When one solve for X using the full (or complete) matrix U, one get a vector $\boldsymbol{w}$ 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=sdpsettings('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.

There is a typo in the paper, it should be $+\boldsymbol{s}$ instead of $-\boldsymbol{s}$ (otherwise there is a trivial solution: $\boldsymbol{s} = \mathbf{1}$ and $\boldsymbol{w} = \mathbf{0}$. Actually it was corrected in the video's slides, but I only remembered it too late). All the following is subject to $\boldsymbol{w}, \boldsymbol{s} \geq 0$:

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

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

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

because of M.

Nonetheless, as there is no constraints and thus no Lagrange multipliers:

$$\frac{d\Lambda}{d\boldsymbol{w}} = 2(M^T M + \mu A^T A)\boldsymbol{w} - 2\mu A^T(\boldsymbol{s} + \mathbf{1})$$

Listing 2: Solve in one call (for testing purpose, only try this on very small dataset)

```
function [w, A, H, f, L, s] = fully_solve(X, kind, mew)
n = size(X, 1);
m = nchoosek(n, 2);
w = [];
[H, U] = get_complete_matrices(X);
f = sparse(m, 1);
A = abs(U);
A_constraint = -A;
b = -ones(n, 1);
lower_bound = zeros(m, 1);
if strcmpi(kind, 'soft')
  H = [H+mew*(A'*A) -mew*A'; -mew*A mew*eye(n)];
  % remove the factor 2 as MATLAB optimize 1/2 x'*H*x + f'*x
  f = -mew*[ones(1, n)*A -ones(1, n)]';
  A_constraint = [];
  b = [];
  lower_bound = zeros(m+n, 1);
end
if (validatestring(version('-release'), {'2013a', '2013b'}))
  o = optimoptions(@quadprog, 'Algorithm', 'interior-point-convex', 'MaxIter', 500, '
     Display', 'off', 'TolFun', 1e-15);
else
  o = optimset('Algorithm', 'interior-point-convex', 'MaxIter', 500);
end
[w] = quadprog(H, f, A_constraint, b, [], [] , lower_bound, [] , w, o);
if strcmpi(kind, 'soft'); s = w(m+1:end); w = w(1:m);
sprintf('%f\t%f', norm(H(1:m,1:m)*w)^2, mew*norm(ones(n,1) - A*w +s)^2)
end
L = U*diag(w)*U';
end
```

Listing 3: Solving minimization problem with the subset method

```
function [w, A, H, f] = 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 = [];
[HK, UK] = get_complete_matrices(X);
AK = abs(UK);

% 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(0.1*(d+1)*n));
% shamelessly cheating!
% load('edges.mat');
%
% 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 is was simply a matter of finding the
  % maximum possible bounds.
  [positive, negative] = from_edges_to_index(edges, bin_upper, size(U));
  % in order to represent the newly selected edges.
  U(positive) = 1;
  U(negative) = -1;
  [is, js] = from_edges_to_index(to_remove, bin_upper, size(U));
  U(is) = 0;
  U(js) = 0;
  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 kth 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
H = M'*M;

% 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
if strcmpi(kind, 'hard')
  % In one method, she had to ensured that the weighted sum of
  % degree was at least one for each node, or in the language of
  % the oracle:  -Aw <= -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).
  % y=sdpvar(m,1);
  % C=[y>=0, A*y-ones(n,1)>=0];
  % O=y'*(H)*y;
  % os=sdpsettings('solver', 'sdpt3', 'savesolveroutput', 1, 'usex0', 1, 'verbose', 0);
  % tic;
  % sol=solvesdp(C,O,os);
  % w = double(y);
  % toc
  % f=-mean(sol.solveroutput.obj);
  % do only one iteration since we need a way to compute the derivative
  % break;
  if (strmatch('2013', version('-release')))
    o = optimoptions(@quadprog, 'Algorithm', 'interior-point-convex', 'MaxIter',
        500);
  else
    o = optimset('Algorithm', 'interior-point-convex', 'MaxIter', 500);
  end
  % [w, f, flag, output, lambda] = quadprog(H, sparse(m, 1), -A, -(sum(A, 2)>0), [], [], zeros(m,1),
      [], w, o);
  [w, f, flag, output, lambda] = quadprog(H, sparse(m, 1), -A, -(ones(n, 1), [],
      [], zeros(m,1), [], w, o);
```

```
    z = lambda.ineqlin;
    % TODO: since quadprog put a factor 1/2 in front of H, make
    % sure the derivative is correct.
    derivative = 2*HK*w - AK'*z;
    save('out.mat', 'lambda', 'derivative');
    % break;
  else
    % There was another method were a portion $\alpha$ of the nodes
    % were allowed to have degree less than one.  But she still
    % has to think about to formulate
    % $\min_{w,s} ||Mw||^2 + \mu||\mathbf{1} - Aw - s||^2$
    % 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,
  % $\frac{d\Lambda}{dw}$ 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)));
  % This was badly erroneous
  may_be_added = find(derivative<0)'
  % perform cheap regularization, namely remove weirdly large value
  % (find another way to do it in general)
  w(w>1e6) = 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 ((isempty(may_be_added)) || (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)))';
  edges = may_be_added;
  % this is quite arguable
  w(may_be_added) = mean(w);
  considered_last_time = may_be_added;
  to_remove = find(w<1e-8)';
  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.
% TODO: When it will work, use this as output argument
Aw = A*w;
W = spdiags (w, [0], m, m);
L = U*W*U';
end
```

Listing 4: Computing hard graph

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

Listing 5: Computing α-soft graph

```
function [w, Aw, L, report] = compute_alpha_graph(X, alpha, tol)
[n, ~] = size(X);
m = nchoosek(n, 2);
mew = normrnd(1, .2);
% tau0 = 1.5;
nb_iter = 1;
MAX_ITER = 13;
% Set λ so that τ ≥ 1 with equality at MAX_ITER.
% lambda = (tau0 - 1)/MAX_ITER;
can_improve = true;
report = zeros(MAX_ITER, 5);
while (can_improve)
  % [w, Aw, L] = compute_graph(X, 'soft', mew);
  [w, A, H, f, L] = fully_solve(X, 'soft', mew);
  tmp = max(zeros(n, 1), ones(n, 1) - A(:,1:m)*w);
  alpha_bar = tmp'*tmp/n;
  % Maybe we don't need this complication and keep a fixed τ = τ_0.
  % tau = tau0/(1 + nb_iter*lambda);
  delta = (alpha_bar - alpha)/alpha;
  % if (alpha_bar < alpha)
    % We want to increase ᾱ so we need decrease μ,
    % which in turn require τ ≤ 1.
  % tau = 1/tau;
  % end
  % It is supposed to correspond to: "we then adjust μ up or down
  % proportionally to how far η(w)/n = ᾱ is from the
  % desired value of α."
  rep = [alpha_bar abs(alpha_bar - alpha)/alpha tau mew];
  % actually, it's a bit weird to reduce μ when δ is
  % negative, because it happens when ᾱ < α, meaning
  % that the weights satisfy the constraints even "more" that what we
  % are looking for.
  mew = ((abs(delta)+1)^sign(delta))*mew;
  rep = [rep mew];
  report(nb_iter, :) = rep;
  nb_iter = nb_iter + 1;
  can_improve = (abs(alpha_bar - alpha) > tol) && (nb_iter <= MAX_ITER);
end
Aw = A(:,1:m)*w;
sprintf('\ta_bar\trel\ttau\tmu_n\tmu_n+1')
report
end
```

Listing 6: 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
```