Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Discrete Mathematics 41 (2013) 157–164

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

Approximating the Lovász θ Function with the Subgradient Method

Monia Giandomenico [1]
Department of Information Engineering, Computer Science and Mathematics
University of L'Aquila
L'Aquila, Italy

Adam N. Letchford [2]
Department of Management Science
Lancaster University
Lancaster, United Kingdom

Fabrizio Rossi, Stefano Smriglio [1]
Department of Information Engineering, Computer Science and Mathematics
University of L'Aquila
L'Aquila, Italy

Abstract

The famous Lovász theta number $\theta(G)$ is expressed as the optimal solution of a semidefinite program. As such, it can be computed in polynomial time to an arbitrary precision. Nevertheless, computing it in practice yields some difficulties as the size of the graph gets larger and larger, despite recent significant advances of semidefinite programming (SDP) solvers. We present a way around SDP which exploits a well-known equivalence between SDP and lagrangian relaxations of non-convex quadratic programs. This allows us to design a subgradient algorithm which is shown to be competitive with SDP algorithms in terms of efficiency, while being preferable as far as memory requirements, flexibility and stability are concerned.

Keywords: Maximum Stable Set, Quadratic Programming, Lagrangian Relaxation, Subgradient Method.

http://dx.doi.org/10.1016/j.endm.2013.05.088

1 Introduction

Given an undirected graph $G = (V, E)$, the Lovász θ number, introduced in [11], is defined as follows:

$$\begin{aligned} \theta(G) := \max \; & \sum_{i \in V} Y_{0i} \\ \text{s.t.} \; & Y_{0i} = Y_{ii} \quad (i \in V) \\ & Y_{ij} = 0 \quad (\{i, j\} \in E) \\ & Y \in \mathcal{S}_{n+1}, \end{aligned} \tag{1}$$

where $\mathcal{S}_{n+1}$ denotes the cone of real symmetric square positive semidefinite matrices of order $n+1$. $\theta(G)$ plays a central role in combinatorial optimization as it can be computed in polynomial time to an arbitrary precision (being the optimal value of a Semidefinite Program) [9] but, at the same time, provides tight bounds for the *stability number* $\alpha(G)$ and for the *chromatic number* $\chi(G)$ of a graph. In detail, $\alpha(G) \leq \theta(G) \leq \chi(\bar{G})$, where $\bar{G}$ is the complement graph of G. These nice features led to several interesting results, such as a polynomial time algorithm to compute $\alpha(G)$ and $\chi(G)$ when G is a perfect graph [9] However, most of the results based on the θ number have mainly theoretical significance. On the contrary, the practical potential of this strong bound has not yet been fully exploited. In fact, computing $\theta(G)$ of graphs from standard libraries has been a popular test problem for general SDP solvers since the beginning of their developments (we refer the reader to the repository [4] for a complete list of available resources on Semidefinite Programming). Among them, the boundary point method of Povh *et al.* [16] and the regularization method of Malick *et al.* [13] turned out to be efficient to compute $\theta(G)$.

Conversely, one can consider Lagrangian reformulations of structured SDPs (as in the Spectral Bundle method by Helmberg and Rendl [10]) or other non-linear approaches (such as those by Burer and Monteiro [1] and, recently, Malick and Roupin [14]). Indeed $\theta(G)$ can be computed as the optimal solution of a certain Lagrangian dual problem. In this paper we describe a subgradient algorithm for solving this dual approximately, and thereby approximating $\theta(G)$ from above. Of course, the subgradient method is a basic first-order method for non-differentiable optimisation, that is known to suffer from slow convergence in many cases. Nevertheless, and surprisingly, we found that our particular implementation gave results that have a good trade-off between tightness and computing time.

[1] Email: `name.surname@univaq.it`

[2] Email: `a.n.letchford@lancaster.ac.uk`

2 Lagrangian relaxation

A description of $\theta(G)$ can be derived starting from the standard quadratic formulation of the stable set problem:

$$\max \sum_{i \in V} x_i$$

$$x_i^2 - x_i = 0 \qquad (i \in V) \tag{2}$$

$$x_i x_j = 0 \qquad (\{i,j\} \in E) \tag{3}$$

If we associate multipliers λ_i to constraints (2) and μ_{ij} to (3), the resulting Lagrangian relaxation has the form:

$$L(\lambda, \mu) = \max_{x \in \mathbb{R}^{|V|}} f(x, \lambda, \mu) = \max_{x \in \mathbb{R}^{|V|}} \sum_{i \in V} (1 + \lambda_i) x_i - \sum_{i \in V} \lambda_i x_i^2 - \sum_{\{i,j\} \in E} \mu_{ij} x_i x_j$$

or the following more compact form:

$$L(\lambda, \mu) = \max_{x \in \mathbb{R}^{|V|}} [(\mathbf{1} + \lambda)^T x - x^T A(\lambda, \mu) x] \tag{4}$$

with $A(\lambda, \mu) = Diag(\lambda) + M(\mu)$, where $M(\mu)$ denotes the symmetric matrix with $\mu_{ij}/2$ in the ith row and jth column whenever $\{i, j\} \in E$, and zeroes elsewhere. In [8] it is shown that the optimal value of the Lagrangian dual problem yields the Lovász theta bound, that is:

$$\theta(G) = \min_{(\lambda,\mu) \in \mathbb{R}^{|V|+|E|}} L(\lambda, \mu) = L(\lambda^*, \mu^*) \tag{5}$$

where λ^* and μ^* coincide with the optimal dual solution to problem (1). Note that the function $f(x, \lambda^*, \mu^*)$ is concave and the matrix $A(\lambda^*, \mu^*)$ is positive semidefinite. We remark that Luz and Schrijver [12] showed that an optimal solution $A(\lambda, \mu)$ of problem (5) exists such that $\lambda = \mathbf{1}$ and the minimim eigenvalue $\lambda_{\min}$ of $A(\mathbf{1}, \mu)$ is equal to zero. Observe however that imposing the second restriction causes the Lagrangian dual to become non-convex. We decided not to impose either restriction in our subgradient algorithm.

3 Subgradient algorithm

The subgradient algorithm is initialized with multipliers $\lambda^1 = \mathbf{1}$ and $\mu^1 = \mathbf{0}$, i.e., $A(\lambda^1, \mu^1) = I_{|V|}$. In all cases in which $A(\lambda, \mu)$ is positive definite (and obviously $I_{|V|}$ is so), the global minimizer $\tilde{x}$ of the lagrangian relaxation (4)

Algorithm 1 Update multipliers

```
Input:        $g^k$, $h^k$, $L(\lambda^k, \mu^k)$, $\epsilon$
Output:       true if succeeded, false otherwise,
              updated multipliers: $\lambda^{k+1}, \mu^{k+1}$,
              lagrangian solution: $\tilde{x}$
Parameters:   $\gamma$, step_tolerance, an estimate of $L(\lambda^*, \mu^*)$ : $\hat{L}$

/* Initialization */          $t := \gamma \frac{\hat{L} - L(\lambda^k, \mu^k)}{\|g^k\|^2 + \|h^k\|^2}$

/* Main loop */               while ($t \geq$ step_tolerance) {

/* Multipliers update */          $\lambda^{k+1} := \lambda^k + tg^k$,  $\mu^{k+1} := \mu^k + th^k$;

/* Check pd-ness */               if (chol($A(\lambda^{k+1}, \mu^{k+1})$))
                                  then compute $\tilde{x}$;
                                          return true;
/* Lambda perturbation */                 else $\lambda^{k+1} := \lambda^{k+1} + \epsilon \cdot \mathbf{1}$
                                              if (chol($A(\lambda^{k+1}, \mu^{k+1})$))
                                              then compute $\tilde{x}$
                                                      return true
/* Stepsize reduction */          $t := t/2$;
                                  }
                              return false;
```

is unique and can be analytically computed, being the root of the equation $\nabla_x f(x, \lambda, \mu) = -2Ax + (\lambda + \mathbf{1}) = \mathbf{0}$.

At the generic iteration k of a subgradient algorithm, one has to evaluate the search directions $g^k \in \mathbb{R}^{|V|}$, $h^k \in \mathbb{R}^{|E|}$ and to update the multipliers. In order to improve practical performance, search directions are evaluated by using a *deflection* strategy, that is, by defining the new search directions as a combination of the previous directions and the current subgradients [5]. In our implementation we use the deflection described in [19]. In detail:

$$g^k = \nabla_\lambda f(\tilde{x}) + \delta^k g^{k-1}; \quad h^k = \nabla_\mu f(\tilde{x}) + \delta^k h^{k-1}; \quad \text{with } \delta^k = \frac{\|\nabla^k_{\lambda,\mu} f(\tilde{x})\|}{\|(g^{k-1} \;\; h^{k-1})\|}.$$

As far as multiplier updates are concerned, we design an algorithm that maintains the matrix $A(\lambda^{k+1}, \mu^{k+1})$ positive definite through a “safeguard” strategy: Algorithm 1 receives the optimal value of the current Lagrangian relaxation $L(\lambda^k, \mu^k)$ and the search directions g^k, h^k. The procedure begins with updating the multipliers by a standard stepsize t. If the resulting matrix is positive definite, then multipliers λ^{k+1}, μ^{k+1} are returned and subgradient iteration $k+1$ starts. Otherwise, the positive definiteness is tentatively restored: first by applying a (small) perturbation ϵ to the diagonal of $A(\lambda^{k+1}, \mu^{k+1})$

(that is, to λ multipliers). Note that $\epsilon > |\lambda_{\min}(A(\lambda^{k+1}, \mu^{k+1}))|$ would guarantee the positive definiteness of the matrix. However, in practice, this can correspond to large values of ϵ that results in bound impairment since it is equivalent to adding a term $\epsilon \cdot \|x\|_2^2$ to $f(x, \lambda^{k+1}, \mu^{k+1})$. For this reason, we halve the stepsize and repeat the main loop until either the matrix is positive definite or t drops below a given tolerance `step_tolerance`.

The bottleneck of the algorithm is the pd-ness test that is carried out by the Cholesky Decomposition (see [9], Chapter 9.3 for details). Function `chol`(A) in Algorithm 1, based on the implementation developed in [2], returns `true` if the matrix is pd and `false` otherwise. If the matrix $A(\lambda^{k+1}, \mu^{k+1})$ is positive definite then `chol`(A) also returns a lower-triangular matrix L such that $A = LL^T$. This decomposition allows one to evaluate the current optimal solution of the lagrangian subproblem $-2Ax + (\lambda + \mathbf{1}) = \mathbf{0}$ being equivalent to the two triangular systems $-2Lz + (\lambda + 1) = 0$, $L^T x = z$.

A further device of our implementation is that the main loop execution can be easily parallelized. Precisely, a number `n_threads` of different sets of new candidate multipliers associated with different values of t can be computed in parallel. If multiple sets yield a pd matrix, the one corresponding to the largest stepsize is taken.

4 Computational Results

The test bed consists of graphs from the DIMACS Second Challenge available at the web site [3]. The SDP solver used is `mprw2` by Malik et al., which is publicly available in [18], running under MATLAB (R2008b). Such an algorithm currently outperforms other approaches on medium/large size instances. The subgradient algorithm is implemented in C++ in a `linux` architecture (Ubuntu 12.04), compiler `g++ 4.7`. The computations were run on a machine equipped by 2 Intel Xeon 5150 processors (for a total of 4 cores) clocked at 2.6 GHz and having 8GB of RAM. The parameter setting of Algorithm 1 is as follows: $\gamma = 0.8$, $\epsilon \in \{10^{-6}, 10^{-4}\}$, `step_tolerance` $= 10^{-12}$. The subgradient algorithm stops when it reaches either an iteration limit or an optimality tolerance fixed to 10^{-6} (same as the SDP solver).

In Table 1, besides the graph name, size, density and stability number, we report: a bound on $\theta(G)$ and the CPU time required by [18] to compute it; the best value Z found by the subgradient algorithm and the corresponding CPU time; the values $\theta_{105\%} = 1.05 \cdot \theta$ and $\theta_{110\%} = 1.1 \cdot \theta$ along with the CPU times required by the subgradient algorithm to reach these values ($***$ if $\theta_{105\%} < Z$).

The results give clear indications. In all cases Z is significantly close to $\theta(G)$, that is, the subgradient algorithm returns quite good solutions to (5). In several cases the SDP solver is faster, but also the subgradient algorithm takes reasonable CPU times. As pointed out in [13], the SDP solver performs poorly on some classes of instances (see, e.g., p_hat, san, sanr), while the subgradient algorithm still guarantees restrained CPU times, showing a more robust behaviour. This is also expressed by the times elapsed to reach $\theta_{105\%}$, $\theta_{110\%}$, which are quite competitive. This shows that the subgradient algorithm quickly computes quite good approximations of $\theta(G)$.

Overall, robustness and efficiency let the subgradient method be suitable to be embedded into a branch-and-bound algorithm to compute $\alpha(G)$ or $\chi(\bar{G})$. Recall that $\theta(G)$ turns out to be a much stronger upper bound for $\alpha(G)$ than those obtained by traditional polyhedral combinatorics techniques [6], used in previous branch-and-cut algorithms [17] [15]. In fact, several attempts have been recently made to design exact algorithms based on upper bounds close to the Lovász θ bound, either computed by SDP [20] or by LP [8] [7] [6] algorithms. The subgradient method proves to be a promising third competitor, as it allows faster reoptimization with respect to SDP algorithms and much easier implementation if compared to either LP or SDP methods.

References

[1] Burer, S & R.D.C. Monteiro (2003) *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Math. Program., Ser. B, **95**, 329–357.

[2] Choi, J., J.J. Dongarra, L.S. Ostrouchov, A.P. Petitet, D.W. Walker and R.C. Whaley, *The Design and Implementation of the ScaLAPACK LU, QR and Cholesky Factorization Routines*, 1996.

[3] DIMACS Repository
ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique.

[4] Semidefinite Programming Repository
http://www-user.tu-chemnitz.de/ helmberg/semidef.html.

[5] D'Antonio, G., and A. Frangioni, *Convergence Analysis of Deflected Conditional Approximate Subgradient Methods*, SIAM Journal on Optimization, **20/200** (1), (2009), 357–386. ISSN 1052-6234.

[6] Giandomenico, M., Rossi, F., and S., Smriglio, *Strong lift-and-project cutting*

planes for the Stable Set Problem, Math. Program. Ser. A, online first DOI: 10.1007/s10107-012-0513-3.

[7] Giandomenico, M., Letchford, A., Rossi, F., and S. Smriglio, *An Application of the Lovàsz-Schrijver $M(K, K)$ Operator to the Stable Set Problem*, Math. Program. Ser. A, **120/2** (2009), 381–401.

[8] Giandomenico, M., Letchford, A.N., Rossi, F., and S., Smriglio, *A new approach to the stable set problem based on ellipsoids*, Lecture Notes in Computer Science, LNCS 6655, Proc. of the 15th IPCO conference, (2011), NY, 223–234.

[9] Grötschel, M., Lovász, L., and A.J. Schrijver *Geometric Algorithms in Combinatorial Optimization*. New York: Wiley, 1988.

[10] Helmberg, C., and F. Rendl, *A Spectral Bundle Method for Semidefinite Programming*, SIAM Journal on Optimization, **10**, (1997), 673–696.

[11] Lovász, L., *On the Shannon capacity of a graph*, IEEE Trans. Inform. Th., IT-25, (1979), 1–7.

[12] Luz, C.J., and A.J. Schrijver, *A Convex Quadratic Characterization of the Lovász Theta Number*, SIAM J. Discrete Math., **19**(2) (2005), 382–387.

[13] Malick, J., Povh, J., Rendl, F., and A. Wiegele, *Regularization Methods for Semidefinite Programming*, SIAM J. Optimization, **20**(1) (2009), 336–356.

[14] Malick, J., and F. Roupin, *On the bridge between combinatorial optimization and nonlinear optimization: a family of semidefinite bounds for 0-1 quadratic problems leading to quasi-Newton methods*, Math. Program. Ser. B, to appear.

[15] Nemhauser, G.L., and G., Sigismondi, *A strong cutting plane/branch-and-bound algorithm for node packing*, J. Opl Res. Soc., **43** (1992), 443–457.

[16] Povh, J., Rendl, F., and A. Wiegele, *A boundary point method to solve semidefinite programs*, Computing, **78** (2006), 277–286.

[17] Rossi, F., and S., Smriglio, *A branch-and-cut algorithm for the maximum cardinality stable set problem*, Oper. Res. Lett., **28** (2001), 63–74.

[18] SDP solver `mprw2.m`, Alpen-Adria-Universität Klagenfurt website `http://www.math.uni-klu.ac.at/or/Software`.

[19] Sherali, H.D., and O., Ulular, *A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems*, Applied Mathematics and Optimization, **20** (1989), 193–221.

[20] Wilson, A.T., *Applying the boundary point method to an SDP relaxation of the maximum independent set problem for a branch and bound algorithm*, Master thesis, New Mexico Institute of mining and technology, 2009.

				SDP mprw2		Subgradient					
Graph	$\|V\|$	density	α	$\theta(G)$	Time	Z	Time	$\theta_{105\%}$	Time	$\theta_{110\%}$	Time
brock200_1	200	0.25	21	27.45	20.75	27.75	25.21	28.82	5.80	30.20	2.52
brock200_2	200	0.50	12	14.23	19.65	14.58	35.21	14.94	19.44	15.65	6.57
brock200_3	200	0.39	15	18.82	18.97	19.08	33.65	19.76	11.45	20.70	4.69
brock200_4	200	0.34	17	21.29	19.51	21.53	35.59	22.35	10.19	23.42	3.95
brock400_1	400	0.25	27	39.70	134.56	40.35	140.83	41.69	45.13	43.67	18.22
brock400_2	400	0.25	29	39.56	133.63	40.02	209.64	41.54	39.09	43.52	14.72
brock400_3	400	0.25	31	39.48	133.35	39.99	175.24	41.45	34.45	43.43	15.07
brock400_4	400	0.25	33	39.60	133.27	40.13	164.37	41.58	40.99	43.56	12.99
C.125.9	125	0.10	34	37.81	8.29	37.99	14.48	39.70	0.42	41.59	0.16
C.250.9	250	0.10	44	56.24	45.46	56.77	54.02	59.05	18.47	61.87	5.96
c-fat200-1	200	0.57	12	12.00	28.27	12.42	60.34	12.60	40.23	13.20	10.91
c-fat200-2	200	0.84	24	24.00	763.17	24.09	28.09	25.20	5.15	26.40	1.25
c-fat200-5	200	0.57	58	60.32	113.59	60.57	10.12	63.34	9.18	66.35	8.60
DSJC125.1	125	0.09	34	38.40	8.70	38.49	12.23	40.32	0.99	42.24	0.12
DSJC125.5	125	0.50	10	11.47	6.00	11.68	5.53	12.05	2.06	12.62	0.96
DSJC125.9	125	0.10	4	4.00	10.90	4.09	4.68	4.20	3.46	4.40	2.31
gen200_p0.9_44	200	0.10	44	44.00	71.40	44.95	70.82	46.20	21.86	48.40	7.09
gen200_p0.9_55	200	0.10	55	55.38	791.25	55.28	52.64	58.15	0.41	60.92	0.13
hamming6-2	64	0.10	32	32.05	2.65	32.00	0.01	33.60	< 0.01	35.20	< 0.01
hamming6-4	64	0.65	4	5.33	0.24	5.37	2.35	5.61	0.97	5.86	0.45
hamming8-2	256	0.03	128	128.00	228.81	128.00	0.03	134.40	< 0.01	140.80	< 0.01
hamming8-4	256	0.36	16	16.00	12.44	16.52	80.35	16.80	17.17	17.60	8.65
johnson8-2-4	28	0.44	4	4.00	0.03	4.00	1.1	4.20	0.32	4.40	0.04
johnson8-4-4	70	0.23	14	14.00	0.48	14.00	0.01	14.70	< 0.01	15.40	< 0.01
johnson16-2-4	120	0.24	8	8.00	1.37	8.18	14.22	8.40	9.31	8.80	4.91
keller4	171	0.35	11	14.01	10.77	14.17	22.34	14.71	8.53	15.41	3.00
p_hat300-1	300	0.76	8	10.07	119.36	10.57	142.32	10.57	142.32	11.08	79.79
p_hat300-2	300	0.51	25	26.97	423.82	27.25	171.6	28.32	22.07	29.66	10.89
p_hat300-3	300	0.26	36	41.17	204.77	41.51	157.7	43.23	20.63	45.29	5.72
p_hat500-1	500	0.75	9	13.07	416.25	14.01	568.74	13.73	***	14.38	441.85
p_hat500-2	500	0.50	36	38.98	1927.48	39.53	717.02	40.93	104.33	42.88	31.61
p_hat500-3	500	0.25	50	58.57	1136.02	59.14	940.9	61.50	166.82	64.43	44.32
p_hat700-1	700	0.75	11	15.12	1029.39	16.55	2797.81	15.88	***	16.63	2605.65
p_hat700-2	700	0.50	44	49.02	5319.12	49.78	1807.36	51.47	149.32	53.93	55.08
p_hat700-3	700	0.25	62	72.74	4277.59	74.09	778.5	76.37	174.57	80.01	57.97
san200_0.7-1	200	0.30	30	30.00	684.73	30.00	0.75	31.50	0.24	33.00	0.12
san200_0.7-2	200	0.30	18	18.00	764.01	18.44	34.72	18.90	13.84	19.80	3.23
san200_0.9-1	200	0.10	70	70.00	813.33	70.00	0.68	73.50	0.10	77.00	0.03
san200_0.9-2	200	0.10	60	60.00	798.70	60.00	0.85	63.00	0.04	66.00	0.01
san200_0.9-3	200	0.10	44	44.00	691.16	44.67	33.63	46.20	5.78	48.40	1.28
san400_0.5-1	400	0.50	13	13.00	3761.83	13.41	527.08	13.65	401.04	14.30	120.34
san400_0.7-1	400	0.30	40	40.00	4373.72	40.02	48.12	42.00	5.30	44.00	1.30
san400_0.7-2	400	0.30	30	30.00	4007.51	30.04	187.61	31.50	56.21	33.00	23.32
san400_0.7-3	400	0.30	22	22.00	449.68	22.77	879.23	23.10	600.32	24.20	457.32
san400_0.9-1	400	0.10	100	100.00	4963.48	100.50	5.71	105.00	2.10	110.00	0.97
sanr200_0.7	200	0.30	18	23.83	21.06	23.95	41.06	25.02	7.00	26.21	1.95
sanr200_0.9	200	0.10	42	49.27	28.88	49.64	57.61	51.73	10.02	54.20	4.10

Table 1
Computational results