Exam projects for INF580

Leo Liberti, LIX CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France

liberti@lix.polytechnique.fr

Notes

- None of the projects below are well-defined. Come talk to me (or write me an email) and we'll define it together.
- All of the projects below contain some theoretical and some implementation work.
- Be aware that local NLP solvers launched on nonconvex formulations do not guarantee that they will find the global optimum. A way to increase chances to find global optima of a nonconvex MP formulation

$$P \equiv \min\{f(x) \mid \forall i \leq m\ g_i(x) \leq 0 \wedge x \in \mathbb{R}^n\},$$

where $f, g_i : \mathbb{R}^n \to \mathbb{R}$ are functions at least one of which is nonconvex, is to run a simple *multi-start algorithm*:

$x^* = 0 \in \mathbb{R}^n$ *#this is the solution*
$f^* = \infty$ *#this is its optimal value*
`iteration` $= 0$
while `iteration` $\leq$ `limit` **do**
  $x^0 = \mathsf{random\_vector}(n)$
  $(x', f') = \mathsf{solver}(P, x^0)$
  if $f' < f^*$ **then**
    $x^* \leftarrow x'$
    $f^* \leftarrow f'$
  end if
  `iteration` $\leftarrow$ `iteration` $+ 1$
end while
return (x^*, f^*)

where `limit` is a given iteration limit. This algorithm is very easy to implement in either AMPL or Pyhon.

Projects

1. Create a graph-of-words from a sentence, enrich it with semantic distances, then use MP formulations for DG to embed the graph in a low-dimensional space. You will obtain a *word embedding* for the sentence. Test your word embedding on a natural language processing task.

2. Try solving the DGP system of quadratic equations using Newton's method. Chances are, it won't work very well. Improve the situation by devising methods for selecting a good starting point. Consider solving the system using Gröbner's bases.

3. (Theoretically challenging) Consider the objective function $F \bullet X$ for the SDP formulation of the DGP given in the course. Consider choosing F as advised in [1]. Explain why Barvinok's choice decreases the rank. Verify experimentally whether this choice really achieves what it promises.

4. (Even more challenging) Look for a theoretical estimation of the difference between the optimal objective function the SemiDefinite Programming (SDP) "push-and-pull" formulation and that of the corresponding primal and dual Diagonally Dominant Programming (DDP) formulation.

5. Form and realize the graph of secondary structures of a protein: detect them using means and variances of bond angles on the backbone, then connect them with edges if some of their atoms are connected by edges in the protein graph (the weight of the secondary structure edges could be the sum of the weights of the corresponding protein graph edges). More precise instructions:

   - download some proteins from the PDB database, each should tell you the backbone and the 3D positions for each atom (you have to read the documentation in order to know which fields are relevant, you can probably also find Python interfaces to the PDB)
   - using the 3D coordinates of the atoms of the backbone, find out all of the bond angles
   - use statistics on the means and variances of bond angle sines/cosines computed of subsequences of fixed size k (loop for $k \in \{5, \ldots, 50\}$), find what subsequences refer to beta-strand, alpha-helices, loops, according to the rule: alpha-helix: nonzero mean, low variance; beta-strand: zero mean, low variance; loop: high variance
   - form a protein backbone graph where the vertices are the backbone atoms, and each edge $\{u, v\}$ means that the distance between atom u and atom v is $\leq 5\mathring{A}$, encode this in a `.dat` file format like `tiny_gph.dat`
   - form a secondary structure graph where the vertices are beta-strands, alpha-helices, loops. Connect two of them with an edge $\{A, B\}$ if some atoms in A are connected to some atoms in B in the protein backbone graph. Assign a distance to edges which is equal to the sum of the corresponding distances in the protein backbone graph
   - realize the protein backbone graph and the corresponding secondary structure graph. Try and "superimpose" them (to be discussed).

6. Verify whether random projections work well on LPs of a few interesting classes: the diet problem, max-flow problems, multicommodity flow problems (LP); max stable set, facility location (MILP), etc. In general, compare a bisection code for original formulations to a bisection code for projected formulations; measure and compare objective function values and CPU times. More precise instructions:

   - you can use a single code for bisection. This code should read an LP or MILP in `.mps` or `.lp` format, it should transform the objective function $\min c^\top x$ to a constraint

$c^\top x \leq c_0$, add it to the constraints $Ax = b \wedge x \geq 0$, and then perform bisection on the feasibility problem $c^\top x \leq c_0 \wedge Ax = b \wedge x \geq 0$; this code can be written in AMPL or in Python, as you wish

- you need to write another code, in Python this time, which reads an LP or MILP in `.mps` or `.lp` format, reduces to standar form $\min\{c^\top x \mid Ax = b \wedge x \geq 0\}$, and then produces, in output, an `.mps` or `.lp` format of the projected problem $\min\{c^\top x \mid TAx = Tb \wedge x \geq 0\}$. This code should also output the CPU time taken to do the projection
- you need to write different codes, all in AMPL, which model some LP classes and output instances in `.mps` or `.lp` format (use the AMPL command `write minstancename;` to output to `instancename.mps`).
- finally, you need to write random instance generators for each of the AMPL codes that model LP classes.

References

[1] A. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete and Computational Geometry*, 13:189–202, 1995.