Journal of Combinatorial Optimization manuscript No.
(will be inserted by the editor)

Separator-Based Data Reduction for Signed Graph Balancing

Falk Hüffner · **Nadja Betzler** · **Rolf Niedermeier**

the date of receipt and acceptance should be inserted later

Abstract Polynomial-time data reduction is a classical approach to hard graph problems. Typically, particular small subgraphs are replaced by smaller gadgets. We generalize this approach to handle any small subgraph that has a small separator connecting it to the rest of the graph. The problem we study is the NP-hard BALANCED SUBGRAPH problem, which asks for a 2-coloring of a graph that minimizes the inconsistencies with given edge labels. It has applications in social networks, systems biology, and integrated circuit design. The data reduction scheme unifies and generalizes a number of previously known data reductions, and can be applied to a large number of graph problems where a coloring or a subset of the vertices is sought. To solve the instances that remain after reduction, we use a fixed-parameter algorithm based on iterative compression with a very effective heuristic speedup. Our implementation can solve biological real-world instances exactly for which previously only approximations were known. In addition, we present experimental results for financial networks and random networks.

Keywords Preprocessing · Exact algorithm · Parameterized algorithmics · Algorithm engineering · Gene-regulatory network · Financial network

1 Introduction

1.1 Data Reduction

Polynomial-time data reduction is a classical way of dealing with hard problems: before starting the actual solving process, one tries to reduce the size of the instance by

A preliminary shorter version of this paper appeared under the title "Optimal edge deletions for signed graph balancing" in the Proceedings of the 6th Workshop on Experimental Algorithms (WEA 2007), June 6–8, 2007, Rome, Italy, volume 4525 in Lecture Notes in Computer Science, pages 297–310, Springer, 2007.

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany. E-mail: {hueffner,betzler,niedermr}@minet.uni-jena.de.

removing or simplifying parts. More precisely, a *data reduction rule* reduces in polynomial time an instance to a smaller instance, without destroying the possibility of finding an optimal solution. Data reduction has proven useful as a general technique in coping with NP-hard problems (Guo and Niedermeier, 2007).

Many data reduction rules have been developed in a problem-specific and ad-hoc way based on small fixed-size substructures. A typical example is the following one from Wernicke (2003) for VERTEX BIPARTIZATION, a problem closely related to our main study problem BALANCED SUBGRAPH (see Sect. 1.2). VERTEX BIPARTIZATION asks to delete a minimum number of vertices from an undirected graph to make it bipartite, or equivalently, to destroy all cycles of odd length.

Rule 1 *Let $G = (V,E)$ be a* VERTEX BIPARTIZATION *instance and let abcd be an induced four-vertex cycle (C_4) in G, where the two nonadjacent vertices b and d have degree* 2 *in G. Then remove b.*

This rule is correct because without loss of generality we never need to delete b, since deleting a or c destroys at least as many odd-length cycles, and further there is an odd cycle containing b iff there is an odd cycle containing d. In a similar way, considering structures of a few vertices, Wernicke (2003) presented several more data reduction rules. Another example is the "vertex folding" rule, which gets rid of degree-2 vertices in VERTEX COVER instances (Chen et al., 2001).

When looking at the correctness proofs of these reduction rules, one notices that they are often, implicitly or explicitly, based on a separator (that is, a set of vertices whose deletion separates the graph into at least two connected components): we have a small number of vertices (e. g. b and d in Rule 1) that are separated by a small separator (e. g. a and c in Rule 1) from the rest of the graph. Similarly, in the case of VERTEX COVER a degree-2 vertex forms a component of size 1 that is divided from the rest of the graph by a separator of size 2. Our aim is to generalize this kind of data reduction rule.

In Sect. 2, we present a scheme that can provide a data reduction for such structures in BALANCED SUBGRAPH instances in general, without the need to manually derive and prove rules for each fixed structure as in Rule 1. The idea is to find a small separator S that cuts off a small component C from the rest of the graph. Then, we replace S and C by a smaller gadget that exhibits the same behavior with respect to the underlying graph problem.

A similar method has been suggested by Polzin and Vahdati Daneshmand (2006) for the STEINER TREE problem. However, they do not employ gadgets and have no formal characterization of reducible cases. Another similar method are *crown reduction* rules (Abu-Khzam et al., 2007). Crown reductions also work by finding a separator S that cuts off a component C, and impose additional demands on S and C (for instance for VERTEX COVER, C must be an independent set and there must be a matching between S and C that matches all vertices of S (Abu-Khzam et al., 2004)). The main difference to our approach is that we do not assume any particular properties of S and C, except that they are small.

Another related technique are tree decompositions (Hicks et al., 2005; Bodlaender and Koster, 2008). Here, the major difference is that they require the graph to be "covered" with small separators; for example, a large clique thwarts the approach.

In contrast, our approach can be applied profitably even when only some parts of the graph have small separators, which commonly occurs with real-world inputs like those examined in Sect. 4.

1.2 Balanced Subgraph

The NP-hard BALANCED SUBGRAPH problem is defined on *signed graphs*, which are undirected graphs where each edge is annotated with an element of the sign group (that is, the unique two-element group, which can for example be denoted by the two elements 0 and 1 and the binary operation $a \circ b := (a+b) \bmod 2$). The concept of signed graphs has been introduced first by Harary (1953) in the context of social networks, and has been rediscovered frequently since, as it is a natural model for many applications; see Zaslavsky (1998) for a bibliography of signed graphs. The central concept is that of a *balanced* signed graph: A signed graph $G = (V,E)$ with edges labeled by $h : E \to \{0,1\}$ is balanced if there is a vertex coloring $f : V \to \{0,1\}$ such that

$$\forall \{u,v\} \in E : h(\{u,v\}) \equiv (f(u) + f(v)) \pmod 2. \tag{1}$$

Put another way, a 0-edge demands that its endpoints have the same color, and a 1-edge demands that they have different colors. Therefore, in the following we use the notations "=-edge" and "≠-edge" instead. Let further $E_=$ be the set of =-edges and $E_\neq$ the set of ≠-edges.

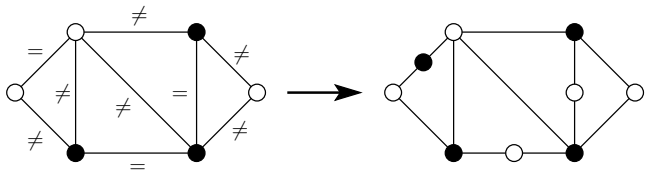
Balanced graphs generalize bipartite graphs, since bipartite graphs are balanced graphs that contain only ≠-edges. Kőnig (1936) proved the following characterization of balanced graphs. For a graph $G = (V,E)$, the following are equivalent:

1. V can be partitioned into two sets V_1 and V_2 called *sides* such that there is no ≠-edge $\{v,w\} \in E$ with both $v,w \in V_1$ or both $v,w \in V_2$ and no =-edge $\{v,w\}$ with $v \in V_1$ and $w \in V_2$.
2. V can be colored with two colors such that for all $\{v,w\} \in E_\neq$ the vertices v and w have different colors, and for all $\{v,w\} \in E_=$ the vertices v and w have the same color. The color classes correspond to the sides.
3. G does not contain cycles with an odd number of ≠-edges.

Using the characterization by a coloring, it is easy to see that balance of a signed graph can be checked in linear time by depth-first search. The BALANCED SUBGRAPH problem is now defined as follows:

> BALANCED SUBGRAPH
> **Instance:** A signed graph $G = (V,E)$ and an integer $k \geq 0$.
> **Question:** Can G be transformed by up to k edge deletions into a balanced graph?

EDGE BIPARTIZATION is the edge-deletion version of VERTEX BIPARTIZATION. Since EDGE BIPARTIZATION is the special case of BALANCED SUBGRAPH where there are only ≠-edges, NP-hardness and approximation hardness results for EDGE BIPARTIZATION carry over to BALANCED SUBGRAPH. In particular, BALANCED SUBGRAPH remains NP-hard even in triangle-free graphs with maximum degree



(a) (b)

Fig. 1: Example for a yes-instance of BALANCED SUBGRAPH and an equivalent yes-instance of EDGE BIPARTIZATION. Colors serve to indicate that the graph is balanced (a) resp. bipartite (b).

three (Yannakakis, 1981). The problem is MaxSNP-hard (Papadimitriou and Yannakakis, 1991), and assuming Khot's Unique Games Conjecture, it is even NP-hard to approximate within any constant factor (Khot, 2002).

There is a simple reduction from BALANCED SUBGRAPH to EDGE BIPARTIZATION, which allows to transfer some tractability results from EDGE BIPARTIZATION. For this, we subdivide each $=$-edge by one vertex (see Figure 1). After this, it is easy to show that a BALANCED SUBGRAPH instance can be solved with k edge deletions iff the transformed EDGE BIPARTIZATION instance can be solved with k edge deletions. In particular, the above reduction implies that BALANCED SUBGRAPH can be solved in polynomial time on *weakly bipartite* graphs, which comprise bipartite graphs and planar graphs (Grötschel and Pulleyblank, 1981). Further, BALANCED SUBGRAPH can be approximated to a factor of $O(\sqrt{\log n})$ in polynomial time (Agarwal et al., 2005), where n is the number of vertices in the input. Another approximation algorithm finds in polynomial time a solution of size $O(k \log k)$, where k is the size of an optimal solution (Avidor and Langberg, 2007).

MAXCUT is EDGE BIPARTIZATION with the dual optimization objective of maximizing the number of undeleted edges. If we consider this objective for BALANCED SUBGRAPH, we cannot directly obtain the same approximation factor of 0.878 as for MAXCUT (Goemans and Williamson, 1995), since the number of edges might double in the reduction. However, it was shown by Thagard and Verbeurgt (1998) and independently by DasGupta et al. (2007) that the semidefinite programming of Goemans and Williamson (1995) can be adapted to BALANCED SUBGRAPH without impairing the approximation factor. While these are much stronger guarantees than for the minimization variant, minimizing the number of deleted edges is in many settings the more natural model; for example, in the biological networks considered by DasGupta et al. (2007), only few edges need to be deleted to make them balanced (see Table 1 in Sect. 4).

Similar to EDGE BIPARTIZATION, polyhedral approaches have been used for BALANCED SUBGRAPH (Barahona and Ridha Mahjoub, 1989; Boros and Hammer, 1991), which also cover the weighted case. Coleman et al. (2008) examine the practical performance of several approximation algorithms for BALANCED SUBGRAPH.

A different approach to BALANCED SUBGRAPH is parameterized algorithmics (Downey and Fellows, 1999; Flum and Grohe, 2006; Niedermeier, 2006). The idea is to confine the combinatorial explosion to a parameter k. A problem is called *fixed-*

parameter tractable with respect to a parameter k if an instance of size n can be solved in $f(k) \cdot n^{O(1)}$ time, where f is an arbitrary computable function depending only on k. EDGE BIPARTIZATION can be solved in $O(2^k m^2)$ time (Guo et al., 2006), where m is the number of edges in the input, and thus by the transformation described above BALANCED SUBGRAPH can be solved in the same time bound, showing the fixed-parameter tractability of BALANCED SUBGRAPH. Since in many applications k is much smaller than m, this is a promising approach.

Applications. BALANCED SUBGRAPH has a large number of applications. One of the oldest is in modeling social networks (Harary, 1959). Here, an $=$-edge models a positive or friendly connection, a $\neq$-edge models a negative or unfriendly connection, and a non-edge a neutral connection or lack of contact. The conjecture is that changes in social networks can be explained by a striving towards balance in this signed graph. The number of edge deletions required to obtain a balanced graph is then a measure of the distance from stability. An example was given by Antal et al. (2006) for the relations between nations prior to World War I.

The following application for gene regulatory networks will be central to some of our experiments in Sect. 4. DasGupta et al. (2007) used balance in signed graphs to model the concept of "monotone subsystems" under the name of "sign-consistent graphs". They examined dynamical systems, where a gene is modeled as a vertex and an activating connection is modeled as an $=$-edge and an inhibiting connection is modeled as a $\neq$-edge. The claim is that biological dynamical systems are close to being balanced, and that finding a minimum set of edges to delete to make the graph balanced can be used to decompose the graph into "monotone subsystems", which exhibit stable behavior and thus allow a better understanding of the dynamics of a system.

Further applications of BALANCED SUBGRAPH appear in statistical physics (Barahona, 1982), portfolio risk analysis (Harary et al., 2002), and VLSI design (Chiang et al., 2007).

1.3 Contributions

We show a general scheme for data reduction for BALANCED SUBGRAPH by replacing small components C that can be cut off by a small separator S with gadgets (Sect. 2.1). In particular, all separators with $|S| = 2$ and $|C| \geq 1$ and all separators with $|S| = 3$ and $|C| \geq 2$ allow to simplify the instance (Corollary 1). To solve the instances that remain after data reduction, we adapt a fixed-parameter algorithm for EDGE BIPARTIZATION yielding an $O(2^k \cdot m^2)$ running time with m denoting the number of edges. Further, we present a speed-up trick that often reduces the running time of this algorithm in our experiments from days to few seconds (Sect. 4). We experimented with the real-world biological instances provided by DasGupta et al. (2007). Our program needs similar amounts of time (up to about 1 h), but can solve them optimally. Moreover, we experimented with synthetic data and further real-world instances (such as financial networks) to chart the border of feasibility of our algorithm.

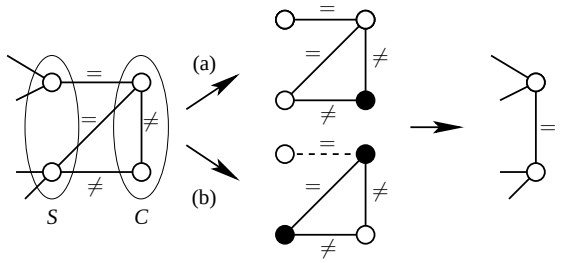


Fig. 2: Example for Reduction Scheme 1

2 Data Reduction

In this section, we present polynomial-time executable data reduction rules for the BALANCED SUBGRAPH problem. We assume that G is a *multigraph*, which can have multiple labeled edges between two vertices. This is useful for several applications and actually makes the data reductions easier to formulate.

The data reduction scheme is based on finding small separators and a novel gadget construction scheme. It unifies and generalizes a number of previously known data reductions (Wernicke, 2003) and seems applicable to a wider range of graph problems where a coloring or a subset of the vertices is sought.

2.1 Data reduction scheme

In this section, we describe how to obtain data reduction for BALANCED SUBGRAPH from a small separator S that cuts off a small component C from the rest of the graph. For this, we examine the effect of S and C on optimal solutions and replace them with an equivalent smaller gadget. As is standard with separator-based methods, the behavior of an (S,C)-pair is examined by exhaustively enumerating possible states of the separator and finding exact solutions to the small component C. For BALANCED SUBGRAPH, the states are possible colorings of the vertices in the separator in an optimal solution. We now present the scheme more formally.

Reduction Scheme 1 *Let S be a separator and let C be a small component obtained by deleting S from the given graph G. Then, determine for each of the (up to symmetry) $2^{|S|-1}$ colorings of S the size of an optimal solution for the induced subgraph $G[S \cup C]$ and replace in G the subgraph $G[S \cup C]$ by a gadget that contains the vertices of S and possibly some new vertices.*

The above scheme leaves open some details. Before filling them in, let us show a simple example.

Example 1 In Figure 2, the separator S cuts off the vertices in C from the rest of the graph. Up to symmetry, there are only two possibilities how the vertices in S can be colored: equal or unequal. If they are colored equal (a), the subgraph $G[S \cup C]$ is balanced without edge deletions. Otherwise (b), one edge deletion is required (dashed

line). We can simulate this behavior with a single $=$-edge between the two vertices in S: it also incurs a cost of 0 when the two vertices of S are colored equal, and a cost of 1 otherwise. Therefore, we can replace the subgraph $G[S \cup C]$ by the gadget shown on the right.

To fully describe the reduction scheme, four questions have to be answered:

(a) The instances $G[S \cup C]$ have some vertices (those of the separator) pre-colored. How to solve these already partly colored instances?
(b) There is a combinatorial explosion with the sizes of S and C affecting the running time. Therefore, how do we restrict the choices of S and C?
(c) How can we efficiently find useful (S,C)-combinations?
(d) If existing, how can we construct a gadget that is smaller than $G[S \cup C]$ and correctly "simulates" $G[S \cup C]$?

Regarding (a), we reduce the instance to an instance without pre-colored vertices, and then solve the instance recursively. For this, we merge all vertices pre-colored black into a single uncolored vertex and all vertices pre-colored white into a single uncolored vertex. Here, to *merge* two vertices v and w means to delete v and w and all incident edges, and add a new vertex x with edges from x to each vertex that was connected to v or w. We then add a sufficient number (e. g. $|E|$) of edges labeled $\neq$ between the two new vertices, to ensure that no solution colors them equally. Any solution for this instance will then color the two vertices differently, and we can (possibly by flipping all colors) reconstruct a solution for the pre-colored instance.

Regarding (b), this can be simply done by imposing a fixed limit. In the implementation used in our experiments, we restrict the size of S to at most 4, mainly because of difficulties with the gadget construction. The size of C is (somewhat arbitrarily) restricted by 32; however, due to the structure of our instances, this limit did not play a role, because all components found were much smaller.

As to (c) and (d), we will answer these questions in the next two subsections. Finally, note that our approach obviously is only promising in case of graphs possessing small separators. Clearly, this excludes "highly" connected graphs. Fortunately, many real-word networks contain "enough" small separators.

2.2 Efficiently finding separators

To improve running time, we special-case the search for separators of size 0 (that is, the graph consists of more than one connected component) and separators of size 1 (that is, articulation points). They can be found in linear time using depth-first search (Gabow, 2000). For these cases, the gadget construction can be omitted: the 2-connected components[1] can be treated independently, and optimal colorings of two components can always be merged (possibly by flipping all colors in one component), since they overlap only in one vertex. Note that this phase in particular removes all degree-1 vertices.

[1] A graph is 2-connected if there are at least two vertex-disjoint paths between any pair of vertices from this graph.

Separators of size 2 can also be found in linear time (Hopcroft and Tarjan, 1973). However, we did not implement this algorithm, since it is quite complicated and error-prone to implement (several errors in the original publication have been pointed out (Gutwenger and Mutzel, 2000)).

Separators of size k for small k can be found efficiently by flow techniques (Henzinger et al., 2000). However, after some experiments we settled for the subsequently described heuristic instead, which is faster and produced no worse results in our tests. For a vertex set X, let $N(X) := \{u \mid \{u,v\} \in E \wedge v \in X\} \setminus X$. For each vertex v, set $C := \{v\}$ and iteratively enlarge C by a vertex v' that minimizes the size of $S := N(C \cup \{v'\})$ until $|C|$ exceeds the size limit. The size of S can grow and shrink during this process; we record all combinations of S and C with $S \leq 4$.

To get a heuristic speedup, it is useful to first treat separators that are easy to deal with, but promise large reductions. Therefore, we sort the (S,C)-combinations primarily by increasing size of S and secondarily by decreasing size of C. In our experiments, the finding of separators in the above way altogether never took more than few seconds for graphs with up to about 1000 vertices.

2.3 Gadget construction

The goal is to show how the subgraph $G[S \cup C]$ induced by the separator S and the small component C can be replaced by a smaller, "equivalent" subgraph (gadget). A simple case has already been described in Example 1. Now, we describe a general methodology, leading also to theoretically interesting problems that deserve further investigation.

Let us call a separator of size i simply *i-cut*. As mentioned before, it is easy to deal with 0- and 1-cuts. Hence, we focus on larger separators, thereby describing constructions delivering optimal gadgets in case of 2- and 3-cuts and a heuristic approach for 4-cuts. We also briefly discuss the mathematical and algorithmic challenges behind constructing gadgets for i-cuts for general i.

By an *optimal* gadget we refer to one with a minimum number of vertices (the alternative setting of minimizing the number of edges might be worth consideration as well). When speaking of an *equivalent* gadget which replaces the subgraph $G[S \cup C]$, we refer to a subgraph H with the following properties:

1. Gadget H contains all vertices from S and possibly more; in particular, S forms the "interface" where H is plugged in instead of $G[S \cup C]$.
2. The original graph G has a solution for BALANCED SUBGRAPH of size k iff the modified graph where H replaces $G[S \cup C]$ has a solution of size $k' \leq k$, where the difference between k' and k is determined by the gadget. Moreover, an optimal solution for G can be reconstructed from an optimal solution for the modified graph.

2.3.1 Gadget construction for 2-cuts

Example 1 shows a special case of 2-cuts. Up to symmetry, there are only two colorings of the two separator vertices u and v. In each of these two cases, we compute

recursively an optimal solution for $G[S \cup C]$, which can be done quickly, since only small S and C are considered.

Let n_e be the size of an optimal solution for $G[S \cup C]$ where u and v have equal colors, and let n_d be the size of an optimal solution where they have distinct colors. We perform the following gadget construction, where the gadget consists solely of vertices from S. If $n_e \geq n_d$, then remove C and all edges within S and add $n_e - n_d$ edges labeled $\neq$ between u and v. Otherwise, remove C and all edges within S and add $n_d - n_e$ edges labeled $=$ between u and v. Note that reducing 2-cuts in particular gets rid of all vertices of degree 2.

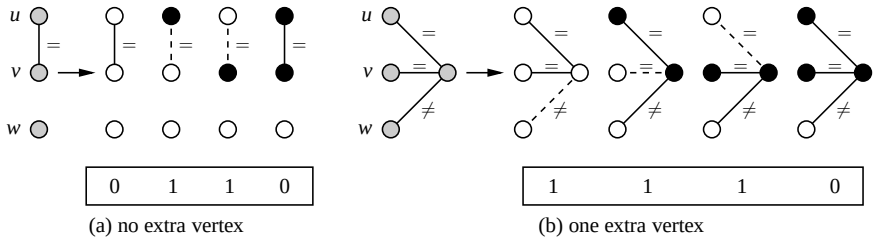
Lemma 1 *Let G be the original graph and let G' be the graph obtained from G by performing the described gadget replacement. Then G has a solution of size k iff G' has a solution of size $k - \min\{n_e, n_d\}$.*

Proof Consider first the case $n_e \geq n_d$. From a solution of size k for G, we can construct a solution of size $k - n_d$ for G' by using the same coloring restricted to the remaining vertices and deleting all inconsistent edges. If this solution colors u and v differently, we save n_d edges within $G[S \cup C]$; the $\neq$-edges do not incur any additional cost. If this solution colors u and v equally, we save n_e edges within $G[S \cup C]$, but need to delete all $n_e - n_d$ $\neq$-edges between u and v, also resulting in a solution of size $k - n_d$. In the same way, we can construct from a solution of size $k - n_d$ for G' a solution of size k for G. The case $n_e < n_d$ works in complete analogy. □

2.3.2 Gadget construction for 3-cuts

The basic approach is the same as for 2-cuts. The gadget construction, however, becomes more intricate. The idea is to construct the final gadget from *atomic gadgets*, which can be added independently until in total they have the desired effect. To characterize the effect of an atomic gadget, we introduce the concept of a *cost vector*. In the case of 3-cuts, up to symmetry, we have four possibilities to color the vertices from the separator S. For each case, we compute the cost of an optimal BALANCED SUBGRAPH solution of $G[S \cup C]$. For a fixed order of the colorings, these values build the cost vector of the form (c_1, c_2, c_3, c_4). The goal is then to find atomic gadgets such that their corresponding atomic cost vectors add up to the cost vector associated with $G[S \cup C]$.

We show that it is sufficient to consider atomic gadgets that, besides S, have at most one additional vertex. The first type of atomic gadgets are gadgets exclusively made of vertices from S. More specifically, there are six possibilities to put exactly one edge, either labeled $=$ or $\neq$, between the three possible vertex pairings in S. Each of these possibilities yields an atomic gadget. Moreover, each of these atomic gadgets naturally one-to-one corresponds to a cost vector with 0/1-entries. For instance, let $\{u, v, w\}$ form the separator. Then, the atomic gadget with an $=$-edge between u and v corresponds to the cost vector $(0, 1, 1, 0)$ (see Figure 3a): If u and v have the same color (once white, once black), then the insertion of the $=$-edge does not cause an inconsistency. Thus, we have an additional solution cost of 0, justifying the two zero-entries in the cost vector. If u and v have different colors, then the insertion of the $=$-edge causes an inconsistency, generating an additional solution cost of 1, justifying



(a) no extra vertex (b) one extra vertex

Fig. 3: Examples for atomic gadgets for a size-3 separator $\{u,v,w\}$

the two one-entries in the cost vector. Generalizing this to the five other possibilities of putting exactly one labeled edge, we arrive at the following:

Lemma 2 *By inserting exactly one edge labeled $=$ or $\neq$ between the vertices from S, we obtain the six atomic cost vectors* $(0,0,1,1)$, $(0,1,0,1)$, $(0,1,1,0)$, $(1,0,0,1)$, $(1,0,1,0)$, *and* $(1,1,0,0)$.

All cost vectors in Lemma 2 have even parity. Hence, we need a second type of gadgets to be able to construct cost vectors with odd parity: gadgets that contain all vertices from S plus a new vertex connected to all vertices from S. We derive four atomic gadgets of this kind with different cost vectors, namely the cases that the edges connecting S to the new vertex are labeled $(\neq,\neq,\neq)$, $(=,=,\neq)$, $(\neq,=,\neq)$, or $(=,\neq,\neq)$ (an example is shown in Figure 3b).

Lemma 3 *By inserting one new vertex and connecting it to all vertices from S and assigning various edge labels, we obtain four atomic gadgets corresponding to the atomic cost vectors* $(0,1,1,1)$, $(1,0,1,1)$, $(1,1,0,1)$, *and* $(1,1,1,0)$.

The four atomic cost vectors from Lemma 3 all have odd parity. In this sense, we now may speak of *even* or *odd* cost vectors.

Now, we can describe the general gadget construction. To do so, first note that vectors where all entries have the same value x are easy because this means that the solution for $G[S \cup C]$ is independent of the coloring of S and hence one can simply remove C and all edges between vertices of S and decrease the parameter k by x. This means that if we are given a cost vector (c_1,c_2,c_3,c_4), then without loss of generality we can *normalize* it by simply subtracting or adding the vector $(1,1,1,1)$, each time decreasing or increasing the parameter by one. Now, given a cost vector (c_1,c_2,c_3,c_4), the gadget construction task one-to-one corresponds to finding a way to subtract atomic cost vectors from (c_1,c_2,c_3,c_4) such that one receives the vector $(0,0,0,0)$. If we arrive at a cost vector with at least two 0-entries that cannot be transformed into $(0,0,0,0)$, then due to the above reasoning we may also add the vector $(1,1,1,1)$. Altogether, this results in the following algorithm:

1. Compute the cost vector for given S and C.
2. Normalize the cost vector by subtracting the vector $(1,1,1,1)$ until at least one entry becomes 0.

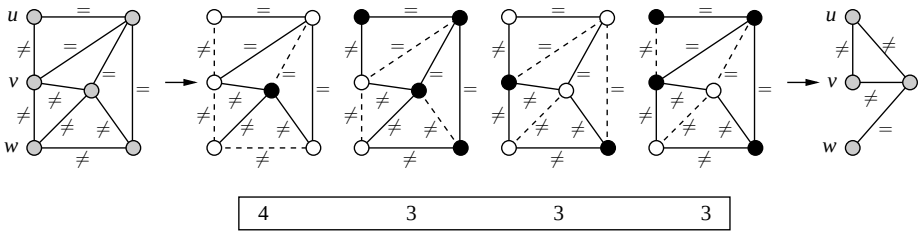


Fig. 4: Example for the construction of a gadget with $|S| = 3$

3. If the cost vector has odd parity and has more than one 0-entry, then add $(1,1,1,1)$.
4. If the cost vector has odd parity, then subtract a suitable odd atomic cost vector (that is, one that does not produce negative entries).
5. While the vector is not $(0,0,0,0)$, repeat:
   (a) If the cost vector has three 0-entries, then add $(1,1,1,1)$.
   (b) Subtract a suitable even atomic cost vector that decreases the maximum entry.

Example 2 In Figure 4, we start with the induced subgraph $G[S \cup C]$, where $S = \{u,v,w\}$ is the separator. In the middle we show optimal solutions for the (up to symmetry) four possible colorings of S and mark by a dashed line the edges that have to be deleted. The number of edge deletions are displayed below these figures, forming the cost vector $(4,3,3,3)$. Normalization yields the vector $(1,0,0,0)$. Since this is an odd vector with more than one zero, it gets padded to $(2,1,1,1)$. This is an odd vector, so we need a gadget using an extra vertex and three edges (Lemma 3). From the three vectors whose subtraction decreases the maximum element 2, we arbitrarily choose $(1,1,1,0)$, corresponding to adding from a new vertex a $\neq$-edge to u, a $\neq$-edge to v, and an $=$-edge to w. The remaining cost vector $(1,0,0,1)$ can be covered by adding a $\neq$-edge between u and v, leaving the all-zero vector. The resulting gadget is shown on the right of Figure 4. We have subtracted the all-1 vector twice and added it once, and therefore the parameter decreases by one.

Theorem 1 *The above algorithm produces a gadget with the minimum number of vertices for every pair (S,C) where S is a 3-cut.*

Proof First of all, it is clear from the one-to-one correspondence between atomic gadgets and atomic cost vectors that by "superimposing" the atomic gadgets corresponding to each (possibly multiple times) subtracted atomic cost vector, one directly arrives at an overall gadget (with possibly multiple edges). Concerning the usage of the normalization vector $(1,1,1,1)$, we have already argued before that this does not affect the correctness of the gadget construction. Hence, in the remainder we focus on showing that the algorithm always terminates with having generated the vector $(0,0,0,0)$ by subtracting atomic cost vectors and possibly subtracting or adding $(1,1,1,1)$.

Once subtracting a suitable odd atomic vector, we arrive at a cost vector with even parity (Steps 1–4). In the further process (Step 5), we will always have an even parity

and it suffices to concentrate on the termination of the while-loop of the algorithm. Since the six atomic cost vectors represent all possible vectors with exactly two 1-entries, as long as we have at least two nonzero-entries in the cost vector, there is at least one even atomic cost vector that we can subtract. Now, assume that we have a cost vector with three zero-entries and one nonzero-entry *e* for an even *e* (this is the only remaining possibility besides having the all nonzero-entry). Then, the algorithm adds $(1,1,1,1)$. Now, after this addition we can three times subtract an atomic vector, decreasing the entry *e* by two before again all originally zero-entries become zero again. Repeatedly proceeding this way, we thus always can arrive at the vector $(0,0,0,0)$ in a finite number of steps. The construction is optimal because the gadget has at most one additional vertex (besides *S*), and this happens only for odd cost vectors, where it is unavoidable. □

Note that the construction is not necessarily optimal with respect to the number of edges introduced, nor with respect to the decrease in *k*. However, in our experiments these objectives rarely had different optimal solutions.

As a consequence of the considerations so far, we obtain the following result, illustrating the power of our approach.

Corollary 1 *With the described data reduction scheme, all separators with* $|S| = 2$ *and* $|C| \geq 1$ *and all separators with* $|S| = 3$ *and* $|C| \geq 2$ *are subject to data reduction.*

2.3.3 Gadget construction for larger cuts

The gadget construction for 3-cuts already has required quite some machinery. The case of 4-cuts becomes still much more involved due to the increased combinatorial complexity. A provably optimal gadget construction as for 3-cuts currently does not seem practically feasible. Thus, we have chosen a heuristic approach for finding and constructing gadgets for 4-cuts.

We conjecture that atomic gadgets with at most two vertices in addition to the four separator vertices suffice. Thus, we generated 2^6 atomic gadgets with no extra vertex (corresponding to the choices of labels for the 6 edges within a 4-vertex separator), 2^4 atomic gadgets with one extra vertex (4 edges connecting a vertex in the separator to the new vertex), and 2^9 atomic gadgets with two extra vertices (8 edges to the new vertices, and one edge connecting the two new vertices). We then filtered out those that can be obtained by combining cheaper ones, and arrived after about five minutes of computation time on a standard PC at a set of 2948 atomic gadgets. They are stored in a fixed lookup table.

Once given this toolbox of atomic gadgets, we again try to derive the all-zero vector in a way analogous to the case of 3-cuts. This procedure is now realized by an exhaustive branch & bound algorithm. We start with the normalized vector. Should this fail, the vector $(1,1,1,1)$ is added once and the procedure is repeated. Each gadget vector is associated with a cost corresponding to its number of extra vertices; this number is minimized. In fact, it is not too hard to see that this algorithm produces for 3-cuts, when given the 10 atomic cost vectors, the same result as the algorithm given for 3-cuts.

The branch & bound algorithm works quite well for cost vectors with small entries, but can become a bottleneck for vectors with high entries. We examine a simple heuristic to mitigate this in Sect. 4. We close with a description of challenges for further research that arise in our work with cost vectors. For this, we describe the scenario in a more abstract way.

Given a set S of n vectors of length l with nonnegative integer components, let a *linear combination* be a sum of some vectors of S, where vectors can occur multiple times (equivalently, have a positive integer scalar factor). Let a *basis* be a set that allows to obtain any vector of length l with nonnegative integer components as a linear combination. (The terms are chosen in analogy to vector spaces, but because of the nonnegative integer restriction, we do not have a vector space here.) We face the following problems:

- How to recognize whether a vector set is a basis?
- Given a basis and a target vector t, how to find a linear combination that produces t?
- Given a large set of vectors, how can we find a smallest or minimal basis?

In our work, we actually have a small modification of this problem because as single vector with negative components also the vector $(-1,-1,\ldots,-1)$ is allowed. Also, the vectors come at different costs (number of new vertices), and we would like to find linear combinations of minimum cost.

This touches a deep and old subject in mathematics (see e. g. Barvinok and Woods (2003); Sturmfels (1996)). Seemingly, our questions seem to be more special than what is generally studied there, but this clearly deserves future theoretical studies.

3 Fixed-Parameter Tractability

While the data reduction rules presented in Sect. 2 can often much reduce the instance size, there will typically remain a "core" that cannot be further reduced. To solve the remaining instances exactly while getting a useful worst-case time bound, we use a fixed-parameter algorithm.

As mentioned in the introductory section, EDGE BIPARTIZATION on an m-edge graph can be solved in $O(2^k m^2)$ time (Guo et al., 2006), which together with the reduction from BALANCED SUBGRAPH to EDGE BIPARTIZATION shown there demonstrates the fixed-parameter tractability of BALANCED SUBGRAPH.

Theorem 2 *Given an m-edge graph with at most k edge deletions allowed,* BALANCED SUBGRAPH *can be solved in $O(2^k \cdot m^2)$ time.*

Theorem 2 improves an $O(n^{2L} \cdot (nm)^3)$ time exact algorithm by DasGupta et al. (2007, Remark 1), where L is the number of $\neq$-edges (since clearly $k \leq L$).

In our implementation, we do not use the reduction from EDGE BIPARTIZATION, but directly modify the EDGE BIPARTIZATION algorithm to work for BALANCED SUBGRAPH. Further, we employ a heuristic speedup trick similar to the one used for an iterative compression algorithm for VERTEX BIPARTIZATION (Hüffner, 2005). We inferred speedups with this trick up to a factor of about 10^{12}.

To give the basic idea of the algorithm and to be able to describe the speedup, we briefly sketch how the iterative compression algorithm for EDGE BIPARTIZATION works; we refer to the original work (Guo et al., 2006) for details and correctness proofs. A presentation of both results can also be found in the first author's doctoral thesis (Hüffner, 2007), and a general survey on the iterative compression technique is given by (Guo et al., 2008). The key idea is to use a *compression routine* that, given a size-$(k+1)$ solution, either computes a size-k solution or proves that there is no smaller solution. We then build up a solution for a graph $G = (V,E)$ inductively: start with $E' = \emptyset$ and $X = \emptyset$; clearly, X is an optimal solution for $G[E']$ (the graph induced by E'). Now add one edge $e' \notin E'$ from E to both E' and X. Then X is still a solution for $G[E']$, although possibly not a minimum one. We can, however, obtain a minimum one by applying our compression routine. This process is repeated until $E' = E$.

The tricky part is to come up with a compression routine. For this, two additional properties are imposed: the given solution is inclusion minimal, and the smaller solution that is sought for is disjoint from the given one. The first property is easy to ensure. The second property can be assumed without loss of generality by applying a simple input transformation: we subdivide each edge that was part of the edge bipartization set by two vertices, and add the middle segment of each subdivided edge into the new edge bipartization set. This ensures that no edge of the original solution needs to be reused in a smaller solution, since one of the two neighboring edges is always suitable as replacement.

After these prerequisites, one can show that all that the compression routine has to do in order to compress a solution X of size $k+1$ to a solution of size k is to find an edge cut of size k between the two halves of a *valid partition* (Guo et al., 2006). Here, a valid partition is a partition of the endpoints of the edges in X into two halves where each of the two endpoints of one edge is in a different half. All $O(2^k)$ valid partitions are then simply tried by brute force.

It is easy to verify that a small tweak to the algorithm suffices to make it work for BALANCED SUBGRAPH (Hüffner, 2007). The only required change is in the inductive main loop: when adding an edge and ensuring the minimality of the new solution, one has to take the edge labels into account.

Heuristic speedup. In our experiments, the plain iterative compression algorithm turned out to be too slow in many cases. Hence, we developed a technique for speeding up the algorithm. Although heuristic in nature (that is, without provable worst-case running time improvement), this approach turned out to be very effective and it was decisive in order to optimally solve hard problem instances quickly. A similar speed-up trick has been used (Hüffner, 2005) for an iterative compression algorithm for the VERTEX BIPARTIZATION problem.

The idea is, in the transformation that ensures disjointness of a smaller solution, to choose one of the two end edges into the solution instead of the middle edge. The correctness arguments are still valid with this change. However, it can happen then that two edges in the solution share an endpoint. This can be exploited: the number of valid partitions halves, since it is no longer possible to place the coinciding endpoints into different halves. To obtain the maximum gain from this, we have to choose the side where we take the solution edge such that the number of incident solution edges

is maximized. In other words, for the graph induced by X, we need to find a minimum set of vertices such that each edge contains at least one of these vertices. This is the well-known VERTEX COVER problem. While VERTEX COVER is NP-hard, the resulting instances are small and sparse (e. g., 80 vertices and 80 edges), and can be easily solved by a simple branching strategy.

Clearly, the above approach does not give any worst-case improvements, since it is easy to construct examples where the solution set X has no incident edges at all. In our experiments, however, we experienced significant speed-ups. For instance, whereas without the above trick the iterative compression algorithm took days for a yeast graph instance, with the heuristic it could be solved in seconds. In an instance where without the heuristic one would have checked 2^{74} valid partitions of the endpoints of the edges in X, due to the trick only 2^{34} valid partitions had to be considered. Thus, only due to the trick iterative compression became feasible, saving a factor of $2^{40} \approx 10^{12}$ in the running time.

4 Implementation and experiments

We applied our data reduction for BALANCED SUBGRAPH (Sect. 2) combined with the improved iterative compression routine (Sect. 3) to gene regulatory networks, randomly generated graphs, and financial networks. Our implementation consists of about 1600 lines of Objective Caml (Leroy et al., 1996) code and about 300 lines of C code that implements the time-critical compression routine of the iterative compression method. All experiments were run on an AMD Athlon 64 3400+ machine with 2.4 GHz, 512 KB cache, and 1 GB main memory running under the Debian GNU/Linux 3.1 operating system. The program was compiled with Objective Caml 3.08.3 and the GNU gcc 3.3.5 compiler using the options "-O3 -march=athlon". For the approximation algorithm by DasGupta et al. (2007), we used MATLAB version 7.0.1.24704 (R14). Our source code is available as free software under the GNU General Public License from http://theinf1.informatik.uni-jena.de/bsg/.

Besides the data reduction rules described in Sect. 2, we additionally delete self loops and pairs of edges sharing the same end vertices if the edges have different signs. These reductions can be seen as special cases of our data reduction scheme from Sect. 2 with $|C| = 0, |S| = 1$ and $|C| = 0, |S| = 2$, respectively. Furthermore, we only replace a small component by a gadget if this leads to an improvement; that is, either the number of vertices is reduced, or, in the case of an equal number of vertices, the number of edges is reduced.

Additionally, we tested a heuristic running time improvement to circumvent a problem with the data reduction based on 4-cuts: For some instances the running time drastically increased because we encountered a cost vector with entries having high values. This increased the number of possible linear combinations and therefore the running time. An example appeared when the algorithm processed the regulatory yeast network: it ran into the cost vector $(2,8,8,0,31,39,39,31)$, and therefore the instance could not be solved within several hours (whereas it could be solved without 4-cut reduction within minutes). To take advantage of 4-cut reductions without wasting hours of running time through such (rarely occurring) cases, based on ex-

Table 1: Comparison of approximation (DasGupta et al., 2007) and our exact algorithm. Here, t denotes the running time in minutes. For the approximation algorithm, "$k \leq$" is the solution size, and "$k \geq$" is the lower bound gained from the approximation guarantee. The approximation algorithm was run with 500 randomizations.

			Approximation			Exact alg.	
Data set	n	m	$k \geq$	$k \leq$	t	k	t
EGFR	330	855	196	219	7	210	108
Yeast	690	1082	0	43	77	41	1
Macrophage	678	1582	218	383	44	374	1

perimental findings we introduced a new cut-off parameter. More precisely, we stop the gadget construction if the sum of the entries of a cost vector is more than 25. We experimentally show below that this cut-off value is sufficient for the considered biological and random networks.

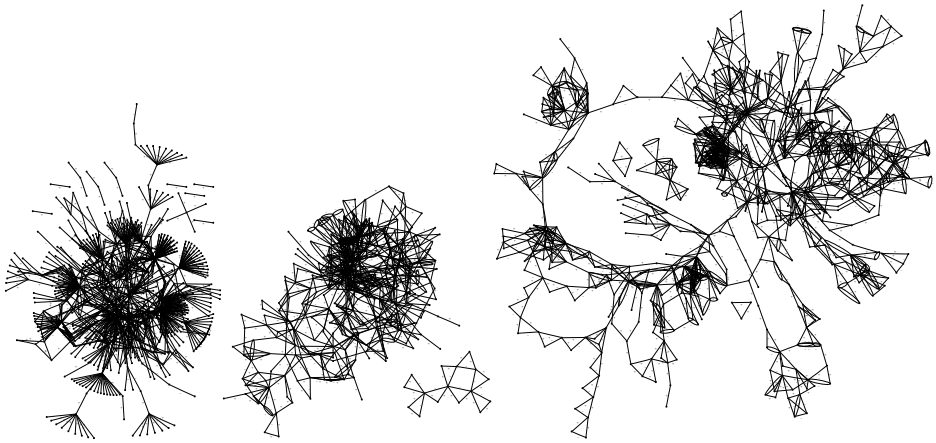
As a further comparison point, we implemented an integer linear programming (ILP)-based approach based on the following simple ILP:

$$
\begin{aligned}
&\{c_e \mid e \in E\} : \text{binary variables} \quad \textit{(edge e deleted: yes or no)}\\
&\{s_v \mid v \in V\} : \text{binary variables} \quad \textit{(vertex v colored 0 or 1)}\\
&\text{minimize} \sum_{e \in E} c_e\\
&\text{s. t. } \forall \{v,w\} \in E_{\neq} : s_v + s_w + c_{vw} \geq 1,\\
&\qquad \forall \{v,w\} \in E_{\neq} : s_v + s_w - c_{vw} \leq 1,\\
&\qquad \forall \{v,w\} \in E_{=} : s_v - s_w + c_{vw} \geq 0,\\
&\qquad \forall \{v,w\} \in E_{=} : s_w - s_v + c_{vw} \geq 0.
\end{aligned}
$$

Here, a 1 in c_e models that e is deleted, and the variables s_v model the coloring of the balanced graph that remains when deleting all edges e with $c_e = 1$. However, when solved by GNU GLPK (Makhorin, 2004), the ILP was consistently outperformed by the iterative compression approach as soon as the heuristic speedup mentioned in Sect. 3 was employed; therefore, we do not give details on its performance.

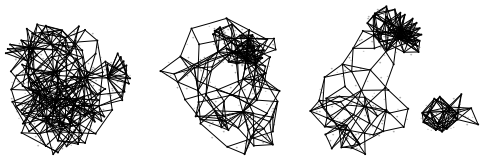
Gene regulatory networks. We started our experimental investigations with gene regulatory networks up to the size of about 700 vertices and more than 7000 edges. Unfortunately, it is still very costly to construct real-world instances from biological data, and therefore currently only few instances are available.

We begin with comparing our algorithm to the randomized approximation algorithm of DasGupta et al. (2007). The authors considered the regulatory networks of yeast and human epidermal growth factor (EGFR). We additionally examined a macrophage network (Oda et al., 2004). The three networks are graphically displayed before and after data reduction in Figure 5 and Figure 6. The yeast network is larger than the EGFR network, but one can immediately see applicability of data reduction rules in form of many degree-1 and -2 vertices. The results of both algorithms are



(a) Yeast (b) EGFR (c) Macrophage

Fig. 5: Example gene regulatory networks



(a) Yeast (b) EGFR (c) Macrophage

Fig. 6: Gene regulatory networks after data reduction

given in Table 1. Apart from giving an optimal solution instead of an approximative one, we could also decrease the running time for the yeast and macrophage networks from about one hour to less than a minute. Note, however, that the running time of the approximation algorithm could probably be much improved by implementing it in a more efficiently executed language, or simply by doing fewer randomized trials at the cost of a possibly worse result. For the macrophage network, we could compute an optimal solution of size $k = 374$. This emphasizes the importance of our data reduction rules, since for such high solution sizes the iterative compression algorithm (Sect. 3) cannot be applied directly. Furthermore, here it is remarkable that the data reduction breaks up the network into several smaller components of up to 70 vertices that can be solved by iterative compression independently, whereas for the other two networks only one large component remains after data reduction. As a further comparison point, the ILP-based approach was not able to solve the three instances even after applying the data reduction.

To investigate the power of our data reduction rules for different sizes s of the separator S, we investigated stepwise in terms of s the results for the yeast, EGFR, and macrophage networks. The results are given in Table 2, where setting s to $4r$

Table 2: Size of the largest component remaining and overall running time t (including solution by iterative compression) when reducing only separators up to size s.

	Yeast			EGFR			Macrophage		
s	n	m	t	n	m	t	n	m	t
0	690	1080	91 s	329	783	$>$ 15 h	678	1582	$>$ 1 day
1	321	709	77 s	290	727	$>$ 15 h	535	1218	$>$ 1 day
2	173	469	11 s	167	468	$>$ 15 h	140	397	$>$ 1 day
3	155	424	4 s	99	283	$>$ 15 h	113	335	$\approx$ 1 day
4	?	?	$>$ 5 h	89	259	108 min	70	228	4.5 h
4r	144	405	5.6 s	89	260	97 min	70	228	18 s

means that we use a cut-off of 25 for the sum of the entries of a cost vector in the case of cut sets of size 4. We denote applying our data reduction to a separator of size s by *s-reduction*.

The yeast network can already be solved with improved iterative compression and 2-reduction. In contrast, the EGFR network cannot be solved within reasonable time without also using 3- and 4-reduction. For the macrophage network, the use of 4-cuts reduces running time severely.

We now investigate 4-reduction with and without cut-off value. For all networks, we could achieve the best data reduction results by using 4r-reduction: As mentioned above, for the yeast network the "normal" 4-reduction does not return any results within 5 hours. In contrast to the other entries for which we aborted the experiments in Table 2, here the running time is caused by the data reduction itself and not due to the iterative compression routine. Therefore, we cannot give the size of the reduced graph. Setting the cut-off parameter to 25, we obtained an instance that is reduced more than by applying 3-reduction alone. The reason that we still cannot achieve a better overall running time is the running time for the 4r-reduction itself. For the EGFR network, the size of the largest component barely changes going from 4- to 4r-reduction, indicating that we do not lose much by the cut-off; in fact, we achieve a better overall running time for 4r. Applying 4r-reduction instead of 4-reduction to the macrophage network does not change the size of the remaining largest component, but decreases the running time from hours to seconds.

Altogether, we emphasize that we really need the combination of data reduction and the improvements of iterative compression to solve the instances.

We also considered four small regulatory networks obtained from the Panther pathways database (Mi et al., 2005), consisting of about 100 vertices and up to 200 edges. With 3-reduction we could compute optimal solutions ranging in size from 20 to 28 in split seconds.

Finally, we describe our results for two larger networks that cannot yet be solved optimally with our method. For the regulatory network for a toll-like receptor (Oda and Kitano, 2006), we could reduce the number of vertices from 688 to 244 and the number of edges from 2208 to 1159 within three minutes. For the regulatory network of the archaeon *Methanosarcina barkeri* (Feist et al., 2006), we were less successful. The number of vertices was decreased from 628 to 500 and the number of edges

Table 3: Reduction effect for random networks. Average over 5 instances for each column. Here, n is the number of vertices in the original graph, n' is the number of vertices after data reduction, m' is the number of edges after data reduction, and t is the running time in seconds.

n	100	200	300	400	500	600	700	800	900	1000
m	172.6	336.8	492.4	640.2	791.2	970.6	1108.8	1286.6	1435.6	1585.6
n'	29	48.8	75	95	119.8	153.2	169.2	193.4	211.6	239.6
m'	102.3	165.8	252	324	398.4	518	565.8	672.4	734.6	815.8
t	1	7	6	5.5	6	8.5	8	15.5	18.5	15.5

from 7302 to 6845 in 25 minutes. This could be a hint that the dense structure of this network is hard to attack by our data reduction.

Random networks. To further substantiate our experimental results, we generated a test bed of random graphs with the algorithm described by Volz (2004). We tried to model the yeast network by choosing the following settings: the cluster coefficient is 0.016, the distribution of vertex degrees follows power-law with $\alpha = -2.2$, and the probability to assign $\neq$ to an edge is 20.5%.

We generated five instances each for graph sizes ranging from 100 to 1000 vertices. The number of edges of the generated instances is slightly more than 1.5 times of the number of vertices. We investigated the power of our data reduction by computing the number of vertices and edges of the reduced instances. Table 3 shows the average results for instances of each size. Independent of the instance size, about 75% of the vertices are reduced. Note that this is also true for the yeast network that we try to model.

The results given in Table 3 are obtained by setting the cut-off parameter again to 25. Redoing the test with a higher threshold of 50 did in no case change the number of reduced edges or vertices by more than one, but increased the running time for some instances from seconds to several hours.

Considering the size of instances that can be solved optimally by improved iterative compression after data reduction, here the threshold seems to be at graphs with about 500 vertices. Three out of the five instances could be optimally solved in up to 20 hours, where the sizes of the optimal solutions are between $k = 76$ and $k = 91$. The solution sizes are higher than for the yeast network, which has more than 600 vertices and an optimal solution of size 41. Because of this, the random instances seem to be somewhat more difficult than the yeast network itself, which is consistent with observations by DasGupta et al. (2007).

Financial networks. A further real-world application of BALANCED SUBGRAPH arises in the context of portfolio analysis in risk management (Harary et al., 2002). A portfolio of securities can be represented as a signed graph with the vertices denoting the securities and the edges representing the correlation between the securities. The balancedness of a subgraph then can be used to rate the predictability of the portfolio. For details of using signed graphs in risk analysis we refer to (Harary et al.,

2002). Here we show that our algorithms can be used to compute the balancedness of financial networks of up to several hundred vertices.

Data and experimental setup. We used publicly available stock data from *finance.yahoo.com* to generate market graphs. The stocks are represented by the vertices. For each pair of vertices u and v we compute the corresponding correlation coefficient C_{uv} based on the price fluctuations of u and v in a time range of 90 days (started at randomly chosen dates in the years 2003 and 2004).[2] Informally, two stocks are positively correlated if they show similar daily fluctuation in the prespecified time range and two stocks are negatively correlated if their daily fluctuations behave oppositely. If C_{uv} is higher than some prespecified threshold we set an $=$-edge between u and v and if C_{uv} is smaller than the negative threshold we set a $\neq$-edge. We made test runs with different threshold values between 0.3 and 0.4. It turned out that labelling the edges in that way led only to trivial instances with mostly $=$-edges. We think this is due to a positive correlation of all pairs of stocks depending on the general market situation. To circumvent this effect, we added an offset of 0.05 to all correlation coefficients, resulting in graphs with edges of both labels. Note that the choice of this offset-value was motivated by a study of Boginski et al. (2006) who observed that the mean of the distribution of the correlation coefficients between all pairs of stocks in the market was close to 0.05 (for periods of 500 days).

Our test bed consists of subgraphs of a graph based on 5216 stocks for which data was available at 2003 and 2004. More precisely, we randomly chose subgraphs whose sizes range from 60 to 480 vertices in steps of 30. For each such number of stocks (60, 90, ..., 480), we generated ten graphs and restricted our experiments to the largest connected component of each graph.

To compute solutions of BALANCED SUBGRAPH, in all experiments we used $4r$-reduction in combination with the improved iterative compression routine.

Results. In the following, we present our results with respect to the success of the data reduction rules, the solution sizes, and the corresponding running times. Further, we discuss some general observations.

In Figure 7, we give the results of the performance of our data reduction for the correlation threshold values 0.325, 0.35, and 0.375. In all three settings, the data reduction rules apply better for smaller graphs. We assume that this is because, due to the construction, the density of the graphs seems to increase with increasing number of vertices (e. g., 45 vertices/102 edges and 265 vertices/2538 edges). For the smallest considered graphs we can reduce the number of vertices by more than 90 %; for graphs up to 100–250 vertices (depending on the threshold) we can still reduce about 50 % of the vertices.

The size of the solution computed by the data reduction in combination with the iterative compression routine is given in Figure 8. For all settings of the threshold the size of the solution increases with an increasing number of vertices. For a lower threshold the size of the solution seems to grow faster (e. g., for a threshold of 0.325 there is an instance with 265 vertices and solution size 49 whereas for a threshold of 0.375 there is an instance with 393 vertices and solution size 37).

[2] The computation of the correlation is based on the logarithm of the daily return. Further, we do not take into account penny stocks.

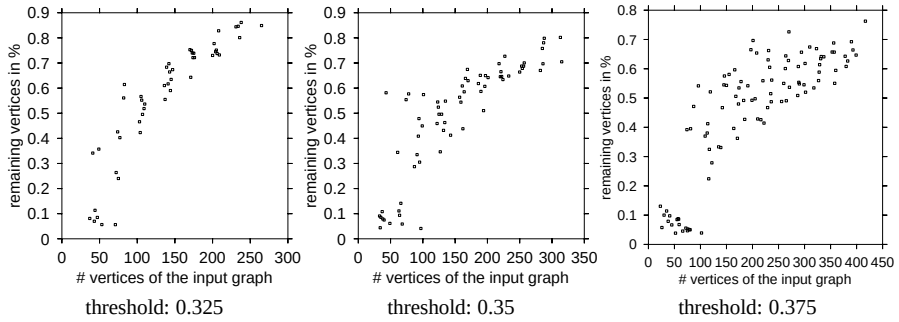


Fig. 7: Data reduction: The *x*-axis denotes the number of vertices of the original graph, and the *y*-axis shows the percentage of remaining vertices after data reduction.

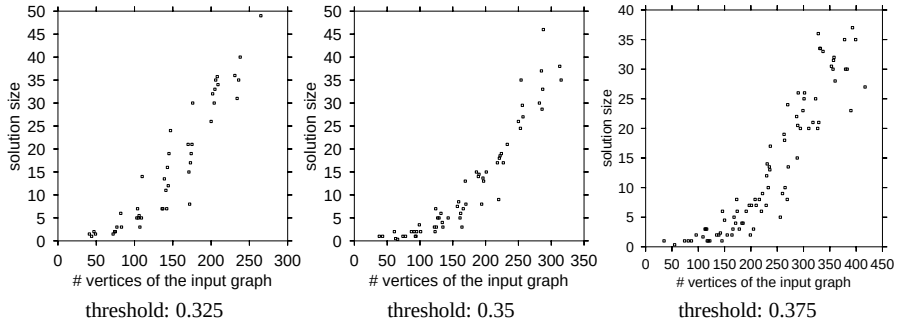


Fig. 8: Solution size/balancedness: The *x*-axis denotes the number of vertices of the original graph, and the *y*-axis gives the size of an optimal solution.

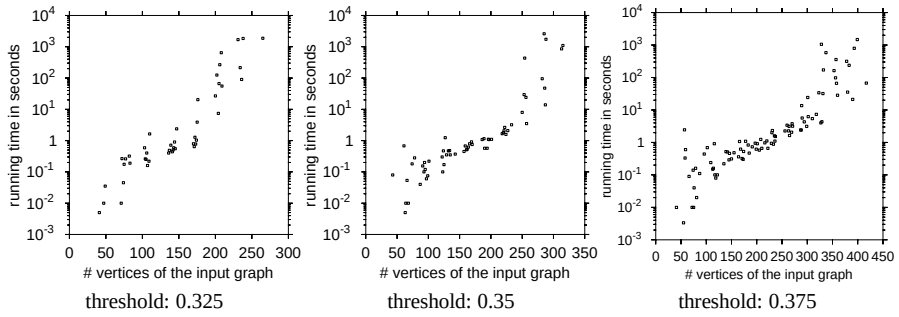


Fig. 9: Running time: The *x*-axis denotes the number of vertices of the original graph, and the *y*-axis denotes the running time in seconds.

Table 4: Number of timeouts out of 10 graphs each. Here, the number of vertices relates to the number of randomly chosen vertices in the graph construction and, therefore, slightly differ from the size of the largest connected components displayed in the other figures.

#vertices	210	240	270	300	330	360	390	420	450	480
threshold 0.325	0	6	9	10	10	10	10	10	10	10
threshold 0.35	0	0	1	3	8	10	10	10	10	10
threshold 0.375	0	0	0	0	0	0	1	4	9	10

Information about the corresponding running times is displayed in Figure 9. In our experiments, we set a timeout threshold of one hour. The number of timeouts is given in Table 4. The smaller graphs can usually be solved in less than a second for all thresholds. More precisely, for a threshold of 0.325 we can solve instances up to 180 vertices, for 0.35 up to 210 vertices, and for 0.375 up to 240 in up to one second. In running times up to several minutes we can solve instances up to 400 vertices for the threshold value 0.375.

In the following, we describe some general observations regarding the financial data that may help to explain some of our results.

First, regarding the ratio of $=$- and $\neq$-edges, we observed that it seems to be independent from the graph size, but depends on the choice of the threshold value. For a choice of 0.325 the ratio of $=$ to $\neq$ is about 5:1, for 0.35 about 6:1, and for 0.375 about 7:1. The rising number of $=$-edges could be an explanation for the decreasing size of the solutions for increasing values of the threshold (see Figure 8). Thus, we think that the better performance for instances generated with a higher threshold is due to two effects. First, the smaller density due to the higher threshold results in a stronger reduction of the instances. Second, the smaller solution size (probably due to the increased ratio of $=$- and $\neq$-edges) makes the iterative compression routine working more effectively.

Second, note that the number of edges for graphs of the same size strongly differs. For example, for the threshold of 0.375 there is a graph with 260 vertices and 2542 edges and one with the same number of vertices and only 986 edges. This might be an explanation for the variation of running times for graphs with the same number of vertices (see Figure 9).

Finally, we like to mention some test results with threshold values of 0.3 and 0.4. Whereas for 0.3 the instances become more difficult to solve (we still could solve instances up to 180 vertices) for 0.4 the instances became completely trivial due to the fact that the ratio of $=$ and $\neq$-edges increased and the graphs became very sparse. (Note the threshold value that was suggested by Boginski et al. (2005) was even 0.5 for generating the full network. But, as we only consider small subnetworks, this threshold seems to be too high to get significant connected components.)

Altogether, we showed that our algorithm can compute optimal solutions for non-trivial financial networks with up to several hundreds of vertices in reasonable time.

5 Outlook

There are numerous avenues for future research. DasGupta et al. (2007) also introduced a directed version of the BALANCED SUBGRAPH problem. The approximation results are worse than for the undirected case(DasGupta et al., 2007), which is probably why there is no implementation yet. Fortunately, the directed case can be reduced to the VERTEX BIPARTIZATION problem, which can be solved in $O(3^k \cdot mn)$ time (Reed et al., 2004; Hüffner, 2005). Again, this opens the route for experimental studies. We conclude with some further research possibilities.

- For some applications, weighted edges are of interest. Integer weights are clearly equivalent to multiple edges, which our approach already can handle. For the iterative compression part, it might be possible to handle arbitrary weights in a similar way as for CLUSTER VERTEX DELETION (Hüffner et al., 2008).
- An interesting question arising directly from our work, is to investigate how one can minimize the number of edges instead the number of vertices in the gadget construction.
- EDGE BIPARTIZATION and BALANCED SUBGRAPH still lack a problem kernel (Guo and Niedermeier, 2007; Niedermeier, 2006) with a nontrivial size bound on the problem kernel size. Perhaps our data reduction scheme can be a first step in this direction.
- Chiang et al. (2007) used the fact that BALANCED SUBGRAPH is polynomial-time solvable on planar graphs to obtain good results for their "nearly-planar" instances. In the spirit of parameterizing according to "distance from triviality" (Guo et al., 2004), it would be interesting here to have a fixed-parameter algorithm where the parameter is the "distance from planarity". It is NP-hard to solve BALANCED SUBGRAPH for graphs that are planar with already a single vertex added (Barahona, 1980); however, the number of edges added, or the minimum number of crossings of a plane drawing might be a useful parameter.
- The theoretical problems in the construction of optimal gadgets (see Sect. 2.3.3) deserve further investigation.
- In principle, our data reduction scheme is applicable to all graph problems where a coloring of the vertices is sought. This includes problems where a subset of the vertices is sought, such as VERTEX COVER or DOMINATING SET. However, it remains to find appropriate gadget constructions for problems other than BALANCED SUBGRAPH. It seems promising to extend our data reduction scheme to practical solutions of other graph problems. A loosely related approach—also based on graph separation but without the gadgeteering—has been used for solving Steiner tree problems (Polzin and Vahdati Daneshmand, 2006).
- Estivill-Castro et al. (2006, Sect. 3.3) also sketch a general approach to data reduction rules. In particular, they suggest to use the algebraic Myhill–Nerode machinery adapted to graph theory (Fellows and Langston, 1989; Downey and Fellows, 1999). It is possible that this approach can be adapted to the task of computing gadgets for arbitrary-sized separators. This might also lead to a formal characterization of graphs for which our separation-based data reduction scheme is useful. This is clearly an interesting area of further research.

– Using iterative compression Razgon and O'Sullivan (2008) have shown the problem of deleting the minimum number of clauses from a 2CNF-formula such that it becomes satisfiable to be fixed-parameter tractable with respect to the parameter "number of deleted clauses". This problem is structurally similar to BALANCED SUBGRAPH, and perhaps similar data reductions can be developed.

Acknowledgements We thank the authors of DasGupta et al. (2007) for making their source code available. Further, we thank our students Manuel Sorge, Tamara Steijger, and Thomas Zichner for help with the experiments and Jiří Matoušek (Prague) for helpful references for the integer linear combination problem. Our work has been supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms, NI 369/4), project PEAL (parameterized complexity and exact algorithms, NI 369/1), project ITKO (iterative compression for solving hard network problems, NI 369/5), and project DARE (data reduction and problem kernels, GU 1023/1). Falk Hüffner was partly supported by a fellowship of the Edmond J. Safra bioinformatic program.

References

Abu-Khzam FN, Collins RL, Fellows MR, Langston MA, Suters WH, Symons CT (2004) Kernelization algorithms for the vertex cover problem: Theory and experiments. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX '04), SIAM, pp 62–69

Abu-Khzam FN, Fellows MR, Langston MA, Suters WH (2007) Crown structures for vertex cover kernelization. Theory of Computing Systems 41(3):411–430

Agarwal A, Charikar M, Makarychev K, Makarychev Y (2005) $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC '05), ACM, pp 573–581

Antal T, Krapivsky PL, Redner S (2006) Social balance on networks: The dynamics of friendship and enmity. Physica D: Nonlinear Phenomena 224(1–2):130–136

Avidor A, Langberg M (2007) The multi-multiway cut problem. Theoretical Computer Science 377(1–3):35–42

Barahona F (1980) On the complexity of max cut. Tech. Rep. 186, IMAG, Université Joseph Fourier, Grenoble, France

Barahona F (1982) On the computational complexity of Ising spin glass models. Journal of Physics A: Mathematics and General 15(10):3241–3253

Barahona F, Ridha Mahjoub A (1989) Facets of the balanced (acyclic) induced subgraph polytope. Mathematical Programming 45(1–3):21–33

Barvinok AI, Woods K (2003) Short rational generating functions for lattice point problems. Journal of the American Mathematical Society 16(4):957–979

Bodlaender HL, Koster AMCA (2008) Combinatorial optimization on graphs of bounded treewidth. The Computer Journal 51:255–269

Boginski V, Butenko S, Pardalos PM (2005) Statistical analysis of financial networks. Computational Statistics & Data Analysis 48(2):431–443

Boginski V, Butenko S, Pardalos PM (2006) Mining market data: A network approach. Computers and Operations Research 33(11):3171–3184

Boros E, Hammer PL (1991) The max-cut problem and quadratic 0–1 optimization; polyhedral aspects, relaxations and bounds. Annals of Operations Research 33(3):151–180

Chen J, Kanj IA, Jia W (2001) Vertex cover: Further observations and further improvements. Journal of Algorithms 41(2):280–301

Chiang C, Kahng AB, Sinha S, Xu X, Zelikovsky AZ (2007) Fast and efficient bright-field AAPSM conflict detection and correction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(1):115–126

Coleman T, Saunderson J, Wirth A (2008) A local-search 2-approximation for 2-correlation-clustering. In: Proceedings of the 16th Annual European Symposium on Algorithms (ESA '08), Springer, LNCS, vol 5193, pp 308–319

DasGupta B, Enciso GA, Sontag ED, Zhang Y (2007) Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. Biosystems 90(1):161–178

Downey RG, Fellows MR (1999) Parameterized Complexity. Springer

Estivill-Castro V, Fellows MR, Langston MA, Rosamond FA (2006) FPT is P-time extremal structure I. In: Proceedings of the 1st Algorithms and Complexity in Durham Workshop (ACiD '06), College Publications, Texts in Algorithmics, vol 4, pp 1–41

Feist AM, Scholten JCM, Palsson BØ, Brockman FJ, Ideker T (2006) Modeling methanogenesis with a genome-scale metabolic reconstruction of *Methanosarcina barkeri*. Molecular Systems Biology 2:2006.0004

Fellows MR, Langston MA (1989) An analogue of the Myhill–Nerode theorem and its use in computing finite-basis characterizations. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS '89), IEEE, pp 520–525

Flum J, Grohe M (2006) Parameterized Complexity Theory. Springer

Gabow HN (2000) Path-based depth-first search for strong and biconnected components. Information Processing Letters 74(3–4):107–114

Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM 42(6):1115–1145

Grötschel M, Pulleyblank WR (1981) Weakly bipartite graphs and the max-cut problem. Operations Research Letters 1(1):23–27

Guo J, Niedermeier R (2007) Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1):31–45

Guo J, Hüffner F, Niedermeier R (2004) A structural view on parameterizing problems: Distance from triviality. In: Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04), Springer, LNCS, vol 3162, pp 162–173

Guo J, Gramm J, Hüffner F, Niedermeier R, Wernicke S (2006) Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. Journal of Computer and System Sciences 72(8):1386–1396

Guo J, Moser H, Niedermeier R (2008) Iterative compression for exactly solving NP-hard minimization problems. In: Proceedings of the DFG SPP 1126 "Algorithmics of Large and Complex Networks", Springer, LNCS, to appear

Gutwenger C, Mutzel P (2000) A linear time implementation of SPQR-trees. In: Proceedings of the 8th International Symposium on Graph Drawing (GD '00), Springer, LNCS, vol 1984, pp 77–90

Harary F (1953) On the notion of balance of a signed graph. Michigan Mathematical Journal 2(2):143–146

Harary F (1959) On the measurement of structural balance. Behavioral Science 4(4):316–323

Harary F, Lim MH, Wunsch DC (2002) Signed graphs for portfolio analysis in risk management. IMA Journal of Management Mathematics 13(3):201–210

Henzinger MR, Rao S, Gabow HN (2000) Computing vertex connectivity: New bounds from old techniques. Journal of Algorithms 43(2):222–250

Hicks IV, Koster AMCA, Kolotoğlu E (2005) Branch and tree decomposition techniques for discrete optimization. In: TutORials 2005, Tutorials in Operations Research, INFORMS, pp 1–29

Hopcroft JE, Tarjan RE (1973) Dividing a graph into triconnected components. SIAM Journal on Computing 2(3):135–158

Hüffner F (2005) Algorithm engineering for optimal graph bipartization. In: Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA '05), Springer, LNCS, vol 3503, pp 240–252, extended version to appear in *Journal of Graph Algorithms and Applications*.

Hüffner F (2007) Algorithms and experiments for parameterized approaches to hard graph problems. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena

Hüffner F, Komusiewicz C, Moser H, Niedermeier R (2008) Fixed-parameter algorithms for cluster vertex deletion. In: Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN '08), Springer, LNCS, vol 4598, pp 711–722, extended version to appear in *Theory of Computing Systems*.

Khot S (2002) On the power of unique 2-prover 1-round games. In: Proceedings of the 34th ACM Symposium on Theory of Computing (STOC '02), ACM, pp 767–775

Kőnig D (1936) Theorie der endlichen und unendlichen Graphen. Akademische Verlagsgesellschaft, Leipzig, English translation: *Theory of Finite and Infinite Graphs*, Birkhäuser, 1990

Leroy X, Vouillon J, Doligez D, et al. (1996) The Objective Caml system. Available on the web, http://caml.inria.fr/ocaml/

Makhorin A (2004) GNU Linear Programming Kit Reference Manual Version 4.8. Department of Applied Informatics, Moscow Aviation Institute

Mi H, Lazareva-Ulitsky B, Loo R, Kejariwal A, Vandergriff J, Rabkin S, Guo N, Muruganujan A, Doremieux O, Campbell MJ, Kitano H, Thomas PD (2005) The PANTHER database of protein families, subfamilies, functions and pathways. Nucleic Acids Research 33(Supplement 1):284–288

Niedermeier R (2006) Invitation to Fixed-Parameter Algorithms. Oxford University Press

Oda K, Kitano H (2006) A comprehensive map of the toll-like receptor signaling network. Molecular Systems Biology 2:2006.0015

Oda K, Kimura T, Matsuoka Y, Funahashi A, Muramatsu M, Kitano H (2004) Molecular interaction map of a macrophage. AfCS Research Reports 2:14

Papadimitriou CH, Yannakakis M (1991) Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43(3):425–440

Polzin T, Vahdati Daneshmand S (2006) Practical partitioning-based methods for the Steiner problem. In: Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA '06), Springer, LNCS, vol 4007, pp 241–252

Razgon I, O'Sullivan B (2008) Almost 2-SAT is fixed-parameter tractable. In: Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP '08), Springer, LNCS, vol 5125, pp 551–562

Reed B, Smith K, Vetta A (2004) Finding odd cycle transversals. Operations Research Letters 32(4):299–301

Sturmfels B (1996) Gröbner Bases and Convex Polytopes, University Lecture Series, vol 8. American Mathematical Society

Thagard P, Verbeurgt K (1998) Coherence as constraint satisfaction. Cognitive Science 22(1):1–24

Volz E (2004) Random networks with tunable degree distribution and clustering. Physical Review E 70(5):056115

Wernicke S (2003) On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen

Yannakakis M (1981) Edge-deletion problems. SIAM Journal on Computing 10(2):297–309

Zaslavsky T (1998) Bibliography of signed and gain graphs. Electronic Journal of Combinatorics DS8, updated version available at http://www.math.binghamton.edu/zaslav/Bsg/.