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

$$
\begin{aligned}
\mathcal{L} &= \min_{\boldsymbol{w},\boldsymbol{s}} ||M\boldsymbol{w}||^2 + \mu||\mathbf{1} - A\boldsymbol{w} - \boldsymbol{s}||^2 \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T M^T M\boldsymbol{w} + \mu\left((\mathbf{1}^T - \boldsymbol{w}^T A^T - \boldsymbol{s}^T)(\mathbf{1} - A\boldsymbol{w} - \boldsymbol{s})\right) \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T M^T M\boldsymbol{w} + \mu\left(\mathbf{1}^T\mathbf{1} - \mathbf{1}^T A\boldsymbol{w} - \mathbf{1}^T\boldsymbol{s} - \boldsymbol{w}^T A^T\mathbf{1} + \boldsymbol{w}^T A^T A\boldsymbol{w} + \boldsymbol{w}^T A^T\boldsymbol{s} - \boldsymbol{s}^T\mathbf{1} + \boldsymbol{s}^T A\boldsymbol{w} + \boldsymbol{s}^T\boldsymbol{s}\right) \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T(M^T M + \mu A^T A)\boldsymbol{w} + \mu\boldsymbol{s}^T I^T I\boldsymbol{s} + \mu\left(1 - 2\mathbf{1}^T A\boldsymbol{w} - 2\mathbf{1}^T\boldsymbol{s} + 2\boldsymbol{w}^T A^T\boldsymbol{s}\right) \\
&= \min_{\boldsymbol{y}} \boldsymbol{y}^T \begin{pmatrix} M^T M + \mu A^T A & 0 \\ 0 & \mu I \end{pmatrix} \boldsymbol{y} + \begin{pmatrix} -2\mu\mathbf{1}^T A \\ -2\mu\mathbf{1}^T \end{pmatrix} \boldsymbol{y} + 2\mu\boldsymbol{w}^T A^T\boldsymbol{s}
\end{aligned}
$$

Listing 1: Solving minimization problem with the subset method

```
function [w, Aw, L] = compute_graph(X, kind, mu)
[n, d] = size(X);
m = nchoosek(n, 2);
M = sparse(d*n, m);
U = sparse(n, m);
w = [];
MAX_ITER = 50;
nb_iter = 0;

% At the end, we cannot have more than (d+1)/n edges according to
% Theorem 3.1.  But we must start with only a small subset of them.  So we first
% select randomly 7% of them (for no specific reason but cinematographic
% one).  Actually, it may be more sensitive to use some kind of heuristic like
% nearest neighbors at this stage to be more efficient later.

edges = randi(m, 1, 0.07*(d+1)*n);

while (numel(edges) > 0 && nb_iter < MAX_ITER)
  % Remove the multiple of n to avoid self loop (and the first one in case
  % it's 1).
  edges = edges(mod(edges, n) ~= 0);
  edges = edges(2:end);
  % Then we update the corresponding element (i,j) = e of U with
  % respectively 1 and -1.
  vertex_j = rem(edges, n);
  vertex_i = mod(edges, n) + 1;
  positive = bsxfun (@(x,y) sub2ind(size(U), x, y), vertex_i, edges);
  negative = bsxfun (@(x,y) sub2ind(size(U), x, y), vertex_j, edges);
  U(positive) = 1;
  U(negative) = -1;
  A = abs(U);
  assert(sum(A(:))/2 <= (d+1)*n, 'there are too many edges');
```

```
  T = U'*X; % y^(k) = U^T x^k is thus the kth column of T
  % TODO: use parfor
  for k=1:d
    first_row = 1 + (k-1)*n;
    last_row = n + (k-1)*n;
    Yk = spdiags(T(:,k), [0], m, m);
    M(first_row:last_row, :) = U*Yk;
  end
  % Now that we have built our matrices, we can solve the minimization problem
  % TODO use SDPT3, although the documentation is quite intimidating:
  % http://www.math.nus.edu.sg/ mattohkc/sdpt3/guide4-0-draft.pdf
  if strcmpi(kind, 'hard')
    % we only want to constrain the nodes that have edges to be of
    % degree at least 1.
    [w, f, flag, output, lambda] = quadprog(M'*M, sparse(m, 1), -A, -(sum(A, 2)>0),
        [], [], [], [], w);
    z = lambda.ineqlin;
    derivative = 2*M'*M*w - A'*z;
  else
    % According to the paper, we want to solve
    % min_{w,s} ||Mw||^2 + μ||1 - Aw - s||
    % subject to w, s ≥ 0, but I don't see how to formulate that for
    % quadprog or lsqnonneg (see http://math.stackexchange.com/q/545280)
    error(strcat(kind, ' is not yet implemented'));
  end
  [val, may_be_added]=sort(derivative(find(derivative<0)));
  % The paper says:  we add to our quadratic program the edges with the smallest
  % dΛ/dw_{i,j} values, which I think mean not all.  For now,
  % let's take half of them.  TODO take only the one below average or look at
  % diff.
  edges = may_be_added(1:end/2);
  nb_iter = nb_iter + 1;
end
Aw = A*w;
W = spdiags (w, [0], m, m);
L = U*W*U';
```

Listing 2: Computing hard graph

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

Listing 3: 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 λ so that τ ≥ 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 τ = τ0.
  tau = tau0/(1 + nb_iter*lambda);
  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."
  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 4: 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
```