Extracting Pure Network Submatrices in Linear Programs Using Signed Graphs. II

G. GUTIN*   A. ZVEROVITCH†

Abstract

It was shown, by N. Gulpinar, G. Gutin, G. Mitra and A. Zverovitch (to appear in Discrete Applied Mathematics), that the problem of detecting a maximum embedded network in a linear program is related to balancing of subgraphs in signed graphs. This approach led us to an efficient heuristic to extract an embedded network that, in the quality-running time experimental comparisons, outperformed three state-of-the-art heuristics. In this short paper, we consider an extension of the heuristic that was suggested to investigate in Gulpinar et al. By computational experiments, we show that a certain version of the extension gives more robust results, but still remains a very efficient heuristic.

1 Introduction, terminology and notation

Linear programming (LP) models which are of interest to applications often have sparse coefficient matrices with special structure. If a special structure can be recognized, it can be used to significantly speed up the process of solving the LP problem. A well-known family of such special structures is networks; a number of heuristics to extract (reflected) networks in LP problems have been developed and analyzed, see, e.g., [1, 3, 4, 5, 6, 7, 8]. From the computational point of view, it is worthwhile extracting a reflected network only if the LP problem under consideration contains a relatively large reflected network.

In this paper, we consider an approach that makes use of signed graphs to detect reflected networks. This approach was introduced in [7]. This approach leads to a heuristic to find reflected networks embedded within LP problems. In the quality-running time experimental comparisons, the authors of [7] showed that their heuristic outperformed

*Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, gutin@cs.rhul.ac.uk

†Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, alexei@zverovitch.com

three state-of-the-art heuristics. Moreover, their heuristic always detects when the whole matrix A is a reflected network.

One of the steps of their heuristic consists of generating a spanning forest in the underlying signed graph. In [7], the authors suggested to study an extension of the heuristic in which several forests are generated. It was not clear whether such an extension will lead to better results without a significant loss of efficiency. In this short paper, by computational experiments, we show that only a few forests are actually required to obtain more robust results and this version still remains a fast running heuristic.

We consider an LP problem in the standard form stated as

$$\text{Minimize } \{cx : \ Ax = b, \ x \geq 0\}.$$

LP problems have a number of equivalent, in a sense, forms that can be obtained from each other by various operations. Often multiplications of rows and columns of A by non-zero constants, are applied, see, e.g., [1, 3, 5, 6]. In the sequel, we assume that certain multiplications on A have been carried out and will not be applied again apart from row reflections defined below.

A matrix B is a *network (matrix)* if B is a $(0, \pm 1)$-*matrix* (that is, entries of B are in $\{1, 0, -1\}$) and every column of B has at most one entry equal to 1 and at most one entry equal to -1. The operation of *reflection* of a row of a matrix B changes the signs of all non-zero entries of this row. A matrix B is a *reflected network (matrix)* if there is a sequence of row reflections that transform B into a network matrix.

In this paper, we study the problem of finding the maximum number $\nu(A)$ of rows of A that form a reflected network matrix. This problem is known to be NP-hard [2]. In the rest of theoretical discussions, we assume, for simplicity, that A is a $(0, \pm 1)$-matrix itself (since all rows containing entries not in $\{-1, 0, +1\}$ cannot be part of a reflected network matrix).

Consider the following example. Let

$$A = \begin{pmatrix} 1 & 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \end{pmatrix}.$$

The matrix A is not a network matrix: column 1 has two entries equal 1 and column 3 has two entries equal -1. However, A is a reflected network matrix. Indeed, by reflecting row 1 we get

$$A' = \begin{pmatrix} -1 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \end{pmatrix}.$$

It is easy to verify that A' is a network matrix. Hence, $\nu(A) = 3$.

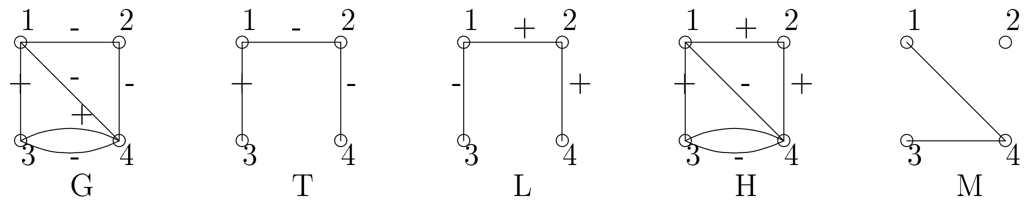

Figure 1: Example for SGA

We allow graphs to have parallel edges, but no loops. A graph $G = (V, E)$ along with a function $s: E \to \{-, +\}$ is called a *signed graph*. We assume that signed graphs have no parallel edges of the same sign. An edge is *positive* (*negative*) if it is assigned plus (minus). Let $G = (V, E, s)$ be a signed graph. For a non-empty subset W of V, the W-*switch* of G is the signed graph G^W obtained from G by changing the signs of the edges between W and $V - W$. In Figure 1, $L = T^{\{2\}} = T^{\{1,3,4\}}$, $H = G^{\{2\}}$. More information on signed graph can be found in [10, 11].

2 Embedded networks and signed graphs

For a $(0, \pm 1)$-matrix $A = [a_{ik}]$ with n rows, we construct a signed graph $G(A)$ as follows: the vertex set of $G(A)$ is $\{1, 2, ..., n\}$; $G(A)$ has a positive (negative) edge ij if and only if $a_{ik} = -a_{jk} \neq 0$ ($a_{ik} = a_{jk} \neq 0$) for some k. $G(A)$ may have parallel edges of opposite signs. For a signed graph $G = (V, E, s)$, G^- (G^+) denotes the subgraph of G whose vertices are the same as in G and whose edges are the negative (positive) edges in G. The number of negative edges in G will be denoted by $\mu(G)$.

In Figure 1, $M = H^-$ and $G = G(A)$, where

$$A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Observe that $G(A)$ has no edge between 2 and 3, and it has parallel edges between 3 and 4.

A set I of vertices of a graph H is *independent* if no two vertices in I are adjacent; $\alpha(H)$ denotes the cardinality of a maximum independent set of vertices in H, the *independence number* of H. For a signed graph $G = (V, E, s)$, define

$$\tilde{\alpha}(G) = \max\{\alpha((G^W)^-) : W \subseteq V\}.$$

The following important connection between signed graphs and reflected network matrices formed the basis for the heuristic introduced in [7].

Theorem 2.1. *[7] For a $(0, \pm 1)$-matrix A, we have $\nu(A) = \tilde{\alpha}(G(A))$.*

To use this theorem, we construct $G(A)$ for matrix A and try to find a W-switch that maximizes $\alpha((G^W)^-)$. Since the problem of computing $\nu(A)$ is NP-hard, it is computationally hard to find an optimal W-switch. Thus, we have to settle for a 'good' rather than an optimal W-switch. Since even the maximum independence set problem is NP-hard, to find a good W-switch, we use an approach that does not involve calculating the independence number in a graph. Instead, we try to find W that makes the number of negative edges in G^W relatively small. To this end, one may apply the following:

Lemma 2.2. *[7] For every signed forest T, there exists a W-switch such that $\mu(T^W) = 0$.*

Proof: We prove the lemma by induction on the number of edges in T. The lemma is true when the number of edges is one. Let x be a vertex of T of degree one. By the induction hypothesis, there is a set $W' \subseteq V(T) - x$ such that $\mu((T-x)^{W'}) = 0$. In $T^{W'}$ the edge e incident to x is positive or negative. In the first case, let $W = W'$ and the second case, let $W = W' \cup \{x\}$. Then, $\mu(T^W) = 0$. □

Clearly, the proof above implies a simple recursive procedure to find W such that $\mu(T^W) = 0$.

The heuristic SGA introduced in [7] utilizes this lemma as follows.

1. Construct signed graph $G = G(A) = (V, E, s)$.
2. Find a spanning forest T in G.
3. Using a recursive algorithm based on the proof of Lemma 2.2, compute $W \subseteq V$ such that $\mu(T^W) = 0$.
4. Using some algorithm, find a maximal independent set I in the graph $(G^W)^-$. The vertices of I correspond to the rows of A that form a reflected network.

The independent set heuristic used in [7] is the degree-greedy algorithm (see, e.g., [9]): starting with empty I, append to I a vertex of $H = (G^W)^-$ of minimum degree, delete this vertex together with its neighbors from H, and repeat the above procedure till H has no vertex.

Consider the matrix

$$A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

and show how SGA computes an approximation of $\nu(A)$ (in this example, SGA finds the exact value of $\nu(A)$). The graph $G = G(A)$ as well as its spanning tree T are depicted in Figure 1. The tree $L = T^{\{2\}}$; the graphs $H = G(A)^{\{2\}}$ and $M = H^{-}$; $\alpha(H^{-}) = 3$ and, thus, $\nu(A) \geq 3$. It is easy to see that, in fact, $\nu(A) = 3$.

Clearly, the outcome of SGA depends heavily on the choice of spanning forest T. Thus, it makes sense to generate several spanning forests rather than just one in order to find a better W-switch. However, we cannot generate too many forests since it will make SGA a slow heuristic of very restricted practical value. Thus, it is worth to investigate whether only a few forests will be enough to obtain a robust solution.

3 Computational Experiments with Several Spanning Forests

We investigated the computational performance of SGA for a number of benchmark problems which are readily available to the scientific community. We chose models from Netlib (the models can be downloaded from `http://www.netlib.org/lp/data/`) and carried out computational experiments for most of the models. In total, computational performance was measured on 100 Netlib models of various sizes.

It is difficult to make recommendations as to how to build a "good" spanning forest, that is, a forest that will ultimately lead to a larger network detected by SGA. In this subsection we attempt to get a feel for how much finding such a "good" forest could improve the performance of the algorithm.

In order to do this, we first modify the algorithm so that it builds several spanning forests instead of just one. The forests are chosen by randomly permuting the vertices of the graph and then applying the same deterministic algorithm that finds a spanning forest. Since our spanning forest algorithm examines vertices in a predetermined order, permuting vertices before applying the algorithm generally causes the algorithm to build a different forest.

For each of the test problems presented in Table **??**, we have generated 100 random permutations of the graph vertices and applied Steps 2-4 of Algorithm **??** to each of the permutations. Although permutations were chosen with equal probability, this does not necessarily mean that different forests are built with equal probability. For simplicity we also did not ensure that the set of 100 random forests contained no duplicates.

Let $N_1(\mathcal{M}), \ldots, N_{100}(\mathcal{M})$ denote the number of network rows detected in model $\mathcal{M}$ during each of the the corresponding 100 trials. We computed the statistical range

$$r(N(\mathcal{M})) = \max_{\rangle}\{\mathcal{N}_{\rangle}(\mathcal{M})\} - \min_{\rangle}\{\mathcal{N}_{\rangle}(\mathcal{M})\}$$

and the standard deviation

$$s(N(\mathcal{M})) = \sqrt{\frac{\sum_{i=1}^{100} (N_i(\mathcal{M}) - \bar{N}(\mathcal{M}))^2}{100-1}},$$

where $\bar{N}(\mathcal{M})$ denotes the mean of $N_1(\mathcal{M}), \ldots, N_{100}(\mathcal{M})$. Finally, we computed the coefficient of variation

$$cv(N(\mathcal{M})) = \frac{s(N(\mathcal{M}))}{\bar{N}(\mathcal{M})} \times 100\%.$$

The values of these statistics for the tested problems are displayed in Table 1 on pages 7–8.

It can be seen from Table 1 that in some cases the choice of the spanning forest can have a significant impact on the number of network rows detected by the SGA algorithm. For example, the number of rows detected by the algorithm in model sc205 varied by more than 30% depending on the choice of the spanning forest. However, for approximately 90% of the test models the number of rows did not vary by more than 10% on all 100 forests.

Note that these figures have been obtained with the Minimum Degree Greedy algorithm used for finding an independent set. Using a different algorithm is likely to result in different figures. If a more effective (but likely more costly) algorithm is used to find an independent set, it is possible that the results of SGA will become less sensitive to the choice of spanning forest.

If computational resources permit, an implementation of SGA can build several spanning forests. An embedded network can be constructed for each forest, and finally the algorithm can choose the largest embedded network of those found. Figure **??** shows the increase in the quality of the solutions found by SGA if several spanning forests are built and the best result taken. The results in Figure **??** correspond to the average improvement in the network size over that of an average single run (for a given problem, an average run is represented by the median of the results of the 100 random trials). The results are average over 67 Netlib problems tested and are plotted as a function of the number of random spanning forests generated.

Although the increase in the quality of the results is modest (1-2% improvement), testing several random forests instead of one may still prove worthwhile. The reason for this is that trying several forests substantially reduces the chances of hitting an extreme case that may result in a significant underperformance of the algorithm. We have seen from Table 1 that the results can vary significantly (by up to 30%) depending on the choice of the forest. However, $s(N(\mathcal{M}))$ is relatively small for most of the models. This suggests that the majority of the results are grouped close to the average; sampling even few forests can further reduce the chances of basing the results on a forest that is substantially worse than the average.

Acknowledgements The work of G. Gutin was supported in part by a grant from EPSRC.

Model	TMROW	$\bar{N}(\mathcal{M})$	$r(N(\mathcal{M}))$	$s(N(\mathcal{M}))$	$cv(N(\mathcal{M}))$
25fv47	224	200	15	3.16	1.58%
80bau3b	1374	1338	30	6.77	0.51%
agg	149	46	4	1.25	2.72%
agg2	141	59	5	1.27	2.15%
agg3	141	59	4	1.07	1.81%
bandm	100	76	2	0.81	1.07%
bnl1	275	259	3	0.89	0.34%
bnl2	1418	1335	27	5.66	0.42%
boeing1	141	88	11	1.76	2.00%
capri	112	81	7	1.50	1.85%
cre-a	1247	824	37	8.01	0.97%
cre-b	3690	2628	124	20.16	0.77%
cre-c	997	727	33	7.38	1.02%
cre-d	3040	2483	69	12.46	0.50%
cycle	505	504	0	0.00	0.00%
czprob	719	718	0	0.00	0.00%
d2q06c	844	786	21	4.32	0.55%
degen2	444	208	20	3.76	1.81%
degen3	1503	671	56	7.56	1.13%
dfl001	6019	3161	83	19.08	0.60%
etamacro	116	85	18	2.88	3.39%
fffff800	157	103	20	5.26	5.11%
finnis	263	214	8	1.98	0.93%
ganges	631	517	25	4.95	0.96%
gfrd-pnc	590	511	26	5.04	0.99%
greenbea	915	874	27	4.37	0.50%
maros	305	298	6	1.53	0.51%
modszk1	148	125	6	1.64	1.31%
nesm	190	190	0	0.00	0.00%
osa-07	1048	1041	5	1.40	0.13%
osa-14	2267	2259	5	1.57	0.07%
osa-30	4280	4279	0	0.00	0.00%
perold	150	142	1	0.86	0.61%
pilot.ja	205	196	3	0.92	0.47%
pilot	275	249	4	0.87	0.35%
pilot.we	202	202	0	0.00	0.00%
pilot4	109	109	0	0.00	0.00%
pilot87	339	303	6	1.29	0.42%
pilotnov	209	201	3	1.01	0.50%

continued on the next page...

Table 1: Statistics of 100 runs of SGA on random spanning forests

. . . continued from the previous page

Model	TMROW	$\bar{N}(\mathcal{M})$	$r(N(\mathcal{M}))$	$s(N(\mathcal{M}))$	$cv(N(\mathcal{M}))$
sc205	145	89	29	4.50	5.06%
scagr25	301	300	0	0.00	0.00%
scfxm1	141	123	8	1.42	1.16%
scfxm2	282	247	8	1.64	0.66%
scfxm3	423	371	12	2.05	0.55%
scorpion	164	164	0	0.00	0.00%
scrs8	214	213	0	0.00	0.00%
scsd8	199	199	0	0.00	0.00%
sctap1	120	120	0	0.00	0.00%
sctap2	470	470	0	0.00	0.00%
sctap3	620	620	0	0.00	0.00%
seba	402	130	10	1.94	1.49%
shell	483	479	29	6.59	1.38%
ship04l	348	310	21	3.55	1.15%
ship04s	260	222	14	3.28	1.48%
ship08l	672	606	20	3.67	0.61%
ship08s	400	333	16	3.61	1.08%
ship12l	828	728	21	5.23	0.72%
ship12s	456	355	48	6.42	1.81%
sierra	1155	741	39	6.19	0.84%
stair	160	155	3	0.95	0.61%
standata	266	186	22	3.16	1.70%
standmps	374	284	24	4.41	1.55%
stocfor2	1262	1012	45	7.59	0.75%
stocfor3	9632	7819	279	29.79	0.38%
tuff	125	114	8	1.56	1.37%
woodw	301	301	0	0.00	0.00%

Table 1 continued

References

[1] B.M. Baker and P.J. Maye, A Heuristic for Finding Embedded Network Structure in Mathematical Programmes. Europ. Jour. OR 67 (1993) 52–63.

[2] J.J. Bartholdi, A Good Submatrix is Hard to Find. Oper. Res. Letters 1 (1982) 190–193.

[3] R.E. Bixby and R. Fourer, Finding Embedded Network Rows in Linear Programs I. Extraction Heuristics. Manag. Science 34 (1988) 342–376.

[4] R.E. Bixby, and W.H. Cunningham, Converting Linear Programs to Network Problems. Math. of OR 5 (1980) 321–356.

[5] G.G. Brown and W.G. Wright, Automatic Identification of Embedded Network Rows in Large-Scale Optimization Models. Math. Prog. 29, (1984) 41–56.

[6] N. Gülpınar, G. Gutin, G. Mitra and I. Maros, Detecting Embedded Pure Network Structures by Using GUB and Independent Set Algorithms. Comput. Optim. Applic. 15 (2000) 235–247.

[7] N. Gülpınar, G. Gutin, G. Mitra and A. Zverovitch, Extracting Pure Network Submatrices in Linear Programs Using Signed Graphs. To appear in Discrete Appl. Math.

[8] A.C. Hsu and R. Fourer, Identification of Embedded Network Structure in Linear Programming Models. GSIA Working Paper, 1997-58.

[9] V.Th. Paschos, A $\Delta/2$-Approximation for the Maximum Independent Set Problem. Inform. Proc. Let. 44 (1992) 11–13.

[10] T. Zaslavsky, Bibliography of Signed and Gain Graphs. Electronic J. Combin., Dynamic Surveys DS8, 151 pp. `http://www.combinatorics.org/index.html`

[11] T. Zaslavsky, Glossary of Signed and Gain Graphs. Electronic J. Combin., Dynamic Surveys DS9, 41 pp. `http://www.combinatorics.org/index.html`