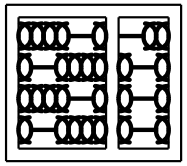

UNIVERSIDADE DE CAMPINAS - UNICAMP
INSTITUTO DE COMPUTAÇÃO - IC

Examples with GUROBI

Knapsack - `knapsack.cpp`

TSP and Lemon Graph Library - `ex_tsp_gurobi.cpp`

Flávio Keidi Miyazawa

Campinas, 2012

Knapsack Problem

Problem KNAPSACK: Given items $S = \{1, \ldots, n\}$ with value v_i and size s_i, $i = 1, \ldots, n$, and Capacity B, find $S' \subseteq S$ such that $\sum_{i \in S'} s_i \leq B$ and $\sum_{i \in S'} v_i$ is maximized.

Define binary variables x_i, $i \in S$ such that
$x_i = 1$ if element i belongs to solution
$x_i = 0$ otherwise.

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i \in [n]} v_i x_i \\ \text{such that} & \left\{ \begin{array}{rcll} \displaystyle\sum_{i \in [n]} s_i x_i & \leq & B & \\ x_i & \in & \{0,1\} & \forall i \in [n] \end{array} \right. \end{array}$$

where $[n] = \{1, \ldots, n\}$.

```
int main()
{ srand48(1);
  int time_limit=10,n=5; // time_limit in seconds; n=number
  double Capacity=100.0;
  vector<double> size(n);      vector<double> value(n);
  vector<GRBVar> x(n);  // Gurobi variables and environment
  GRBEnv env = GRBEnv();
  GRBModel model = GRBModel(env);
  GRBLinExpr expr;
  model.set(GRB_StringAttr_ModelName,"Knapsack Example"); /
  model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE); // maxim
  for (int i=0;i<n;i++) {
    size[i]=(double) (drand48()*Capacity);
    value[i]=(double) (drand48()*Capacity);
    x[i] = model.addVar(0.0, 1.0, value[i],GRB_BINARY,"");
    expr += size[i]*x[i];}
  model.update(); // run update to use model inserted varia
  model.addConstr(expr <= Capacity); // sum of item sizes c
```

```
  try {
    model.update(); // Process any pending model modificati
    if (time_limit >= 0) model.getEnv().set(GRB_DoubleParam
    model.optimize(); // Optimize the model
    cout << "\n\nCapacity = "<< Capacity <<
      "  Solution value = " <<   model.get(GRB_DoubleAttr_O
    for (int i=0;i<n;i++) {
      cout << "i="<< i+1 << " size=" << size[i] << " value=
" x[" << i << "] = " << x[i].get(GRB_DoubleAttr_X) << "\n";
    return 0;
  }catch (...) {printf("Exception...\n");exit(1);}
}
```

Traveling Salesman Problem (TSP)

TSP: Given graph $G = (V, E)$, cost c_e for each $e \in E$, find a hamiltonian circuit $C \subseteq E$ that minimizes $\sum_{e \in C} c_e$.

Use binary variables x_e for each $e \in E$ such that: $x_e = 1$ if and only if e belongs to solution

$$\text{minimize} \quad \sum_{e \in E} c_e x_e$$

$$\text{sujeito a} \quad \begin{cases} \displaystyle\sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\ \displaystyle\sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, \quad S \neq \emptyset \\ & \text{(subtour elimination)} \\ x_e \in \{0,1\} & \forall e \in E \end{cases}$$

First constraint: solution has two incident edges for each node.

Second constraint: Solution is connected (there are no small cicles).

```
#include <gurobi_c++.h>
#include <float.h>
#include <math.h>
#include "readgraph.h"
#include "viewgraph.h"
#include "adjacencymatrix.h"
#include <lemon/list_graph.h>
#include <lemon/gomory_hu.h>
#define BC_EPS 0.001
#define BC_INF 1000000000000.0
```

```
typedef ListGraph::Node Node;
typedef ListGraph::NodeIt NodeIt;
typedef ListGraph::Edge Edge;
typedef ListGraph::EdgeMap<double> EdgeWeight;
typedef ListGraph::EdgeMap<int> EdgeIndex;
typedef ListGraph::NodeMap<string> NodeName;
typedef ListGraph::NodeMap<double> NodePos;
typedef ListGraph::EdgeMap<string> EdgeName;
typedef ListGraph::NodeMap<bool> CutMap;
typedef Preflow<ListGraph, EdgeWeight> PFType;
typedef ListGraph::NodeMap<int> NodeColor;
typedef ListGraph::EdgeMap<int> EdgeColor;
```

```
class TSP_Data {
public:
  TSP_Data(ListGraph &graph,NodeName &nodename,NodePos &pos
   NodePos &posy,EdgeWeight &eweight);
  ListGraph &g;
  int NNodes,NEdges;
  int max_perturb2opt_it; // maximum number of iterations f
  NodeName &vname;
  EdgeName ename;
  NodeColor vcolor;
  EdgeColor ecolor;
  EdgeWeight &weight;
  NodePos &posx;
  NodePos &posy;
  AdjacencyMatrix AdjMat; // adjacency matrix
  vector<Node> BestCircuit; // vector containing the best c
  double BestCircuitValue;
};
```

```
TSP_Data::TSP_Data(ListGraph &graph,NodeName &nodename,Node
   NodePos &posicaoy,EdgeWeight &eweight):
  g(graph),
  vname(nodename),
  ename(graph),
  vcolor(graph),
  ecolor(graph),
  weight(eweight),
  posx(posicaox),
  posy(posicaoy),
  AdjMat(graph,eweight,BC_INF), //
  BestCircuit(countEdges(graph))
{
  NNodes=countNodes(this->g);
  NEdges=countEdges(this->g);
  BestCircuitValue = DBL_MAX;
  max_perturb2opt_it = 3000; // default value
}
```

```
// Return true if variable x is fractional (within a certai
bool IsFrac(double x) {
  double f;
  f = ceil(x)-x;
  if (f<BC_EPS) return(false);
  if (f>1.0-BC_EPS) return(false);
  return(true);
}
```

```
bool Heuristic_2_OPT(AdjacencyMatrix &A,vector<Node> &Circuit,double &BestCircuitValue, int &NNode
{
  double CurrentWeight=0.0,Remove,Insert;
  bool globalimproved,improved;
  vector<Node> CircuitAux(NNodesCircuit);
  int i,j,k,l;
  for (int i=0;i<NNodesCircuit-1;i++)
    CurrentWeight += A.Cost(Circuit[i],Circuit[i+1]);
  CurrentWeight += A.Cost(Circuit[NNodesCircuit-1],Circuit[0]);
  i=0;    globalimproved = false;
  do {
    int i1,i2,j1,j2;
    improved=false;
    i1 = i;
    do {
      // one edge is (i1 , i1+1)
      i2 = (i1+1)%NNodesCircuit;
      for (j=2;j<NNodesCircuit-1;j++) {
        j1 = (i1+j)%NNodesCircuit;
        j2 = (j1+1)%NNodesCircuit;
        // try to replace edges (i1,i2) and (j1,j2) with
        // edges                (i1,j1) and (i2,j2)

        ... // see the file ex_tsp_gurobi.cpp

      }
      i1=(i1+1)%NNodesCircuit;
    } while (i1!=i);
    i = (i+1)%NNodesCircuit;
  }while (improved);
  if (globalimproved) cout << "[Heuristic: 2OPT] New Solution of value " << BestCircuitValue << "\
  return(globalimproved);
}
```

```
// This routine must be called when the vector x (indexed on the edges) is integer.
// The contained circuit is transformed into a circuit represented by a sequence of nodes.
bool Update_Circuit(TSP_Data &tsp, ListGraph::EdgeMap<GRBVar>& x)
{
  ListGraph::NodeMap<Node> Adj1(tsp.g), Adj2(tsp.g);
  double CircuitValue,f;
  Node FirstNode=INVALID,u,ant;
  int n,i;
  // Adj1[v] and Adj2[v] have the two adjacent nodes of v in the circuit
  for (NodeIt v(tsp.g); v != INVALID; ++v) { Adj1[v]=INVALID; Adj2[v]=INVALID; }
  CircuitValue = 0.0;
  for (EdgeIt e(tsp.g); e != INVALID; ++e) {
    f = x[e].get(GRB_DoubleAttr_X);   if (IsFrac(f)) return(false);
    if (f > 1.0-BC_EPS) { Node u,v;   // if the edge is in the solution
      u = (tsp.g.u(e)); v = (tsp.g.v(e));   // then, obtain the edge nodes u and v
      CircuitValue += tsp.weight[e];
      if (Adj1[u]==INVALID) Adj1[u] = v; else Adj2[u] = v; // v is adjacent to u
      if (Adj1[v]==INVALID) Adj1[v] = u; else Adj2[v] = u; // and u is adj. to v
    }
  }
  if (CircuitValue > tsp.BestCircuitValue-BC_EPS) return(false);
  // find one node of the circuit
  for (NodeIt v(tsp.g); v != INVALID; ++v)
if (Adj1[v]!=INVALID) {FirstNode = v; break;}
  tsp.BestCircuitValue = CircuitValue; // Here, the new circuit is better
  i = 1;                               // than the previous best circuit
  tsp.BestCircuit[0] = FirstNode;
  u = Adj2[FirstNode];          ant = FirstNode;
  while (u!=FirstNode) {
    tsp.BestCircuit[i] = u;     i++;
    if (Adj1[u]==ant) {ant=u; u=Adj2[u];} else {ant=u; u=Adj1[u];}
  }
  Heuristic_2_OPT(tsp.AdjMat,tsp.BestCircuit,tsp.BestCircuitValue,tsp.NNodes);
  return(true);
}
```

```
class subtourelim: public GRBCallback
{ TSP_Data &tsp;
  ListGraph::EdgeMap<GRBVar>& x;
  double (GRBCallback::*solution_value)(GRBVar);
public:
  subtourelim(TSP_Data &tsp, ListGraph::EdgeMap<GRBVar>& x) : tsp(tsp),x(x)
  {    }
protected:
  void callback()
  {
    if (where==GRB_CB_MIPSOL){solution_value = &GRBCallback::getSolution;}
    else if (where==GRB_CB_MIPNODE && getIntInfo(GRB_CB_MIPNODE_STATUS)==GRB_OPTIMAL)
      {solution_value = &GRBCallback::getNodeRel;}
    else return;
    try {
      typedef ListGraph::EdgeMap<double> capacityType;
      capacityType capacity(tsp.g);
      for (ListGraph::EdgeIt e(tsp.g); e!=INVALID;++e) capacity[e]=(this->*solution_value)(x[e]);
      GomoryHu<ListGraph, capacityType> ght(tsp.g, capacity);     ght.run();
      // Gomory-Hu tree given as rooted directed tree. Each node has an arc
      // that points to its father. The root node has father -1. Each arc
      // represents a cut. So, if an arc has weight less than 2 than we found
      // a violated cut and in this case, we insert the corresponding constraint.
      for (NodeIt u(tsp.g); u != INVALID; ++u) {
        GRBLinExpr expr=0;            double vcut;
        if ((tsp.g).id(ght.predNode(u))==-1) continue;
        vcut =  ght.predValue(u);
        if (vcut > 2.0 - BC_EPS) continue;
        for(GomoryHu<ListGraph,EdgeWeight>::MinCutEdgeIt a(ght,u,ght.predNode(u));a!=INVALID;++a)
            expr += x[a];      addLazy( expr >= 2 );
      }
    } catch (...) {cout << "Error during callback" << endl;}
  }
};
```

```
void ChangeNode(Node &a,Node &b)
{ Node c;  c = a; a = b; b = c; }

// This routine starts with some solution (current best or a random generated
// solution) and iteratively perturb changing some nodes and applying 2OPT.
bool TSP_Perturb2OPT(TSP_Data &tsp)
{
  ListGraph *g;  int nchanges=1,i,j;  g = &tsp.g;
  vector<Node> Circuit(tsp.NNodes), BestCircuit(tsp.NNodes);
  double BestCircuitValue;
  // Start with a initial solution (if there is no solution, generate any sequence)
  if (tsp.BestCircuitValue < DBL_MAX) { // there is an existing solution
    BestCircuitValue = tsp.BestCircuitValue;
    for (int i=0; i<tsp.NNodes; i++) BestCircuit[i] = tsp.BestCircuit[i];
  } else {
    .... // generage an initial solution. See file ex_tsp_gurobi.cpp
  }
  for (int it=0;it<tsp.max_perturb2opt_it;it++) {
    if (!(it%100)) printf("[Heuristic: Perturbation+2OPT] it = %d (of %d)\n",it+1,tsp.max_perturb2
    for (int k=0;k<tsp.NNodes;k++) Circuit[k] = BestCircuit[k];
    for (int nc=0;nc<nchanges;nc++) {
      i = (int) (drand48()*tsp.NNodes);  // get two random nodes and exchange
      j = (int) (drand48()*tsp.NNodes);  // their positions
      if (i!=j) ChangeNode(Circuit[i],Circuit[j]);
    }
    Heuristic_2_OPT(tsp.AdjMat,Circuit,BestCircuitValue,tsp.NNodes);
    if (BestCircuitValue < tsp.BestCircuitValue) { //update the best circuit used
      tsp.BestCircuitValue = BestCircuitValue;     // by the heuristic
      for (int i=0;i<tsp.NNodes;i++) {
        BestCircuit[i] = Circuit[i];
        tsp.BestCircuit[i] = Circuit[i];}
    }
  }
  return(true);
}
```

```
void ViewTspCircuit(TSP_Data &tsp)
{
  ListGraph h;
  ListGraph::NodeMap<string> h_vname(h);  // node names
  ListGraph::NodeMap<Node> h_g2h(tsp.g);  // maps a node of g to a node of h
  ListGraph::NodeMap<double> h_posx(h);
  ListGraph::NodeMap<double> h_posy(h);
  ListGraph::NodeMap<int> vcolor(h);   // color of the vertices
  ListGraph::EdgeMap<int> acolor(h);  // color of edges
  ListGraph::EdgeMap<string> aname(h);  // name of edges
  for (ListGraph::NodeIt v(tsp.g); v!=INVALID; ++v) {
    Node hv;
    hv = h.addNode();
    h_g2h[v] = hv;
    h_posx[hv] = tsp.posx[v];
    h_posy[hv] = tsp.posy[v];
    h_vname[hv] = tsp.vname[v];
    vcolor[hv] = BLUE;
  }
  for (int i=0;i<tsp.NNodes;i++) {
    ListGraph::Node u,v;
    ListGraph::Edge a;
    u = tsp.BestCircuit[i];
    v = tsp.BestCircuit[(i+1) % tsp.NNodes];
    a = h.addEdge(h_g2h[u] , h_g2h[v]);
    aname[a] = "";
    acolor[a] = BLUE;
  }
  ViewGraph(h,VIEW_NEATO,h_vname,aname,h_posx,h_posy,vcolor,acolor);
}
```

```
int main(int argc, char *argv[])
{
  int time_limit;
  char name[1000];
  double cutoff;
  ListGraph g;
  EdgeWeight lpvar(g);
  EdgeWeight weight(g);
  NodeName vname(g);
  ListGraph::NodeMap<double> posx(g),posy(g);
  srand48(1);
  time_limit = 3600; // solution must be obtained within ti
  if (argc!=2) {cout<<"Usage: "<< argv[0]<<" <graph_filenam
  else if (!FileExists(argv[1])) {cout<<"File "<<argv[1]<<"

  // Read the graph
  if (!ReadGraph(g,vname,weight,posx,posy,argv[1]))
    {cout<<"Error reading graph file "<<argv[1]<<"."<<endl;
```

```
TSP_Data tsp(g,vname,posx,posy,weight);
ListGraph::EdgeMap<GRBVar> x(g);
GRBEnv env = GRBEnv();    GRBModel model = GRBModel(env);
model.getEnv().set(GRB_IntParam_DualReductions, 0); // Dual reductions must be disabled when usi
model.set(GRB_StringAttr_ModelName, "Undirected TSP with GUROBI"); // name to the problem
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE); // is a minimization problem

// Add one binary variable for each edge and also sets its cost in the objective function
for (ListGraph::EdgeIt e(g); e!=INVALID; ++e) {
  sprintf(name,"x_%s_%s",vname[g.u(e)].c_str(),vname[g.v(e)].c_str());
  x[e] = model.addVar(0.0, 1.0, weight[e],GRB_BINARY,name);
}
model.update(); // run update to use model inserted variables

// Add degree constraint for each node (sum of solution edges incident to a node is 2)
for (ListGraph::NodeIt v(g); v!=INVALID; ++v) {
  GRBLinExpr expr;
  for (ListGraph::IncEdgeIt e(g,v); e!=INVALID; ++e) expr += x[e];
  model.addConstr(expr == 2 );
}
```

```
  try {
    model.update(); // Process any pending model modifications.
    if (time_limit >= 0) model.getEnv().set(GRB_DoubleParam_TimeLimit,time_limit);

    subtourelim cb = subtourelim(tsp , x);
    model.setCallback(&cb);

    tsp.max_perturb2opt_it = 10000; // number of iterations used in heuristic TSP_Perturb2OPT
    TSP_Perturb2OPT(tsp);
    if (tsp.BestCircuitValue < DBL_MAX) cutoff = tsp.BestCircuitValue-BC_EPS; //
    // optimum value for gr_a280=2586.77, gr_xqf131=566.422, gr_drilling198=15808.652
    if (cutoff > 0) model.getEnv().set(GRB_DoubleParam_Cutoff, cutoff );

    model.optimize();

    double soma=0.0;
    for (ListGraph::EdgeIt e(g); e!=INVALID; ++e) {
      lpvar[e] = x[e].get(GRB_DoubleAttr_X);
      if (lpvar[e] > 0.99) soma += tsp.AdjMat.Cost(e);
    }

    cout << "Solution cost = "<< soma << endl;
    Update_Circuit(tsp,x); // Update the circuit in x to tsp circuit variable (if better)
    ViewTspCircuit(tsp);

  }catch (...) {
    if (tsp.BestCircuitValue < DBL_MAX) {
      cout << "Heuristic obtained optimum solution"  << endl;
      ViewTspCircuit(tsp);
      return 0;
    }else {  cout << "Graph is infeasible"  << endl;     return 1;  }
  }
}
```