

When one solve for X using the full (or complete) matrix U , one get a vector \mathbf{w} with 32 non zero weight. Using this information, I selected those 32 edges as the “random” ones at the beginning the iterative procedure. Even in that case, `quadprog` failed to get any meaningful answer (even worse, starting from the original complete solution does not seem to help). Still, I found one bad index error I made in the code that add edges. Then I tried SDPT3 using YALMIP to describe the problem:

Listing 1:

```
y=sdpvar(m,1);
Constraints=[y>=0, A*y-ones(n,1)>=0];
Objective=y'*H*y;
option=sdpssettings('solver', 'sdpt3', 'savesolveroutput', 1);
sol=solvesdp(Constraints, Objective, option);
w = double(y);
f=-mean(sol.solveroutput.obj);
```

Things were somewhere better: strangely enough, weights on excluded edges are around $4.41 \cdot 10^4$ but the ones on included edges match those of the complete solution. Yet there are other issues:

- I don't know how to start from a initial solution (although there is the `usex0` option).
- I don't know how to get Lagrange multipliers z which may cause some troubles when computing the derivative (which will be needed in the case we do not cheat by selecting all the right edges at first).

I also tried to complete the full solution on the iris dataset to see if it make sense by testing the classification method but I soon renounced. Going from $n = 15$ to $n = 24$ change solving optimization time from 0.5 seconds to 7.5 and I think the complexity is $O(n^4)$ so it would have taken hours.

All the following is subject to $\mathbf{w}, \mathbf{s} \geq 0$:

$$\begin{aligned}
\mathcal{L} &= \min_{\mathbf{w}, \mathbf{s}} ||M\mathbf{w}||^2 + \mu ||\mathbf{1} - A\mathbf{w} - \mathbf{s}||^2 \\
&= \min_{\mathbf{w}, \mathbf{s}} \mathbf{w}^T M^T M \mathbf{w} + \mu ((\mathbf{1}^T - \mathbf{w}^T A^T - \mathbf{s}^T)(\mathbf{1} - A\mathbf{w} - \mathbf{s})) \\
&= \min_{\mathbf{w}, \mathbf{s}} \mathbf{w}^T M^T M \mathbf{w} + \mu (\mathbf{1}^T \mathbf{1} - \mathbf{1}^T A \mathbf{w} - \mathbf{1}^T \mathbf{s} - \mathbf{w}^T A^T \mathbf{1} + \mathbf{w}^T A^T A \mathbf{w} + \mathbf{w}^T A^T \mathbf{s} - \mathbf{s}^T \mathbf{1} + \mathbf{s}^T A \mathbf{w} + \mathbf{s}^T \mathbf{s}) \\
&= \min_{\mathbf{w}, \mathbf{s}} \mathbf{w}^T (M^T M + \mu A^T A) \mathbf{w} + \mu \mathbf{s}^T I^T I \mathbf{s} + \mu (1 - 2\mathbf{1}^T A \mathbf{w} - 2\mathbf{1}^T \mathbf{s} + 2\mathbf{w}^T A^T \mathbf{s}) \\
&= \min_{\mathbf{y}} \mathbf{y}^T \underbrace{\begin{pmatrix} M^T M + \mu A^T A & \mu A^T \\ \mu A & \mu I \end{pmatrix}}_{C^T C} \mathbf{y} - 2 \underbrace{\begin{pmatrix} \mu \mathbf{1}^T A \\ \mu \mathbf{1}^T \end{pmatrix}}_{d^T} \mathbf{y} + \underbrace{\mu \mathbf{1}^T \mathbf{1}}_{d^T d} \\
&= \min_{\mathbf{y}} \mathbf{y}^T H \mathbf{y} + \mathbf{f} \mathbf{y}
\end{aligned}$$

where \mathbf{y} is the $m + n$ vector $\begin{pmatrix} \mathbf{w} \\ \mathbf{s} \end{pmatrix} \geq 0$. Yet it still does not look like:

$$\min_{\mathbf{y}} ||C\mathbf{y} - \mathbf{d}||^2 = \min_{\mathbf{y}} \mathbf{y}^T C^T C \mathbf{y} - 2\mathbf{d}^T C \mathbf{y} + \mathbf{d}^T \mathbf{d}$$

because of M .

Listing 2: Solving minimization problem with the subset method

```
function [w, Aw, L] = compute_graph(X, kind, mu)
% All stories have a hero and our is not different. So meet X, a handsome and
% brave set of  $n$   $d$ -dimensional vectors.
[n, d] = size(X);
m = nchoosek(n, 2);
% She is in love with the equally beautiful U, a matrix containing all the
%  $m$  possible edges between X's nodes.
U = sparse(n, m);
% Yet for now, U, like many boys of his age, is quite empty. Therefore X,
% must fill him with the weights in w. But to be fair, she has no feasible
% solution to propose so far.
w = [];
% Fortunately, she will be helped by some friends, although they shall be
% presented later, as they are, with all due respect, mainly calculations'
% artifice (and as such, they don't have a clue about how to start).
M = sparse(d*n, m);
MAX_ITER = 50;
nb_iter = 0;
bin_upper = n*(0:n-1) - cumsum(0:n-1);
considered_last_time = [];

% One day, an old man told X that she could for instance start with this
% random small subset: a third of  $(d+1)n$  edges, as this was indeed common
% knowledge, provided by a book called Theorem 3.1, that U cannot contains
% more. At this point, the astute reader may wonder why we do not use a more
% sensible initial choice like bind each node with its closest neighbor. Well,
% I am telling the story so we do like this. But feel free to contribute!
edges = randi(m, 1, floor(0.33*(d+1)*n));

% And thus begin the quest of X, until she can not add more edges to U or
% until she get fed up and realize that organizing illegal fights of turtles is
% much more exciting than finding love.
while (numel(edges) > 0 && nb_iter < MAX_ITER)
    % U was also quite stubborn and felt that linear indexing of edges
    % was not doing justice to his amazing 2D abilities. Therefore X has
    % to resort to her cunning to convert them. The edges 1 through  $n-1$ 
    % were from  $i=1$ , those from  $n$  to  $n+(n-2)-1=2n-3$  started at  $i=2$ 
    % and so on. It turns out that bin_upper has memorized all these
    % upper bounds so finding  $i$  was simply a matter of finding the
    % maximum possible bounds.
    vertex_i = arrayfun(@(x) find(x' <= bin_upper, 1, 'first'), edges) - 1;
    % Then j follows
    vertex_j = vertex_i + edges - bin_upper(vertex_i);
    % and the pairs (i,j) could be converted into U indexes
    positive = bsxfun (@(x,y) sub2ind(size(U), x, y), vertex_i, edges);
    negative = bsxfun (@(x,y) sub2ind(size(U), x, y), vertex_j, edges);
    % in order to represent the newly selected edges.
    U(positive) = 1;
    U(negative) = 0;
    nb_iter = nb_iter + 1;
end
```

```

U(negative) = -1;
A = abs(U);
assert(sum(A(:))/2 <= (d+1)*n, 'there are too many edges');

% To assess their compatibility, the tradition was to compute the
% Frobenius norm of  $X$  times the graph Laplacian. Both find this
% method awkward and they choose to reformulate it as an Euclidean
% distance. But doing so require the help of a friend:  $M$ .
T = U'*X; %  $y^{(k)} = U^T x^k$  is thus the  $k$ th column of  $T$ 
% First  $X$  mixes her columns with  $U$ , producing  $d$  new vectors of
% length  $n$ :  $y^{(k)}$ .
for k=1:d
    first_row = 1 + (k-1)*n;
    last_row = n + (k-1)*n;
    % These new vectors were soon promoted as diagonal matrices and
    % filled  $M$  from top to bottom (although it would have been
    % faster to do it in parallel).
    Yk = spdiags(T(:,k), [0], m, m);
    M(first_row:last_row, :) = U*Yk;
end

% Having done all this preparatory work,  $X$  could finally go see an
% oracle living in the mountain, the so called quadprog, and ask him to
% set  $w$  optimally according to  $M$ . (Actually, she had also heard of
% another one, SDPT3, potentially faster and able to deal with sparse
% matrix instead of converting  $M'*M$  to a full one and taking  $2n^4$ 
% bytes of memory. But she had to ask her question in a slightly
% different language:
% http://www.math.nus.edu.sg/mattohkc/sdpt3/guide4-0-draft.pdf
% http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Tutorials.LinearAndQuadraticProgramming
if strcmpi(kind, 'hard')
    % In one method, she had to ensure that the weighted sum of
    % degree was at least one for each node, or in the language of
    % the oracle:  $-Aw \leq -1$ . But this was only possible
    % for the nodes that were part of at least one edge, that is
    % for the non zero rows of  $A$ .
    o = optimoptions(@quadprog, 'Algorithm', 'active-set', 'Display', 'final-detailed');
    [w, f, flag, output, lambda] = quadprog(M'*M, sparse(m, 1), -A, -(sum(A, 2)>0), [], [], [], [], w, o);
    z = lambda.ineqlin;
    derivative = 2*M'*M*w - A'*z;
    save('out', 'M', 'w', 'A', 'lambda', 'derivative');
else
    % There was another method where a portion  $\alpha$  of the nodes
    % were allowed to have degree less than one. But she still
    % has to think about to formulate
    %  $\min_{w,s} ||Mw||^2 + \mu ||1 - Aw - s||^2$ 
    % for quadprog or lsqnonneg (http://math.stackexchange.com/q/545280)

```

```

    error(strcat(kind, ' is not yet implemented'));
end
% Because the new (w,z) were supposed to be feasible solution,
%  $\frac{d\Delta}{dw}$  has to be positive. Therefore, she finds the
% edges where it was not the case to add them in the next step.
[val, may_be_added]=sort(derivative(find(derivative<0)));
% Of course maybe there was nothing to do. Or more concerning, the
% oracle was rambling and returned a solution that yields the same set
% of edges to add as previously, in which case there was no point in
% continuing any further.
if ((length(may_be_added) == 0) || (length(may_be_added) == length(
    considered_last_time) && all(may_be_added' == considered_last_time)))
    break;
end
% She decide to add only half of them but probably there were other
% ways of doing it (like adding the "smallest one" ?)
edges = may_be_added(1:max(1, floor(end/2)))';
considered_last_time = may_be_added;
nb_iter = nb_iter + 1;
end
% When X finds the perfect weights for her graph (and hopefully not because
% she just give up), she have to fill some paperwork like computing weighted
% degree and Laplacian to make their union official.
Aw = A*w;
W = spdiags (w, [0], m, m);
L = U*W*U';
end

```

Listing 3: Computing hard graph

```

function [w, Aw, L] = compute_hard_graph(X)
    [w, Aw, L] = compute_graph(X, 'hard');
end

```

Listing 4: Computing α -soft graph

```

function [w, Aw, L] = compute_alpha_graph(X, alpha, tol)
[n, d] = size(X);
m = nchoosek(n, 2);
mu = 5*rand();
tau0 = 1.5;
MAX_ITER = 50;
% Set  $\lambda$  so that  $\tau \geq 1$  with equality at MAX_ITER.
lambda = (tau0 - 1)/MAX_ITER;
can_improve = true;
while (can_improve)
    w, Aw, L = compute_graph(X, 'soft', mu);
    tmp = max(zeros(m, 1), ones(m, 1) - A*w);
    alpha_bar = tmp'*tmp/n;
    % Maybe we don't need this complication and keep a fixed  $\tau = \tau_0$ .

```

```

tau = tau0/(1 + nb_iter*lambda);
if (alpha_bar < alpha)
    % We want to increase  $\bar{\alpha}$  so we need decrease  $\mu$ , which in
    % turn require  $\tau \leq 1$ 
    tau = 1/tau;
end
% It is supposed to correspond to: "we then adjust  $\mu$  up or down
% proportionally to how far  $\frac{\eta(w)}{n} = \bar{\alpha}$  is from the
% desired value of  $\alpha$ ."
mu = tau*abs(alpha_bar - alpha)/alpha
nb_iter = nb_iter + 1;
can_improve = abs(alpha_bar - alpha) < tol && nb_iter < MAX_ITER;
end
end

```

Listing 5: Use the built graph to classify samples

```

function result = graph_classify(labelled, label, unlabelled)
n = numel(label);
u = size(unlabelled, 1);
X = [labelled; unlabelled];
[w, Aw, L] = compute_hard_graph(X);
beq = [label; zeros(u, 1)];
Aeq = [eye(n) zeros(n, u); zeros(u, n+u)];
% TODO look at the fast method suggested in the paper: Spielman, D. A.,
% Teng, S.-H. (2004). Nearly-linear time algorithms for graph partitioning,
% graph sparsification, and solving linear systems. Proc. 36th ACM STOC.
x = quadprog(L, [], [], [], Aeq, beq);
result = x(n+1:end) > 0.5;
end

```