demand. The next list contains the not required arcs with starting node, ending node, cost (distance). Finally, the last line is the node number for the depot.

Listing 33: The Data Model

```
model data;
  string parameter File:='egl-e1-A.dat';
  parameter nbNodes; nbRequiredEdges; nbNotRequiredEdges; nbTrucks;
    fN{k}; sN{k}; cost{k}; demand{k}; //endpoint of required nodes
    fNN{nk}; sNN{nk}; costN{nk}; //endpoints of not required nodes
    depotNode;
  Read(File&',%1;2',nbNodes=3);
  Read('%1;3',nbRequiredEdges=3);
  Read('%1;4',nbNotRequiredEdges=3);
  Read('%1;5',nbTrucks=3);
  Read('%1;6',CA=3);
  t:=1..nbTrucks;
  i:=1..nbNodes;
  k:=1..2*nbRequiredEdges;
  set nk:=1..nbNotRequiredEdges  "set of not required edges";
  set ri{i}          "required nodes"; --not used in the model
  Read{k0 in 1..#k/2}
    (',%1;10',fN[k0]=2, sN[k0]=4, cost[k0]=6, demand[k0]=8);
  string parameter dummy;
  Read('@,%1',dummy);
  Read{nk}('@,%1',fNN=2, sNN=4, costN=6);
  Read('@,%1',depotNode=3);
```

```
  parameter Nei{i,j} "i and j are neighbours nodes";
    reqEdgesOrigin{k}; reqEdgesDestination{k};
  {k} (Nei[fN,sN]:=cost, Nei[sN,fN]:=cost);
  {nk} (Nei[fNN,sNN]:=costN, Nei[sNN,fNN]:=costN);
  {k0 in 1..#k/2}
    (reqEdgesCosts[2*k0-1]:=cost[k0], reqEdgesCosts[2*k0]:=cost[k0],
     reqEdgesDemands[2*k0-1]:=demand[k0], reqEdgesDemands[2*k0]:=
        demand[k0],
     reqEdgesOrigin[2*k0-1]:=fN[k0], reqEdgesOrigin[2*k0]:=sN[k0],
     reqEdgesDestination[2*k0-1]:=sN[k0], reqEdgesDestination[2*k0]:=
        fN[k0],
     ri[fN[k0]]:=1, ri[sN[k0]]:=1);
  calcDistances;
  model calcDistances;
    parameter x{i,j}; reqDist{i,j}; o; d;
    {i,j|i<>j} ( o:=i, d:=j,
      reqDist[i,j] := Graph.SPath(Nei,x,o,d));
    {k,l} (distBetweenEdges := reqDist[reqEdgesDestination[k],
       reqEdgesOrigin[l]]);
    {k} (distToDepot := reqDist[reqEdgesDestination[k],depotNode]);
    {k} (distFromDepot := reqDist[depotNode,reqEdgesOrigin[k]]);
  end
end
```

The data model first reads all the data from the file. It then assigns the data to the required edges costs and the required edges demands. All the distances are calculated in a submodel `calcDistances`, which is based on LPL's internal shortest path method. The output is generated by the output model:

Listing 34: The Output Model

```
model output friend data;
  Write('Total␣distance:␣%d\n', totDist);
  Write{t}('Sequence␣of␣truck␣%d␣:␣%s\n',t,
    Format{k in 1..C[t]}('(%d,%d)␣',reqEdgesOrigin[x],
       reqEdgesDestination[x]));
end
```

It is as follows:

```
Sequence of truck 1 : (4,5) (9,10) (12,16) (16,13) (13,14)
                      (17,15) (15,18) (18,19) (54,52) (52,50)
                      (50,49) (47,46) (45,44) (59,69)
Sequence of truck 2 : (58,57) (57,42) (43,44) (44,59) (69,4)
Sequence of truck 3 : (1,2) (59,58) (60,62) (62,66) (68,66)
                      (62,63) (63,65) (60,61) (60,58) (58,69)
Sequence of truck 4 : (2,4) (44,46) (47,48) (47,49) (49,51)
                      (51,21) (21,22) (75,22) (21,19) (19,20)
                      (20,76) (12,11) (11,59) (2,3)
Sequence of truck 5 : (75,23) (23,31) (31,32) (32,34) (33,32)
                      (32,35) (35,41) (55,56)
```

The solution for truck 1 is as follows: From the depot go to node 4 (using the shortest path), then traverse the required arcs $(4,5)$, then go to node 9 (using the shortest path), etc.

15 Further Models (sent to LocalSolver)

This part of the paper contains additional models from the LocalSolver Example Tour which are implemented in LPL. They also can be sent directly to LocalSolver from LPL. The comment `/*$C*/` at the beginning of each model code is a hack and shortcut that the model should be treated as a "PERM" problem an be sent to LocalSolver.

15.1 The 0-1 Knapsack Problem (knapsack1)

Problem: The problem has been explained elsewhere (see knapsack[15]).

Modeling Steps

This model is a 0-1 integer model, it can also be sent to a MIP solver like Cplex or Gurobi. By default it is sent to the LocalSolver.

Listing 35: The Main Model implemented in LPL [2]

```
model knapsack "The 0-1 Knapsack Problem";
  /*$C*/
  --SetSolver(LocalSolver);
  --SetSolver(cplexLSol);
  set i,items;
  parameter weights{i}; prices{i}; knapsackBound;
  binary variable x{i};
  constraint A: sum{i} weights*x <= knapsackBound;
  maximize Value: sum{i} prices*x;
  Writep(Value);
end
```

Solution: To find near optimal solutions of even large knapsack probelms takes no time in LocalSolver. The data set contains larger instances :

Listing 36: The Data Model

```
model data;
  string parameter File:='kn-instances/toy.in';
  --string parameter File:='kn-instances/kp_10000_1.in';
  parameter nbItems;
  Read(File,nbItems);
  i:=1..nbItems;
  Read('@%1', {i} weights);
  Read('@%1', {i} prices);
  Read('@%1', knapsackBound);
end
```

15.2 Facility Location Problem (facilityLocation1)

Problem: In the *Facility Location Problem* a subset S (called facilities) of a set of locations I is selected such that the overall transportation costs is minimized. Models of several

[15] https://lpl.matmod.ch/lpl/Solver.jsp?name=/knapsack

location problems have been implemented and described in my book Case Studies II, see, for example the models lcover[16] or pcenter[17].

The data instances are from OR-Library.

Modeling Steps

Given a set of locations $i, j \in I$ and a distance matrix $w_{i,j}$ between the locations. Furthermore, the number of facility should be fixed at p with $1 \leq p \leq |I|$. Let $wM = \max_{i,j}\ w_{i,j}$ be the largest distance between two locations. A binary variable x_i is used for each location i, which 1 if the location is a facility (distribution center).

The model then can be formulated as follows:

$$\begin{array}{lll} \text{let} & C_{i,j} = \text{if}(x_j, w_{i,j}, 2 \cdot wM) & \text{forall } i \in I \\ \text{let} & Cm_i = \min\limits_j\ C_{i,j} & \text{forall } i \in I \\ \text{s.t.} & \sum\limits_i x_i \leq p & \\ \min & \sum\limits_i Cm_i & \end{array}$$

Let $C_{i,j}$ be the transportation costs between location i and j, and let Cm_i the smallest cost from location i to any other location j. The sum of these costs must be minimized, while the number of facility should not be larger than p (the unique constraint).

Listing 37: The Main Model implemented in LPL [2]

```
model facilityLocation "Facility Location Problem";
  /*$C*/
  set i,j;
  parameter n; e; p; wmax:=0;
    w{i,j};
  binary variable x{i};
  constraint A: sum{i} x[i] <= p;
  expression
    costs{i,j}: if(x[j],w[i,j] , 2*wmax);
    cost{i}: min{j} costs[i,j];
  minimize totalCost: sum{i} cost[i];
end
```

Solution: A instance with 100 location and maximally 5 facilities is solved in no time by LocalSolver.

15.3 The Weighted Max-Cut Problem (maxcut1)

Problem: *The max-cut problem* is the problem to find a partrition of a graph into two parts such that the number of edges separating them (the cut-set) is as large as possible. In the *weighted max-cut problem* each edge has a weight and the problem is to find a weighted maximal number of edges (not to be confounded with the min-cut=max-flow problem, which is a polynomial problem – the max-cut is NP complete). For the max-flow problem see maxflow0[18]. The data instances here are from bigmaclib.

[16] https://lpl.matmod.ch/lpl/Solver.jsp?name=/lcover
[17] https://lpl.matmod.ch/lpl/Solver.jsp?name=/pcenter
[18] https://lpl.matmod.ch/lpl/Solver.jsp?name=/maxflow0

Modeling Steps

Let $G = (V, E)$ be a graph, with a set $i \in V$ of nodes and a set $j \in E$ of edges. Each edge has a weight w_i. Furthermore, for each edge let o_j be the origin node of edge j, and let d_j be the destination node of the edge j.

Define a binary variable x_i for each node i, which is 1 if and only if the node i is in a partition and 0 if it is in the other partition

1. An edge j is in the separation set (cut-set) if it has an extremity (origin or destination) in one partition and the other extremity in the other. Hence, we define IC_j to be true (1) if j is in the cut-set, that is, if the binary value of x_{o_j} (the origin of j) is different from x_{d_j} (the destination of j (note that a Boolean expression returns 0 or 1 for false and true) :

$$IC_j = (x_{o_j} \neq x_{d_j}) \quad \text{forall } j \in E$$

2. The goal is to maximize the weighted cut-set :

$$\max \sum_{j \in E} w_j \cdot IC_j$$

Listing 38: The Main Model implemented in LPL [2]

```
model maxcut "The Weighted Max-Cut Problem";
  /*$C*/
  set i "nodes"; j "arcs";
  parameter n; m;
    origin{j}; dest{j}; w{j};
  binary variable x{i};
  expression incut{j}: x[origin[j]] <> x[dest[j]];
  maximize cutWeight: sum{j} w[j] * incut[j];
  Write('cutWeight: %d',cutWeight);
  Write{i|x}('%3d', i);
end
```

Solution: With the data model reading the instance "ising2.5-200_5555", a graph with 200 nodes and 19900 edges, LocalSolver finds a solution of 6'071'097 after 100secs (the optimum is 6'294'701).

Listing 39: The Data Model

```
model data;
  string parameter File:='mc-instances/g05_60.0';
  --string parameter File:='mc-instances/ising2.5-200_5555';
  Read(File,n,m);
  i:=1..n;   j:=1..m;
  Read{j}('%1;1',origin,dest,w);
end
```

15.4 A larger 2000x1000 general IP (ip2000a)

Problem: This is a general IP with 2000 variables and 1000 linear constraints. Its general formulation is as follows.

$$\begin{array}{ll} \max & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

The data are generated randomly and the fill-factor of the matrix $\mathbf{A}$ is about 2%.

Listing 40: The Complete Model implemented in LPL [2]

```
model Ip2000 "A larger 2000x1000 general IP";
  /*$C*/
  set  m :=  [1..1000];    n :=  [1..2000];
  parameter
    A{m,n} := if(Rnd(0,1)<0.02 , Rnd(0,60));
    c{n}   := if(Rnd(0,1)<0.87 , Rnd(0,9));
    b{m}   := if(Rnd(0,1)<0.87 , Rnd(10,70000));
  integer variable  x{n};
  constraint R{m}:  sum{n} A*x <= b;
  maximize Obj:  sum{n}  c*x;
  Write('Objective␣Value␣=␣%7.2f␣\n', Obj);
  Write{n|x}('␣␣␣␣x%-4s␣=␣%5d\n', n,x);
end
```

Solution: Gurobi solves this problem in 3secs to optimality (opt=77454). LocalSolver finds an feasible solution of 77451 after 9 secs, very close to the optimum! Interestingly, LocalSolver returns also a gap of 0.05.% after 9 secs, and it proves optimality after one minute! This is great.

15.5 The Branin Function (branin1)

Problem: The Branin Function is defined as

$$f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos x_1 + s$$

For more information on the Branin Function see the Internet.

Listing 41: The Complete Model implemented in LPL [2]

```
model branin "The Branin Function";
  /*$C*/  --SetSolver(LocalSolver);
  parameter PI := 3.14159265359;
    a := 1;
    b := 5.1/(4*PI^2);
    c := 5/PI;
    r := 6;
    s := 10;
    t := 1/(8*PI);
  variable
    x1 [-5..10];
    x2 [0..15];
  minimize f: a*(x2 - b*x1^2 + c*x1 - r)^2 + s*(1-t)*Cos(x1) + s;
```

```
  Write('f=%4.3f␣,␣x1=%4.3f␣,␣x2=%4.3f\n', f, x1,x2);
end
```

15.6 Curve Fitting (cFitting1)

Problem: The curve fitting problem consists in finding the optimal parameters for a defined function $f()$ to best map a set of inputs to a set of outputs.

As an example, assume that the function $f()$ has the form:

$$f(x) = a\sin(b - x) + cx^2 + d$$

// minimize square error between prediction and outputs

Listing 42: The Complete Model implemented in LPL [2]

```
model curveFitting "Curve Fitting";
  /*$C*/
  set i "a set of observations";
  parameter inputs{i};  outputs{i};
  variable a [-100..100];  b [-100..100];
           c [-100..100];  d [-100..100];
  expression
    predictions{i}: a*Sin(b-inputs) + c*inputs^2 + d;
    errors{i}: predictions - outputs;
  minimize squareError: sum{i} errors^2;
  model data;
    string parameter File:='cu-instances/observations.in';
    parameter n  "number of observations";
    Read(File, n);
    i:=1..n;
    Read{i}('%1;1',inputs,outputs);
  end
  model output;
    Write('a=%4.3f␣,␣b=%4.3f␣,␣c=%4.3f␣,␣d=%4.3f\n', a,b,c,d);
    parameter I{i}:=i;
    Draw.Scale(1,1);
    Draw.XY(I,predictions,outputs,1);
  end
end
```

Solution: The solution is : $a = -0.848, b = -8.011, c = 0.002, d = 36.355$. The data of the function are plotted in Figure 10.

15.7 Optimal Bucket by Limited Material (oBucket1)

Problem: Design a bucket (a truncated cone) with an given upper bound of surface (quantity of material to be used) that maximizes its volume. The problem is from Datagenetics.

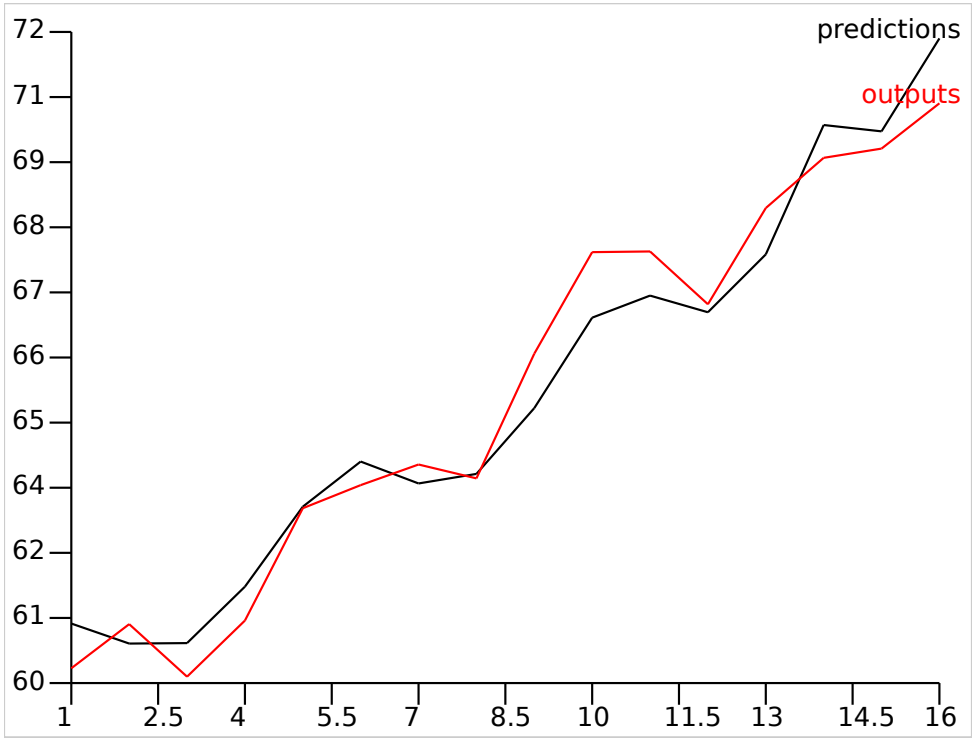

Figure 10: Curve Fitting Plotted

Modeling Steps

The truncated cone has an unknown radius of R on top and an unknown radius r at the bottom, the height h is also unknown. The surface is to be limited to π. The volume V and the surface S are given by the folllowing formula :

$$V = \frac{\pi h}{3}\left(R^2 + Rr + r^2\right)$$

$$S = \pi r^2 + \pi(R + r)\sqrt{(R - r)^2 + h^2}$$

The model is to maximize V subject to a given upper bound of its surface:

$$\begin{array}{ll} \max & V \\ \text{s.t.} & S \leq \pi \end{array}$$

Listing 43: The Complete Model implemented in LPL [2]

```
model bucket "Optimal Bucket by Limited Material";
  /*$C*/
  parameter PI := 3.14159265359;
  variable R [0..1];  r [0..1];  h [0..1];
  constraint surface: PI*r^2 + PI*(R+r)*Sqrt((R-r)^2 + h^2) <= PI;
  maximize volume: PI*h/3*(R^2 + R*r + r^2);
  Write('surface=%4.3f␣,␣volume=%4.3f\n', surface+PI, volume);
  Write('r=%4.3f␣,␣R=%4.3f␣,␣h=%4.3f\n', r,R,h);
end
```

Solution: The solution to this non-linear model is $S = 3.142$, $V = 0.687$, $(r, R, h) = 0.404, 0.735 h = 0.656$.

15.8 Find Smallest Circle (sCircle1)

Problem: Find the smallest circle that contains a number of given points in the plane (see also model circle[19]. The problem and more efficient versions has been explained in my book Puzzlebook.

Modeling Steps

Given a set of points $i \in I$ with the coordinates (X_i, Y_i). Find the radius r and the center (x, y) of a circle with the smallest radius that includes all points. The model is as foolows:

$$\begin{array}{ll} \min & r \\ \text{where} & r = \sqrt{\max_{i \in I} ((x - X_i)^2 + (y - Y_i)^2)} \end{array}$$

Listing 44: The Complete Model implemented in LPL [2]

```
model smallestCircle "Find Smallest Circle";
  /*$C*/ --SetSolver(LocalSolver);
  parameter n; coorX{i}; coorY{i};
  Read('sc-instances/30points.txt',n);
  set i:=1..n "Points";
  Read{i}('%1;1', coorX, coorY);
  parameter
    minX := min{i} coorX;
    minY := min{i} coorY;
    maxX := max{i} coorX;
    maxY := max{i} coorY;
  variable x [minX..maxX]; y [minY..maxY];
  minimize r: Sqrt(max{i}((x-coorX)^2 + (y-coorY)^2));
  Draw.Scale(20,20);
  Draw.Circle(x,y,r,1,0);
  {i} Draw.Circle(coorX,coorY,.1);
end
```

Solution: The output of this model is shown in Figure 11.

15.9 (socialGolfer1)

Problem: A number of golfers, each of whom plays golf once a week, are partioned into groups of given size. One needs to schedule the groups in such a way that each player meets a maximum of other players in his groups (maximal socialization). Another formulation is found in model golfers[20]. The model is explained in my (see Puzzle Book.)

Modeling Steps

Let m be a number of groups, n the group size and k a number of weeks. Define a set of weeks as $w \in W = \{1, \ldots, k\}$, a set of groups as $g \in G = \{1, \ldots, n\}$, and a set of players as $i, j \in I = \{1, \ldots, n \cdot m\}$. The binary variable $x_{w,i,g}$ is 1 if player i is in group g at week w.

[19] https://lpl.matmod.ch/lpl/Solver.jsp?name=/circle
[20] https://lpl.matmod.ch/lpl/Solver.jsp?name=/golfers

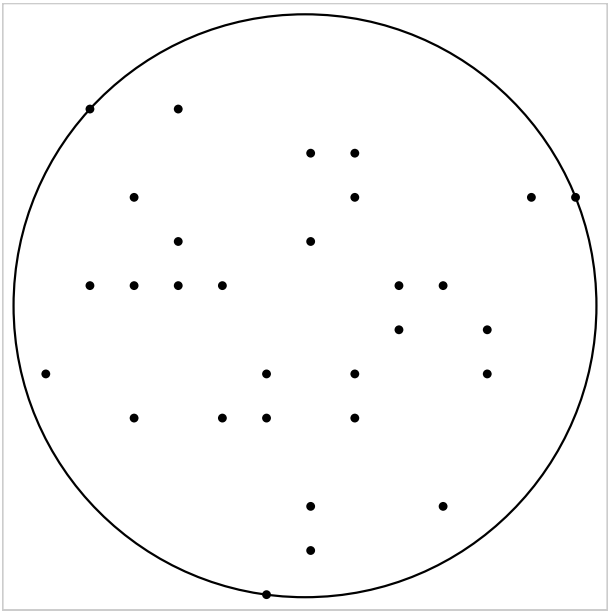

Figure 11: Solution to "30points.txt"

1. Constraint one is: each week w, each golfer i is assigned to exactly one group g:

$$\sum_{g} x_{w,g,i} = 1 \quad \text{forall } w \in W, i \in I$$

2. Constraint two is: each week w, each group g contains exactly n golfers:

$$\sum_{i} x_{w,g,i} = n \quad \text{forall } w \in W, g \in G$$

3. Let $M_{w,g,i,j}$ with $j > i$ be true (1) if golfers i and j meet in group g on week w:

$$M_{w,g,i} = x_{w,g,i} \ \wedge \ x_{w,g,j} \quad \text{forall } w \in W, g \in G, i, j \in I, j > i$$

4. Let $nM_{i,j}$ be the number of meetings between golfers i and j

$$nM_{i,j} = \sum_{w,g} M_{w,g,i,j} \quad \text{forall } i, j \in I, j > i$$

5. Let $maxM_{i,j}$ be the maximum of meetings (minus one) between golfers i and j:

$$maxM_{i,j} = \max(nM_{i,j} - 1, 0) \quad \text{forall } i, j \in I, j > i$$

6. The maximal meeting over all meetings must be minimal:

$$\min \sum_{i,j|j>i} maxM_{i,j}$$

Listing 45: The Complete Model implemented in LPL [2]

```
model socialGolver;
  /*$C*/
  parameter nbGroups; groupSize; nbWeeks;
  Read('c_4_3_3.in', nbGroups, groupSize, nbWeeks);
  --Read('instances/c_10_8_4.in', nbGroups, groupSize, nbWeeks);
  parameter nbGolfers := nbGroups * groupSize;
  set w:=1..nbWeeks;
  set g:=1..nbGroups;
  set i,j:=1..nbGolfers;
  binary variable x{w,g,i};
  constraint A{w,i}: sum{g} x[w,g,i] = 1;
  constraint B{w,g}: sum{i} x[w,g,i] = groupSize;
  expression meetings{w,g,i,j|j>i}: x[w,g,i] and x[w,g,j];
    nbMeetings{i,j|j>i}: sum{w,g} meetings[w,g,i,j];
    redundantMeetings{i,j|j>i}: Max(nbMeetings[i,j] -1, 0);
  minimize obj: sum{i,j|j>i} redundantMeetings[i,j];
  Write('obj: %d\n', obj);
  Write('The group of players is as follows:\n\n');
  Write('          %-14s\n',{w}('Week '&w));
  Write{g}('Group %s %-14s\n',g,
    {w}Strreplace(Format('(%s)',{i|x}(i&',')),',)',')'));
end
```

LPL generates the following model code for the LocalSolver

```
/ LSP file (LocalSolver.com) :: automatically generated by LPL
use io;
function input() {
  groupSize = 3;
  WW = 3;
  GG = 4;
  II = 12;
}

function model() {
  x[w in 0..WW-1][g in 0..GG-1][i in 0..II-1] <- bool();
  for[w in 0..WW-1][i in 0..II-1]
    constraint  sum [g in 0..GG-1] (x[w][g][i]) ==1;
  for[w in 0..WW-1][g in 0..GG-1]
    constraint  sum [i in 0..II-1] (x[w][g][i]) ==groupSize;
  meetings[w in 0..WW-1][g in 0..GG-1][i in 0..II-1][j in 0..II-1 : j>i
     ] <-  and (x[w][g][i],x[w][g][j]);
  nbMeetings[i in 0..II-1][j in 0..II-1 : j>i] <-  sum [w in 0..WW-1][g
      in 0..GG-1] (meetings[w][g][i][j]) ;
  redundantMeetings[i in 0..II-1][j in 0..II-1 : j>i] <- max (
     nbMeetings[i][j]-1,0);
  obj <-  sum [i in 0..II-1][j in 0..II-1 : j>i] (redundantMeetings[i][
     j]) ;
  minimize obj;
}

function param() {
  if (lsTimeLimit == nil) lsTimeLimit = 20;
  lsVerbosity = 2;
}
```

```
function output() {
  if (solFileName == nil) return;
  local solFile = io.openWrite(solFileName);
  solFile.println(getSolutionStatus());
  solFile.println(obj.value);
  solFile.print("x:␣");
  for[w in 0..WW-1][g in 0..GG-1][i in 0..II-1] solFile.print(x[w][g][i
      ].value,",");
  solFile.println();
}
```

Solution: For the "c_4_3_3.in" instance the output is

```
obj: 0
The group of players is as follows:

          Week 1         Week 2         Week 3
Group 1  (2,3,9)        (5,7,9)        (2,10,11)
Group 2  (1,4,7)        (3,4,11)       (1,8,9)
Group 3  (5,8,10)       (1,10,12)      (3,6,7)
Group 4  (6,11,12)      (2,6,8)        (4,5,12)
```

16 Conclusion

The goal of this paper was to show that permutation problems have important applications, and their formulations is straightforward, close to a mathematical notation. The important ingredients are *permutation variables* and the ability to use *these variables as indexes*. The modeling language LPL contains all these elements, and so permutation problems are easy to formulate in LPL. But the proof that this works is that LPL can send these models directly to a very powerful commercial solver: LocalSolver. The interface is still experimental, but already very promising.

Another goal was to explore the power of LocalSolver. The solver is the leading solver if it comes to permutation problems. However, it is also an universal solver for linear, integer, and non-linear problems. As a first test, a 2000×1000 IP problem above was a special highlight, which was solve near optimally within a few seconds.

References

[1] MatMod. Homepage for Learning Mathematical Modeling : https://matmod.ch .

[2] Hürlimann T. Reference Manual for the LPL Modeling Language, most recent version. https://matmod.ch/lpl/doc/manual.pdf.