Fixed-Parameter Algorithms in Analysis of Heuristics for Extracting Networks in Linear Programs*

Gregory Gutin† Daniel Karapetyan‡ Igor Razgon§

August 10, 2009

Abstract

A parameterized problem Π can be considered as a set of pairs (I, k) where I is the main part and k (usually an integer) is the parameter. Π is called fixed-parameter tractable (FPT) if membership of (I, k) in Π can be decided in time $O(f(k)|I|^c)$, where $|I|$ denotes the size of I, $f(k)$ is a computable function, and c is a constant independent of k and I. An algorithm of complexity $O(f(k)|I|^c)$ is called a fixed-parameter algorithm.

It often happens that although a problem is FPT, the practitioners prefer to use imprecise heuristic methods to solve the problem in the real-world situation simply because of the fact that the heuristic methods are faster. In this paper we argue that in this situation a fixed-parameter algorithm for the given problem may be still of a considerable practical use. In particular, the fixed-parameter algorithm can be used to evaluate the approximation quality of heuristic approaches.

To demonstrate this way of application of fixed-parameter algorithms, we consider the problem of extracting a maximum-size reflected network in a linear program. We evaluate a state-of-the-art heuristic SGA and two variations of it with a new heuristic and with an exact algorithm. The new heuristic and algorithm use fixed-parameter tractable procedures. The new heuristic turned out to be of little practical interest, but the exact algorithm is of interest when the network size is close to that of the linear program especially if the exact algorithm is used in conjunction with SGA. Another conclusion which has a large practical interest is that some variant of SGA can be the best choice because in most cases it returns optimal solutions; previously it was disregarded because comparing to the other heuristics it improved the solution insignificantly at the cost of much larger running times.

*A preliminary version of this paper will appear in the Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC'09).

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

‡Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, England, UK, daniel.karapetyan@gmail.com

§Department of Computer Science, University College Cork, Ireland, i.razgon@cs.ucc.ie

1 Introduction, terminology and notation

When a hard optimization problem is to be solved heuristically, it is often difficult to choose which heuristic to use as it rarely happens that one heuristic is both faster and more precise than another one. Often there is a tradeoff: a heuristic providing a more precise solution takes more time than a heuristic of lesser quality. In this case, the slower heuristic may be preferred if it turns out that the solution it returns is usually much closer to the optimal solution. However, to evaluate the quality of the given solution we need a method that can find an optimal solution (even if finding a *provably* optimal solution takes much more time than the heuristic being analyzed). If the considered problem involves a small parameter, solving the problem to optimality can be done by a fixed-parameter algorithm. (We give a short introduction into fixed-parameter algorithmics in Section 4.) Thus, even if a fixed-parameter algorithm is not *directly* used within a problem solving software, it can still be very useful for *testing* different versions of this software. In this paper we consider a problem occurring in large-scale linear programming (LP) to demonstrate the use of fixed-parameter algorithms in this novel way.

Large-scale LP models which arise in applications usually have sparse coefficient matrices with special structure. If a special structure can be recognized, it can often be used to considerably speed up the process of solving the LP problem and/or to help in understanding the nature of the LP model. 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., [3, 6, 7, 8, 13, 14, 19] (a formal definition of a reflected network is given below). 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.

We consider an LP problem in the standard form stated as

$$\text{Minimize } \{p^T x;\ \text{subject to } Ax = b,\ x \geq 0\}.$$

LP problems have a number of equivalent, in a sense, forms that can be obtained from each other by various operations. Often scaling operations, that is multiplications of rows and columns of the matrix A of constraints by non-zero constants, are applied, see, e.g., [3, 6, 8, 13]. In the sequel unless stated otherwise, we assume that certain scaling operations 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 belong to the set $\{1, 0, -1\}$) 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 transforms B into a network matrix. The *problem of detecting a maximum embedded reflected network* (DMERN) is to find the maximum number of rows that form a submatrix B of A such that B is a reflected network. This number is denoted by $\nu(A)$. The DMERN problem is known to be NP-hard [4].

Gülpınar et al. [14] showed that the maximum size of an embedded reflected network equals the maximum order of a balanced induced subgraph of a special signed graph associated with matrix A (for details, see Section 2). This result led Gülpınar et

al. [14] to a heuristic named SGA for detection of reflected networks. Computational experiments in [14] with SGA and three other heuristics demonstrated that SGA and another heuristic, RSD, were of very similar quality and clearly outperformed the two other heuristics in this respect. However, SGA was about 20 times faster, on average, than RSD. Moreover, SGA has an important theoretical property that RSD does not have: SGA always solves the DMERN problem to optimality when the whole matrix A is a reflected network [14]. Since SGA appeared to be the best choice for a heuristic for detection of reflected networks, Gutin and Zverovitch [15] investigated 'repetition' versions of SGA and found out that three times repetition of SGA (SGA3) gives about 1% improvement, while 80 times repetition of SGA (SGA80) leads to 2% improvement. Thus, at the first glance it might seem that SGA3 and, of course, SGA80 heuristics are not of any practical interest because, taking more time, they produce a very little improvement of the solution quality.

In this paper we argue that in fact SGA80 can be viewed as *the best choice* because, being reasonably fast, in most cases it produces an *optimal solution* to the problem under consideration. To solve the DMERN problem to optimality we design a fixed-parameter algorithm for the maximum balanced subgraph problem and we compare the output of the heuristics being analyzed against the output of the algorithm. To design the FPT algorithm we reduce the maximum balanced subgraph problem to the bipartization problem and then use a fixed-parameter algorithm for the latter problem [20, 25]. Thus it turns out that although the fixed-parameter algorithm we use is usually much slower than the heuristic methods, it *helps* to select the best heuristic for the DMERN problem.

As an additional contribution, we investigated another modification of SGA where the use of a greedy-type independent set extracting heuristic (which is part of SGA) is replaced by a fixed-parameter algorithm for finding a minimum vertex cover. Here we used the well-known fact that the complement of an independent set in a graph is a vertex cover. Our experiments with this modification of SGA showed very little improvement and, thus, this modification of SGA appears to be of little practical interest. However, this demonstrated that the independent set extracting heuristic need not be replaced by a more powerful heuristic or exact algorithm.

The rest of the paper is organized as follows. In Section 2 we introduce necessary notation, Section 3 presents the SGA heuristic and its variants, and Section 4 introduces the fixed-parameter algorithms. Section 5 describes a fixed-parameter algorithm for the maximum balanced subgraph problem. In Section 6 we report empirical results and analyze them. Concluding remarks are made in Section 7.

2 Embedded networks and signed graphs

In this section, we assume, for simplicity, that A is a $(0, \pm 1)$-matrix itself (since all rows containing entries not from the set $\{-1, 0, +1\}$ cannot be part of a reflected network). Here we allow graphs to have parallel edges, but no loops. A graph $G = (V, E)$ along with a function $s : E \rightarrow \{-, +\}$ is called a *signed graph*. Signed graphs have been studied by many researchers, see, e.g., [16, 17, 18, 26].

We assume that signed graphs have no parallel edges of the same sign, but may

have parallel edges of opposite signs. An edge is *positive* (*negative*) if it is assigned plus (minus). 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 $\{1, 2, \ldots, n\}$; $G(A)$ 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. 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 $V(G) \setminus W$. A signed graph $G = (V, E, s)$ is *balanced* if there exists a subset W of V (W may coincide with V) such that G^W has no negative edges. Let $\eta(G)$ be the largest order of a balanced induced subgraph of G.

The following important result was proved in [14]. This result allows us to search for a largest balanced induced subgraph of $G(A)$ instead of a largest reflected network in A.

Theorem 1. *[14] Let A be a $(0, \pm 1)$-matrix. A set R of rows in A forms a reflected network if and only if the vertices of $G(A)$ corresponding to R induce a balanced subgraph of $G(A)$. In particular, $\nu(A) = \eta(G(A))$.*

3 SGA and its Variations

The heuristic SGA introduced in [14] is based on the following:

Lemma 1. *[14] Every signed tree T is a balanced graph.*

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 $(T - x)^W$ has no negative edges. 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, $T^{W'}$ has no negative edges. □

Heuristic SGA:
Step 1: Construct signed graph $G = G(A) = (V, E, s)$.
Step 2: Find a spanning forest T in G.
Step 3: Using a recursive algorithm based on the proof of Lemma 1, compute $W \subseteq V$ such that T^W has no negative edges.
Step 4: Let N be the subgraph of G^W induced by the negative edges. Apply the following greedy-degree algorithm [23] to find a maximal independent set I in N: starting from empty I, append to I a vertex of N of minimum degree, delete this vertex together with its neighbors from N, and repeat the above procedure till N has no vertex.
Step 5: Output I.

Proposition 1. *[14] If G is balanced, then $I = V$.*

Proof. It is well-known (see, e.g., Theorem 2.8 in [14]) that a signed graph is balanced if and only if it does not contain cycles with odd number of negative edges. Let T be

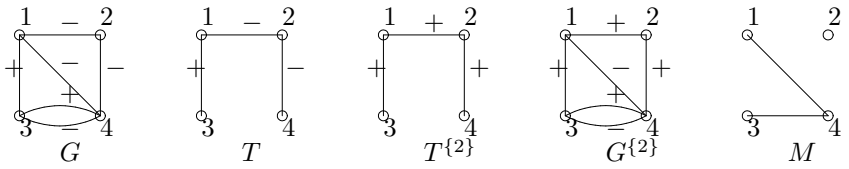

Figure 1: Illustration for SGA; M is the subgraph $G^{\{2\}}$ induced by the negative edges of $G^{\{2\}}$.

a a spanning forest in G. Since T^W has no negative edges, G^W cannot have negative edges. Indeed, if xy was a negative edge in G^W, it would be the unique negative edge in a cycle formed by xy and the (x, y)-path of T^W, a contradiction. □

Gutin and Zverovitch [15] investigated a repetition version of SGA where Steps 2-4 were repeated several time (each time the vertices of G were pseudo-randomly permuted and a new spanning forest of G was built). They found out that three times repetition of SGA gives about 1% improvement, while 80 times repetition of SGA leads to 2% improvement, on average. In our experiments we used a larger text bed and better scaling procedure than in [15] and, thus, we run SGA and its 3 and 80 times repetitions on the new set of instances of the DMERN problem (see Section 6). We will denote these repetition versions of SGA by SGA3 and SGA80, respectively.

In Section 6 we also report results on another modification of SGA, SGA+VC, where we replace Step 4 with finding a vertex cover C of G^W and setting $I = V(G^W) \setminus C$. Since the vertex cover problem is well studied in the area of parameterized complexity [1, 9, 22], to find C we can use a fixed-parameter algorithm for the problem.

4 Fixed-Parameter Algorithmics

We recall some most basic notions of fixed-parameter algorithmics (FPA) here, for a more in-depth treatment of the topic we refer the reader to the monographs [11, 12, 22].

FPA is a relatively new approach for dealing with intractable computational problems. In the framework of FPA we introduce a parameter k, which is often a positive integer (but may be a vector, graph, or any other object for some problems) such that the problem at hand can be solved in time $O(f(k)n^c)$, where n is the size of the problem instance, c is a constant not dependent on n or k, and $f(k)$ is an arbitrary computable function not dependent on n. The ultimate goal is to obtain $f(k)$ and c such that for small or even moderate values of k the problem under consideration can be completely solved in a reasonable amount of time.

As an example, consider the *Vertex Cover problem (VC)*: given an undirected graph G (with n vertices and m edges), find a minimum number of vertices such that every edge is incident to at least one of these vertices. In the (naturally) parameterized version of VC, k-VC, given a graph G, we are to check whether G has a vertex cover with at most k vertices. k-VC admits an algorithm of running time $O(1.2738^k + kn)$ obtained in [9] that allows us to solve VC with k up to several hundreds. Without using FPA, we

would be likely to end up with the obvious algorithm of complexity $O(mn^k)$. The last algorithm is far too slow even for small values of k such as $k = 10$.

Parameterized problems that admit algorithms of complexity $O(f(k)n^c)$ (we refer to such algorithms as *fixed-parameter*) are called *fixed-parameter tractable (FPT)*. Notice that not every parameterized problem is FPT, but there are many problems that are FPT. A parameterized problem is FPT if and only if it admits kernelization [11, 12, 22], which is defined as follows. For a parameterized decision problem Π given by pairs (I, k), where I is an instance of Π and k is the parameter, a *kernelization* is a polynomial time (in the size of I and k) reduction $(I, k) \mapsto (I', k')$ such that I is a Yes-instance if and only if I' is a Yes-instance, the size of I' is bounded (from above) by a function $g(k)$ depending on k only and $k' \leq k$. The instances (I', k') comprise a *kernel* of Π of *size* $g(k)$.

5 Minimum Balanced Deletion problem

By our discussions above, we are interested in the following parameterized problem.

The minimum balanced deletion problem (MBD)
Input: A signed graph $G = (V, E, s)$, an integer k.
Parameter: k.
Output: A set of at most k vertices whose removal makes G balanced or 'NO' if no such set exists.

We show that the MBD problem is FPT by transforming it into the Bipartization problem defined as follows.

The Bipartization problem
Input: A graph G, an integer k
Parameter: k
Output: A set of at most k vertices whose removal makes G bipartite or 'NO' if no such set exists.

The transformation is described in the following theorem.

Theorem 2. *The MBD problem is FPT and can be solved in time* $O^*(3^k)$.

Proof. It is well-known (see, e.g., Theorem 2.8 in [14]) that a signed graph is balanced if and only if it does not contain cycles involving odd number of negative edges. Hence, the MBD problem in fact asks for at most k vertices whose removal breaks all cycles containing an odd number of negative edges.

Let G' be the (unsigned) graph obtained from G by *subdividing* each positive edge. In other words, for each positive edge $\{u, v\}$, we introduce a new vertex w and replace $\{u, v\}$ by $\{u, w\}$ and $\{w, v\}$. We claim that G has a set of at most k vertices breaking all cycles with an odd number of negative edges if and only if G' can be made bipartite by removal of at most k vertices.

Assume the former and let K be a set of at most k vertices whose removal breaks all cycles with an odd number of negative edges. It follows that $G' - K$ is bipartite. Indeed,

each cycle C' of $G' - K$ can be obtained from a cycle C of $G - K$ by subdivision of its positive edges. Hence, C' can be of an odd length only if C has an odd number of negative edges which is impossible according to our assumption about K.

Conversely, let K be a set of at most k vertices such that $G' - K$ is bipartite. We may safely assume that K does not contain the new vertices subdividing positive edges: otherwise each such vertex can be replaced by one of its neighbors. Thus, $K \subseteq V(G)$. Observe that $G - K$ does not have cycles with odd number of negative edges. Indeed, by subdividing positive edges, any such cycle translates into an odd cycle of $G' - K$ in contradiction to our assumption about K.

It follows from the above argumentation that the MBD problem can be solved as follows. Transform G into G' and run on G' the $O^*(3^k)$ algorithm solving the bipartization problem [20]. If the algorithm returns 'NO' then return 'NO'. Otherwise, replace each subdividing vertex by one of its neighbors and return the resulting set of vertices. Clearly, the complexity of the resulting algorithm is $O^*(3^k)$. □

Remarks. The usual trick to avoid the subdivided vertices to be selected to the resulting solution would be to make $k + 1$ copies of each vertex. However, such approach would increase the runtime of the resulting implementation and hence we have used a slightly more sophisticated method. Unfortunately, it is not known yet whether the Bipartization problem has a polynomial-size problem kernel [20]. Thus, it is not known yet whether the MBD problem has a polynomial-size problem kernel. (If one was known, we could try to use it to speed up our fixed-parameter algorithm.)

Note that a version of the MBD problem, where edge-deletions rather than vertex-deletions are used was considered in [5, 10].

6 Experimental Evaluation

In this section we provide and discuss our experiment results for the heuristics SGA, SGA3, SGA80, SGA+VC descried in Section 3 and the exact algorithm given in Section 5. Note that in our experiments we use a larger test bed and better scaling procedure than in [15].

Recall that we consider an LP problem in the standard form stated as

$$\text{Minimize } \{p^T x;\ \text{subject to } Ax = b,\ x \geq 0\}.$$

In Section 2, to simplify our notation we assumed that A is a $(0, \pm 1)$-matrix. However, in general, in real LP problems A is not a $(0, \pm 1)$-matrix. Therefore, in reality, the first phase in solving the DMERN problem is applying a scaling procedure whose aim is to increase the number of $(0, \pm 1)$-rows by scaling rows and columns. Here we describe a scaling procedure that we have used. Our computational experiments indicate that this scaling is often better than the scaling procedures we found in the literature. Let us describe our scaling procedure. Let $A = [a_{ij}]_{n \times m}$.

First we apply simple row scaling, i.e., scale all the rows which contain only zeros and $\pm x$, where $x > 0$ is some constant: for every $i \in \{1, 2, \ldots, n\}$ set $a_{ij} = a_{ij}/x$ for $j = 1, 2, \ldots, m$ if $a_{ij} \in \{0, -x, +x\}$ for every $j \in \{1, 2, \ldots, m\}$.

Then we apply a more sophisticated procedure. Let $[r_i]_n$ be an array of boolean values, where r_i indicates whether the ith row is a $(0, \pm 1)$-row. Let $[b_j]_m$ be an array of boolean values, where b_j indicates whether the jth column is bounded, i.e., whether it has at least one nonzero value in a $(0, \pm 1)$-row: for some $j \in \{1, 2, \ldots, m\}$ the value $b_j = true$ if and only if there exists some i such that $r_i = true$ and $a_{ij} \neq 0$.

Next we do the following for every non $(0, \pm 1)$-row (note that at this stage any non $(0, \pm 1)$-row contains at least two nonzero elements). Let J be the set of indices of bounded columns with nonzero elements in the current row c: $J = \{j : a_{cj} \neq 0 \text{ and } b_j = true\}$. If $J = \emptyset$, i.e., all the columns corresponding to nonzero elements in the current row are unbounded, then we simply scale every of these columns: $a_{ij} = a_{ij}/a_{cj}$ for every $i = 1, 2, \ldots, n$ and for every j such that $a_{cj} \neq 0$. If $J \neq \emptyset$ and $a_{cj} \in \{+x, -x\}$ for every $j \in J$, where x is some constant, then we scale accordingly the current row ($a_{cj} = a_{cj}/x$ for every $j \in \{1, 2, \ldots, m\}$) and scale the unbounded columns: $a_{ij} = a_{ij}/a_{cj}$ for every $j \notin J$ if $a_{cj} \neq 0$. Otherwise we do nothing for the current row.

Every time when we scale rows or columns we update the arrays r and b.

Since the matrices processed by this heuristic are usually sparse, we use a special data structure to store them. In particular, we store only nonzero elements providing the row and column indices for each of them. We also store a list of references to the corresponding nonzero elements for every row and for every column of the matrix.

The computational results for all heuristics apart from SGA+VC as well as for the exact algorithm are provided in Table 1. As a test bed we use all the instances provided in Netlib (`http://netlib.org/lp/data/`). In the table, n denotes the number of $(0, \pm 1)$-rows in the instance, i.e., the number of vertices in the corresponding signed graph G. Also $k_{\min}, k_1, k_3, k_{80}$ denote the values of the difference between n and the number of vertices in a maximum induced balanced subgraph of G found by the exact algorithm and SGA, SGA3 and SGA80, respectively, and t, t_1, t_3, t_{80} stand for the running time (in seconds) of the exact algorithm and SGA, SGA3 and SGA80, respectively. When the exact algorithm could not produce a solution after 1 hour, it was terminated. The average values of k are given over all the instances for SGA, SGA3 and SGA80. We also provide the averages for the instances solved by the exact algorithm (see the optimal set average row).

All algorithms were implemented in C++ and the evaluation platform is based on an AMD Athlon 64 X2 3.0 GHz processor. For the exact algorithm we used a code of Hüffner `http://theinf1.informatik.uni-jena.de/~hueffner/`. In SGA+VC we used a vertex cover code based on [2].

Table 1: Experiment results for the SGA, SGA3 and SGA80 heuristics and for the exact algorithm.

Instance	n	$k_{\min}$	k_1	k_3	k_{80}	t	t_1	t_3	t_{80}
25FV47	283	15	25	25	22	4.40	0.02	0.03	0.39
80BAU3B	1629	—	42	40	40	$>$ 1h	0.08	0.25	9.75
ADLITTLE	31	1	1	1	1	0.02	0.00	0.00	0.00

Instance	n	$k_{\min}$	k_1	k_3	k_{80}	t	t_1	t_3	t_{80}
AFIRO	16	0	0	0	0	0.00	0.00	0.00	0.00
AGG	159	—	107	104	104	$>$ 1h	0.02	0.00	0.09
AGG2	153	—	85	85	83	$>$ 1h	0.00	0.00	0.03
AGG3	153	—	85	85	83	$>$ 1h	0.02	0.00	0.08
BANDM	143	23	24	24	23	1493.12	0.00	0.00	0.08
BEACONFD	118	3	3	3	3	0.00	0.00	0.00	0.02
BLEND	24	1	1	1	1	0.00	0.00	0.00	0.00
BNL1	315	14	17	17	14	1.83	0.00	0.02	0.19
BNL2	1549	—	127	110	99	$>$ 1h	0.05	0.17	4.96
BOEING1	145	—	49	49	48	$>$ 1h	0.00	0.00	0.03
BOEING2	79	15	17	17	15	0.05	0.00	0.02	0.02
BORE3D	131	12	14	13	12	0.14	0.00	0.00	0.03
BRANDY	122	6	7	6	6	0.00	0.00	0.00	0.05
CAPRI	126	—	40	37	34	$>$ 1h	0.00	0.00	0.05
CYCLE	700	—	34	34	34	$>$ 1h	0.02	0.06	1.64
CZPROB	912	1	1	1	1	0.27	0.02	0.03	1.73
D2Q06C	980	—	67	67	67	$>$ 1h	0.02	0.11	3.56
D6CUBE	122	—	61	52	46	$>$ 1h	0.02	0.00	0.16
DEGEN2	444	—	234	233	226	$>$ 1h	0.02	0.03	0.83
DEGEN3	1503	—	822	819	813	$>$ 1h	0.17	0.53	16.91
DFL001	6022	—	2818	2818	2802	$>$ 1h	1.53	5.87	166.05
E226	100	15	18	17	16	1.09	0.00	0.00	0.03
ETAMACRO	145	12	20	20	20	0.47	0.00	0.00	0.09
FFFFF800	178	—	50	41	41	$>$ 1h	0.00	0.02	0.14
FINNIS	325	—	121	120	119	$>$ 1h	0.00	0.02	0.31
FIT1D	10	6	6	6	6	0.00	0.00	0.00	0.00
FIT1P	1	0	0	0	0	0.00	0.00	0.00	0.03
FIT2D	10	6	7	6	6	0.00	0.00	0.02	0.33
FIT2P	4	2	2	2	2	0.00	0.03	0.02	0.76
FORPLAN	61	1	1	1	1	0.00	0.00	0.00	0.05
GANGES	822	—	83	83	77	$>$ 1h	0.03	0.05	1.89
GFRD-PNC	616	—	68	68	68	$>$ 1h	0.03	0.05	0.78
GREENBEA	970	—	48	48	45	$>$ 1h	0.06	0.12	3.46
GREENBEB	970	—	48	48	45	$>$ 1h	0.03	0.17	3.42
GROW15	15	0	0	0	0	0.00	0.00	0.00	0.02
GROW22	22	0	0	0	0	0.00	0.00	0.00	0.02
GROW7	7	0	0	0	0	0.00	0.00	0.02	0.02
ISRAEL	30	8	9	9	8	0.02	0.00	0.00	0.00
KB2	15	1	1	1	1	0.00	0.00	0.00	0.00
LOTFI	105	18	24	22	19	11.23	0.00	0.00	0.02
MAROS-R7	50	0	0	0	0	0.00	0.05	0.05	0.83
MAROS	340	11	17	15	11	0.23	0.00	0.03	0.33
MODSZK1	374	—	237	237	237	$>$ 1h	0.02	0.02	0.30
NESM	232	10	13	11	10	0.03	0.02	0.03	0.19

Instance	n	$k_{\min}$	k_1	k_3	k_{80}	t	t_1	t_3	t_{80}
PEROLD	235	—	28	25	24	$>$ 1h	0.00	0.00	0.12
PILOT.JA	318	16	18	16	16	11.72	0.00	0.02	0.31
PILOT	337	—	45	42	41	$>$ 1h	0.00	0.03	0.70
PILOT.WE	295	—	34	29	28	$>$ 1h	0.00	0.02	0.34
PILOT4	151	3	3	3	3	0.00	0.02	0.02	0.08
PILOT87	479	—	77	76	70	$>$ 1h	0.03	0.02	1.25
PILOTNOV	329	19	21	21	19	201.29	0.03	0.00	0.59
RECIPE	61	0	0	0	0	0.00	0.00	0.00	0.02
SC105	75	16	17	17	17	12.56	0.00	0.00	0.02
SC205	148	—	36	36	36	$>$ 1h	0.00	0.02	0.03
SC50A	35	8	8	8	8	0.02	0.00	0.02	0.00
SC50B	33	6	6	6	6	0.02	0.00	0.00	0.00
SCAGR25	299	0	0	0	0	0.03	0.00	0.02	0.19
SCAGR7	83	0	0	0	0	0.02	0.00	0.00	0.02
SCFXM1	154	12	13	12	12	0.30	0.00	0.00	0.08
SCFXM2	308	—	26	26	24	$>$ 1h	0.00	0.00	0.22
SCFXM3	462	—	39	38	36	$>$ 1h	0.02	0.02	0.56
SCORPION	214	1	1	1	1	0.02	0.00	0.00	0.14
SCRS8	281	9	9	9	9	0.06	0.00	0.02	0.19
SCSD1	39	0	0	0	0	0.03	0.00	0.02	0.02
SCSD6	74	0	0	0	0	0.14	0.00	0.02	0.02
SCSD8	199	0	0	0	0	0.59	0.00	0.00	0.09
SCTAP1	120	0	0	0	0	0.00	0.00	0.00	0.03
SCTAP2	470	0	0	0	0	0.00	0.00	0.03	0.53
SCTAP3	620	0	0	0	0	0.00	0.00	0.06	0.92
SEBA	408	—	274	271	269	$>$ 1h	0.00	0.05	1.95
SHARE1B	49	4	5	4	4	0.00	0.00	0.00	0.02
SHARE2B	36	6	6	6	6	0.00	0.02	0.00	0.00
SHELL	536	2	2	2	2	1.51	0.00	0.02	0.51
SHIP04L	394	—	36	36	36	$>$ 1h	0.02	0.05	0.53
SHIP04S	394	—	36	36	36	$>$ 1h	0.00	0.02	0.41
SHIP08L	762	—	64	64	64	$>$ 1h	0.05	0.08	1.87
SHIP08S	762	—	64	64	64	$>$ 1h	0.02	0.03	1.31
SHIP12L	1141	—	96	96	96	$>$ 1h	0.03	0.14	3.71
SHIP12S	1141	—	96	96	96	$>$ 1h	0.03	0.12	2.76
SIERRA	1161	—	400	399	387	$>$ 1h	0.02	0.05	2.71
STAIR	82	8	11	10	8	0.02	0.00	0.02	0.03
STANDATA	245	—	53	53	53	$>$ 1h	0.00	0.00	0.17
STANDGUB	247	—	53	53	53	$>$ 1h	0.02	0.02	0.19
STANDMPS	353	—	54	54	54	$>$ 1h	0.00	0.02	0.31
STOCFOR1	56	0	0	0	0	0.00	0.00	0.00	0.00
STOCFOR2	1306	—	258	258	243	$>$ 1h	0.05	0.12	3.88
TUFF	175	16	26	17	16	0.58	0.02	0.00	0.11
VTP.BASE	30	4	6	4	4	0.00	0.00	0.00	0.00

Instance	n	$k_{\min}$	k_1	k_3	k_{80}	t	t_1	t_3	t_{80}
WOOD1P	74	0	0	0	0	0.02	0.00	0.00	0.11
WOODW	329	0	0	0	0	0.00	0.00	0.05	0.78
Total average		—	79.3	78.3	76.9	—	0.03	0.10	2.66
Optimal set average		5.8	7.0	6.6	6.1	32.26	0.00	0.01	0.19

The results with SGA+VC are not provided since SGA+VC managed to improve SGA only for four instances: D6CUBE ($k_{SGA+VC} = 59$), DEGEN2 ($k_{SGA+VC} = 230$), DEGEN3 ($k_{SGA+VC} = 806$), and DFL001 ($k_{SGA+VC} = 2809$). Note that in the three of these instances $k_{80} < k_{SGA+VC}$. Since the running time of SGA+VC usually exceeds that of SGA80 and the quality of SGA+VC is not much different even from that of SGA, SGA+VC appears to be of little practical interest. However, SGA+VC demonstrates that there is no need to replace Step 4 of SGA by a more powerful heuristic or exact algorithm.

Observe that the exact algorithm completed its computations for 54 instances out of the total of 93, and for 52 instances the running time was at most 1 minute. Note that SGA achieved the optimal solution in 33 out of 54 cases, SGA3 in 39 cases and SGA80 in 49 cases. Observe that in almost all the cases feasible for the exact algorithm, SGA80, being much faster than the exact algorithm, managed to compute an optimal solution! Although SGA80 is slower than SGA and SGA3, it is much more precise and its running time is still reasonable. It follows that SGA80 is the best choice with respect to the tradeoff between running time and precision. Note that this conclusion can be made only *given the knowledge* about the optimal solution and without the considered fixed-parameter algorithm such knowledge would be very hard to obtain (for example, the instance PILOTNOV with $n = 329$ and $k_{\min} = 19$ would hardly be feasible to a brute-force exploration of all $\binom{329}{19}$ possibilities).

7 Conclusions

In this paper we have demonstrated a novel way of use of fixed-parameter algorithms where they do not substitute heuristic methods but are used to evaluate them. As a case study, we considered heuristics for the problem of extracting a maximum-size reflected network in an LP problem. The main conclusion of our empirical study is that the slowest heuristic, which provided only a minor improvement over the other ones and was basically disregarded due to this fact, has turned out to be the best among the heuristics under consideration because, being reasonably fast, it often produces an optimal solution. The fixed-parameter algorithm in this case has helped us to check whether a solution returned by the heuristic being analyzed is indeed optimal.

We believe that this way of applying fixed-parameter algorithms can be useful for other problems as well. One candidate might be the problem of finding whether the given CNF formula has at most k variables so that their removal makes the resulting formula Renameable Horn. This is called the Renameable Horn deletion backdoor problem and was recently shown FPT [24]. Heuristics for this problem are widely used in modern SAT solvers for identifying a small subset of variables on which an

exponential-time branching is to be performed [21]. Currently it is unclear whether substituting a heuristic approach by the exact fixed-parameter algorithm would result in a better SAT solver. But even if it is not the case, the exact algorithm can be still of a considerable use for ranking the heuristic techniques, especially as producing small Renameable Horn backdoors is vitally important for reducing the exponential-time impact on the runtime of SAT solvers.

Acknowledgements We are grateful to Michael Langston and his group for providing us with a vertex cover code. The work of G. Gutin was supported in part by a grant from EPSRC. The work of I. Razgon was supported by Science Foundation Ireland grant 05/IN/I886.

References

[1] F.N. Abu-Khzam, M.R. Fellows, M. Langston and W.H. Suters, Crown Structures for Vertex Cover Kernelization, Theory of Computing Systems 41 (2007), 411-430.

[2] F.N. Abu-Khzam, M. Langston, P. Shanbhag and C.T. Symons, Scalable Parallel Algorithms for FPT Problems. Algorithmica 45 (2006) 269–284.

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

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

[5] N. Betzler, F. Hüffner and R. Niedermeier, Optimal edge deletions for signed graph balancing. In Proc. WEA'07 , LNCS vol. 4525, pages 297310, Springer, 2007.

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

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

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

[9] J. Chen, I.A. Kanj, and G. Xia, Simplicity is beauty: Improved upper bounds for vertex cover. Tech. Report TR05-008, DePaul University, Chicago IL, 2005.

[10] B. DasGupta, G. A. Enciso, E. D. Sontag, and Y. Zhang. Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. In Proc. 5th WEA, LNCS, vol. 4007, pages 253264, Springer, 2006.

[11] R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer, 1999.

[12] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer, 2006.

[13] 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.

[14] N. Gülpınar, G. Gutin, G. Mitra and A. Zverovitch, Extracting Pure Network Submatrices in Linear Programs Using Signed Graphs. Discrete Applied Mathematics 137 (2004) 359–372.

[15] G. Gutin and A. Zverovitch, Extracting pure network submatrices in linear programs using signed graphs, Part 2. Communications of DQM 6 (2003) 58–65.

[16] P. Hansen, Labelling Algorithms for Balance in Signed Graphs, In: *Problémes Combinatoires et Theorie des Graphes* (Colloq. Internat., Orsay, 1976), 215–217. Colloques Internat. du CNRS 260 Paris (1978).

[17] F. Harary and J.A. Kabell, A Simple Algorithm to Detect Balance in Signed Graphs. Math. Social Science 1 (1980/81) 131–136.

[18] I. Heller and C.B. Tompkins, An Extension of a Theorem of Dantzig's, In: H.W.Kuhn and A.W.Tucker, eds., *Linear Inequalities and Related Systems* 247–252. Annals Math. Studies, 38, Princeton Univ. Press, Princeton, N.J. (1956).

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

[20] F. Hüffner, Algorithm Engineering for Optimal Graph Bipartization. Journal of Graph Algorithms and Applications 13 (2009), 77–98.

[21] S. Kottler, M. Kaufmann, and C. Sinz, Computation of Renameable Horn Backdoors. SAT 2008, pp. 154-160.

[22] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, 2006.

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

[24] I. Razgon and B. O'Sullivan, Almost 2-SAT is fixed-parameter tractable. Journal of Computer and System Sciences, to appear.

[25] B. Reed, K. Smith, and A. Vetta, Finding odd cycle transversals. Operations Research Letters 32 (2004), 299-301.

[26] T. Zaslavsky, Signed Graphs. Discete Applied Math. 4 (1982) 47–74.