Mathematical Programming: Modelling and Applications

Leo Liberti

LIX, École Polytechnique

liberti@lix.polytechnique.fr

January 2018

Outline

Distance geometry problem

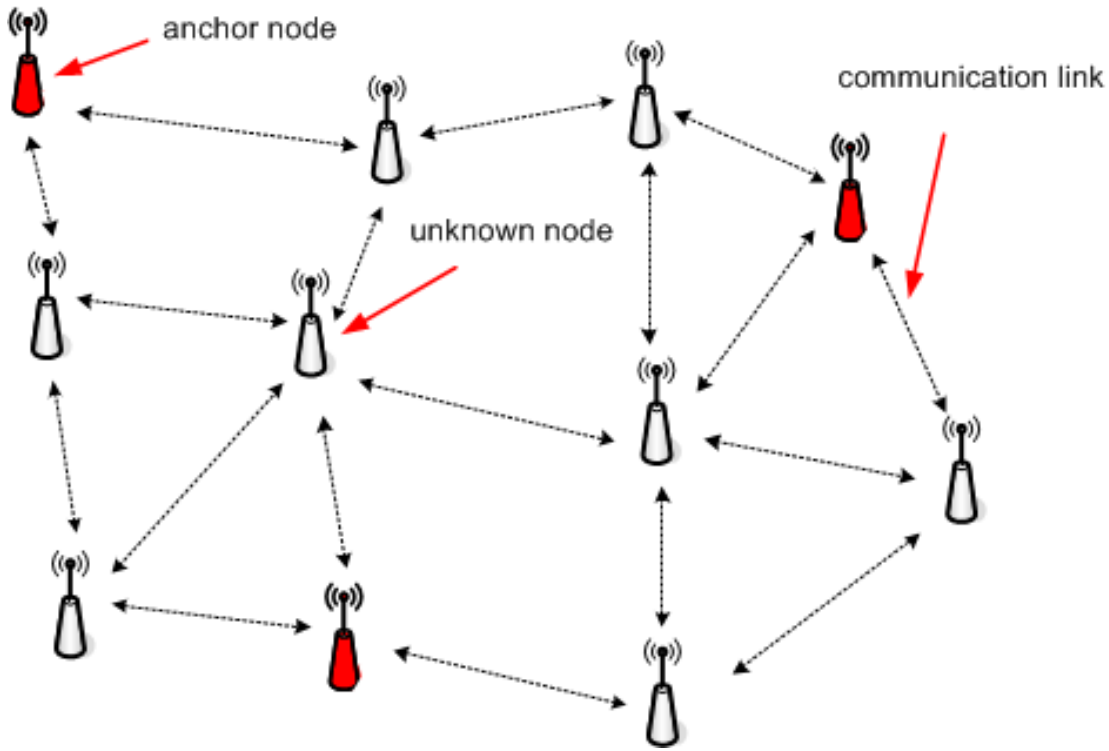
- DISTANCE GEOMETRY PROBLEM (DGP):

  Given a simple, connected, weighted graph $G = (V, E, d)$ and an integer $K > 0$, is there a realization $x : V \to \mathbb{R}^K$ such that:

$$\forall (u, v) \in E \quad ||x(u) - x(v)|| = d_{uv} \quad ?$$

- Note: a realization of a graph is a function, mapping each vertex to a vector of a Euclidean space.

Distance geometry problem: example



- Sensor network localization: for set of mobile devices in a wireless network (unknown positions) a solution of DGP gives a consistent position of the devices at time t.

First mathematical model

- Euclidean Distance Geometry Problem (EDGP):

  If we consider the the 2-norm (*Euclidean norm*), the constraint becomes:

$$\forall (u,v) \in E \quad \sqrt{\sum_{k \leq K} (x_{uk} - x_{vk})^2} = d_{uv}$$

  and if we square both sides:

$$\forall (u,v) \in E \quad \sum_{k \leq K} (x_{uk} - x_{vk})^2 = {d_{uv}}^2$$

First mathematical model

- Remark 1: because it consists of a set of constraints for which we need to find a solution, the DGP is a Constraint Programming (CP) problem. On the other hand, CP problems usually have integer, bounded variables, whereas the DGP has continuous unbounded ones. So it makes more sense to see it as a feasibility Nonlinear programming (NLP) problem, i.e. an NLP problem with zero objective function.
- Remark 2: anyhow, it is difficult to solve DGP as systems of nonlinear constraints. Usually, MP solvers are better at improving optimality than ensuring feasibility. A nice feature of DGP is that we can write them as a minimization of constraint errors.

First mathematical model

- Slack variables:

  In order to get a minimization problem, we could introduce slack variables and rewrite our constraints:

$$\forall (u,v) \in E \quad \sum_{k \leq K} (x_{uk} - x_{vk})^2 = {d_{uv}}^2 + slack_{uv}$$

  and minimize their sum, introducing in this way our objective:

$$\min \left(\sum_{(u,v) \in E} {slack_{uv}}^2 \right)$$

- **Formulate a mathematical model and solve it by AMPL + solvers**

Drawing Graphs in 2D: solvers

- **Baron** (The Optimization Firm): general nonlinear optimizer capable of solving nonconvex optimization problems to global optimality. Decision variables may be continuous, integer, or a mixture of the two.
- **Knitro** (Artelys): nonlinear solver that can deal with varied objective and constraint nonlinearities in continuous and integer variables. Special features: extensive use of shared-memory multi-core computing.
- **Snopt** (Stanford Systems Optimization Laboratory): widely used large-scale optimizer for difficult large-scale nonlinear problems.
- https://ampl.com/products/solvers/solvers-we-sell/

Drawing Graphs in 2D, preparation

- Download: dgpsystem.mod, dgp-preamble.mod, dgp-postprocess.run, dgp.run, data.zip, dgp-postprocess.run and also rlz2.plt, rlz3.plt.
- put everything in a single directory
- unzip data.zip
- download and install gnuplot

Drawing Graphs in 2D

- work on random graphs in 2D first
- ampl < dgp.run
- obtain some realizations (dgp_solution.tab)
- display them with gnuplot (gnuplot < rlz2.plt; on windows open gnuplot and from within gnuplot open file rlz2.plt)
- compare solutions of same formulation/instance for different solvers ("compare" = "plot and see whether solutions differ")
- hint: edit dgp.run to choose formulation, instance, solver

Drawing Graphs in 2D, alternative constraints

- remove the range constraints from the variables: any changes?
- uncomment the barycenter (zerocenter) constraint: changes with regard to when it was commented?

Drawing Graphs in 2D, alternative constraints

Barycenter (zerocenter) constraint makes all translated solutions infeasible

- conceive alternative constraints for the **same purpose**
- conceive constraints for making **rotations** infeasible
- conceive constraints for making **reflections** infeasible

Drawing Graphs in 2D, alternative formulations

- **propose other formulations of DGP and implement them using AMPL**
- choose an instance and to vary formulation/solvers: compare (as above, graphically)

Protein graphs

- move to the three protein graphs
- let Kdim := 3;
- plot solutions over varying formulations/solvers using *gnuplot* $<$ *rlz*3.*plt*, beginning with pept_gph.dat
- plot solutions over varying formulations/solvers using *gnuplot* $<$ *rlz*3.*plt*
- move on to tiny instance (this is a disconnected graph)
- Activate/deactivate the barycenter (zerocenter) constraint (i.e. comment/uncomment): what changes are there?

Multistart

- Multistart: run repeatedly a constructive method from multiple starting points
- Multistart methods, normally, have two phases that are repeated for a certain number of overall iterations:
  1. The first phase generates an initial feasible solution
  2. The second phase seeks to improve it.
- Each overall iteration (i.e. both phases 1 and 2) tries to produce an improved solution (that is typically a local optimum).
- The number of iteration is bounded by some kind of threshold (cpu time . . .)
- The best overall solution is the output of the algorithm.

Propose a multistart algorithm for DGP and implement it using AMPL

Multistart

- Remarks and hints

- Code your multistart algorithm in a .run file named dgp_ms.run
- Look at dgp.run, last lines: "*let solver := multistart*" and "*include "dgp_ms.run*"
- dgp_ms.run is on the same level as baron or knitro;
- Every execution starts from is dgp.run
- Use snopt or knitro to find your initial solution (phase 1 of the algorithm)
- Try to improve it (phase 2 of the algorithm)
- Repeat 1 and 2 from different starting points
- Limit somehow the number of iterations.
- Keep the best of the solutions produced by all iterations

Multistart - implementation: dgp_ms.opt

We could define some configurable parameters in a separate file named dgp_ms.opt, which we can include in our main file

```
### configurable parameters
# optimality tolerance
param ms_epsilon default 1e-3;
# time limit
param ms_maxcputime default 10; #900;
# original variable bounds -- initialize with original var bounds if != +/-M
param orig_xL{V,K} default M;
param orig_xU{V,K} default -M;

# solver for multistart
#option solver snopt;
option solver knitro;
```

Multistart - implementation: dgp_ms.run (1)

The main algorithm begins here...

```
## simple MultiStart for DGP
##   assuming realization is in var x{V,K} <= M, >= -M
option randseed 0;
option solver_msg 0;

include "dgp_ms.opt";

### nonconfigurable parameters

# termination
param ms_termination binary default 0;
# realization error wrt given distances
param ms_err;
# best solution so far
param ms_xstar{V,K};
# lowest realization error so far
param ms_errstar;
# CPU time
param ms_cputime default 0;
```

Multistart - implementation: dgp_ms.run (2)

```
# solve the problem locally
let{v in V, k in K} x[v,k] := Uniform(orig_xL[v,k], orig_xU[v,k]);
solve > nul;
# initialize first "best so far" point
let{v in V, k in K} ms_xstar[v,k] := x[v,k];
# compute the realization error
let ms_errstar :=
   (1/card(E))*sum{(u,v) in E}
      abs(sqrt(abs(sum{k in K} (x[u,k]-x[v,k])^2)) - c[u,v]);
printf "ms: starting solution err = %g\n", ms_errstar;

param ms_opt binary, default 0;
param ms_k integer, default 0;
```

Multistart - implementation: dgp_ms.run (3)

```
repeat while(ms_termination == 0) {

    # initialize iteration counter at each loop

    ...to be completed...

    # choose initial random point

    ...to be completed...

    # solve the problem locally

    ...to be completed...

    # compute the realization error

    ...to be completed...

    # check for global optimality
    if (ms_err < ms_epsilon) ...to be completed...

    # check termination condition

    ...to be completed...
}
```

Multistart - implementation (4)

...and finally...

```
    # save the solution into x

...to be completed...
```