On the solution of large-scale SDP problems by the modified barrier method using iterative solvers, Michal Kočvara and Michael Stingl

This paper continues a line of recent papers in computational semidefinite programming (SDP), which attempt to extend second-order interior-point methods to large-scale SDP by using Krylov-type methods to solve the "Newton" equation arising in each iteration of such methods. Some key questions that the paper addresses are: when could one expect a conjugate-gradient approach to outperform a regular second-order method; and what preconditioning techniques are available in the context of SDP? The primary perspective for analysis is the performance of the authors' code, PENNON.

Both questions are successfully answered in the paper through careful discussion and thorough experiments. The authors should be commended for presenting a multifaceted topic in such an understandable manner.

In my opinion, the main contribution of the paper is additional insight into the challenges of extending algorithms to large-scale SDP. After reading the paper, one is left with a mixed impression: some success has been made, but still much work needs to be done. At this point in time, I think this is the correct impression to have. Accordingly, I would like to recommend the paper for publication subject to revisions that incorporate the following suggestions:

- The second-half of the third full paragraph on p.2 should be edited for greater clarity. It is not clear what is similar to, and what is different from, the paper [22].
- Doesn't the Lagrange multiplier U need to be positive semidefinite? This should be stated explicitly, and it should be highlighted that equation (8) keeps U positive semidefinite because the algorithm will also keep $Z(x)$ positive definite.
- At the top of p.4, what is $D_{\mathcal{A}}$? Also, Φ_p has two arguments here, whereas its definition has just one.
- On p.4, any cluster point will constitute an optimal solution of (1) and its dual.
- In step 4 of the penalty update procedure, what is l?
- In the penalty update procedure, how do you get x_{feas}? Isn't this as difficult as solving the SDP itself?
- On p.8, in the context of PENNON, a specific contribution of the paper is the use of finite differences to calculate Hessian-vector products. The authors explain that this is necessary if one wishes to save memory — the implication being that the only other alternative for performing Hessian-vector products requires the explicit construction of the Hessian. But is it not possible to represent the Hessian in an "operator form" that shows how Hessian-vector products can be done without

explicitly construction the Hessian? For example, a typical primal-dual method Newton system is something like

$$\mathcal{A}^*(X\mathcal{A}(d)Z^{-1}) = -g.$$

So a Hessian-vector product can be done without explicitly constructing the Hessian or doing finite differences by applying $\mathcal{A}$, then pre- and post-multiplying by X and Z^{-1}, respectively, and finally applying $\mathcal{A}^*$ (the adjoint of $\mathcal{A}$). Could the authors please address this issue?

- Figures such as Fig. 1-3 would be easier to interpret if the scale for the vertical axes in the left and right charts were the same.
- On p.15, it is stated that only the BFGS preconditioner was tested in the case of PEN-A-PCG. Was the AINV preconditioner not appropriate — even though it was stated as only requiring Hessian-vector products? If AINV is not really important for the paper (the only other mention seems to be Fig. 8), then please remove Section 4.6.
- The table numbers mentioned in the text do not match the actual table numbers.
- Could the authors provide some additional details about Fig. 9 and 10? In particular, is Fig. 9 with respect to PEN-PCG or PEN-A-PCG? I understood that PEN-PCG could not be run on most of the TOH problems because of PEN-PCG's need to construct the Hessian explicitly. Or does Fig. 9 cover just a subset of TOH? PEN-PCG is also represented in Fig. 10.
- When running SDPLR on the theta problems, was SDPLR run with knowledge that A_0 is the rank-1 matrix of all ones, ee^T? SDPLR is able to exploit this structure and can benefit greatly over a situation in which ee^T is expressed as a general matrix.
- In Tables 8 and 9, five of the six problems are theta problems. Could the authors present a somewhat more representative sample?

Some small types and minor corrections are:

- (abstract,5) *with a large*
- (p.1,6) "the Newton method"?
- (p.2,-15) *there are a few significant differences*
- (p.3,3) *examples of ill-conditioned*
- (p.3,6) *with different backgrounds.*
- (p.4,16) *minimization in Step (i)*

- (p.6,-2) *certain kinds of problems*
- (p.7,12) *takes the most time*
- (p.7,17) *From the complexity viewpoint*
- (p.7,19) *in exact arithmetic, the* (other instances)
- (p.8,6) *may use a finite*
- (p.8,10) *amounts to the number*
- (p.9,-14) *in a few examples*
- (p.9,-10) κ_{opt}
- (p.16,20) *robust than the*
- (p.16,-11) *examples the PEN-PCG(BFGS) is about as fast*
- (p.16,-12) *in a few cases*
- (p.16,-11) *typical of a high*
- (p.17,3) Do you mean *BFGS* instead of *diag*?
- (p.17,-7) *are typified by*
- (p.17,-4) *There are, however, still a few*
- (p.17,-1) *"good"*