Proc. 6th WEA, Vol. 4525 in LNCS, pp. 297–310, Springer, 2007

# Optimal Edge Deletions for Signed Graph Balancing[⋆]

Falk Hüffner, Nadja Betzler, and Rolf Niedermeier

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

**Abstract.** The Balanced Subgraph problem (edge deletion variant) 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. We present an exact algorithm for Balanced Subgraph based on a combination of data reduction rules and a fixed-parameter algorithm. The data reduction is based on finding small separators and a novel gadget construction scheme. The fixed-parameter algorithm is based on iterative compression with a very effective heuristic speedup. Our implementation can solve biological real-world instances exactly for which previously only approximations [DasGupta et al., WEA 2006] were known.

## 1 Introduction

The concept of balanced signed graphs was first introduced by Harary [11] for the analysis of social networks (see [22] for a bibliography of signed graphs), and has been frequently rediscovered since. In particular, motivated by the mathematical analysis of large-scale biological networks, DasGupta et al. [3] use it to model the concept of "monotone subsystems", under the name of "sign-consistent graphs". A 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

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

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. The task of decomposing a network into monotone subsystems is then formulated as the graph modification problem Balanced Subgraph, called Undirected Labeling Problem by DasGupta et al. [3]. This problem also finds numerous other applications, e. g., in statistical physics and integrated circuit fabrication techniques [2, 22].

⋆ 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), and project ITKO (iterative compression for solving hard network problems, NI 369/5).

Given an undirected graph $G = (V, E)$ with edges labeled by $h : E \rightarrow \{=, \neq\}$, the Balanced Subgraph problem is to find a balanced subgraph with maximum number of edges. Balanced Subgraph is a generalization of the NP-hard Maximum Cut problem in graphs. Based on semidefinite programming for Max2SAT [20], DasGupta et al. [3] developed an approximation algorithm that guarantees a solution with at least 87.9% of the number of edges of an optimal solution. They further showed that this approximation factor cannot be improved arbitrarily.

In several applications, one may assume that only a small fraction of the graph edges has to be omitted; therefore, it seems even more attractive to study the formulation as a minimization problem: minimize the number of edges that have to be deleted to make the graph balanced. The special case where all edges are labeled by $\neq$ is the Edge Bipartization problem, which asks for the minimum number of edges to delete to make a graph bipartite. Edge Bipartization, also known as (unweighted) Minimum Uncut, is MaxSNP-hard. Here, the best known approximation algorithm finds in polynomial time a solution of size $O(k \log k)$, where $k$ is the size of an optimal solution [1]. It has been conjectured that it is NP-hard to improve the $\log k$-factor to a constant [15]. Hence, with respect to practical applications using a polynomial-time approximation algorithm of the minimization version seems of hardly any help.

By way of contrast, we have good news from the field of fixed-parameter algorithmics [4, 6, 16]. 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 function depending only on $k$. It is known that Edge Bipartization is fixed-parameter tractable with respect to the parameter $k$ as defined above. More specifically, there is an algorithm exactly solving Edge Bipartization in $O(2^k \cdot m^2)$ time ($m$ denoting the number of graph edges) [8]. For relatively small parameter values $k$, this opens the way to a viable and efficient alternative to employing approximation algorithms.

In this work, we observe that the Balanced Subgraph problem easily reduces to the Edge Bipartization problem and can thus be solved in $O(2^k \cdot m^2)$ time, where $k$ is the number of edges to delete (Sect. 3). Because this (so far only theoretically analyzed) fixed-parameter algorithm in its pure version is still not fast enough to cope with the given real-world instances, we present several tricks and techniques to tremendously speed up its running time in experiments. In this context, our main contribution is a data reduction scheme based on graph separators (Sect. 2). Along with this, we develop a gadget construction that is interesting on its own and deserves further studies from a practical as well as a theoretical side. In addition, directly manipulating the fixed-parameter algorithm [8], we found a speed-up trick that alone 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. [3]. We need similar amounts of time, but can solve them optimally. Moreover, we also experimented with synthetic data and further real-world instances to chart the border of feasibility of our algorithm.

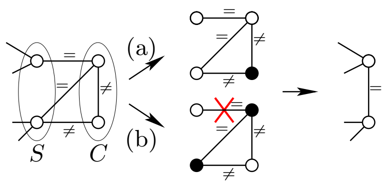


**Fig. 1.** Example for the data reduction scheme.

## 2 Data Reduction Scheme

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 [9]. The corresponding reduction rules, however, have to be developed in a problem-specific way. In this section, we describe an effective data reduction scheme for the Balanced Subgraph problem. The scheme seems to be applicable to a larger set of problems and deserves further investigation.

The idea is to find a small separator $S$ (that is, a set of vertices whose deletion separates the graph into at least two components) 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. The behavior is examined by exhaustively enumerating possible states of the separator and finding exact solutions to the small component $C$. A similar method has been suggested by Polzin and Vahdati Daneshmand [19] for the Steiner Tree problem. However, they do not employ gadgets and have no formal characterization of reducible cases.

**Reduction Scheme.** 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 Fig. 1, 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. We can simulate this behavior with a single =-edge: it also incurs a cost of 0 when the two vertices of $S$ are equally colored, 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 solving 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. We then add $m := |E|$ edges labeled $\neq$ between these two vertices. 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 our implementation, we restrict the size of $S$ to at most 4, mainly because of difficulties with the gadget construction. The size of $C$ is 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.

## 2.1 Efficiently Finding Separators

As mentioned before, we effectively only search for separators of size at most 4. Having found such a separator, we check for the (hopefully) small component that is cut off in this way.

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 [7]. 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. Separators of size 2 can also be found in linear time [13]. 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 [10]). Separators of size $k$ for small $k$ can be found efficiently by flow techniques [12]. However, after some experiments we settled for the subsequently described heuristic instead, which is faster and produces no worse results in our tests. 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 $|C| \geq 32$. Record all combinations of $S$ and $C$ with $S \leq 4$ found in this way.

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

To fix the order in which we try to reduce for an $S/C$-combination, we sort them primarily by increasing size of $S$ and secondarily by decreasing size of $C$. In this way, one deals with the tentatively best data reduction candidates first. In our experiments, the finding of separators in the above way altogether never took more than few seconds.

### 2.2 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. For lack of space, we defer the proofs to the full version of this paper.

Let us call a separator of size $i$ simply *$i$-cut*. As mentioned before, it it 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 challenge behind constructing gadgets for $i$-cuts for general $i$.

By an *optimal* gadget we refer to one with a minimum number of vertices. When speaking of an *equivalent* gadget which replaces the subgraph $G[S \cup C]$, we refer to a subgraph $H$ with the following properties: Gadget $H$ contains all vertices from $S$ and possibly more; in particular, $S$ forms the "interface" where $H$ is plugged in instead of $G[S \cup C]$. Further, 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. In particular, an optimal solution for $G$ can be directly reconstructed from an optimal solution for the modified graph.

**Gadget construction for 2-cuts.** 2-cuts generalize Example 1. Up to symmetry, there are only two colorings of the two separator vertices called $u$ 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 $S$ are considered.

Let $n_e$ be the size of an optimal solution for $G[S \cup C]$ where $u$ and $v$ have the same color 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 edges within $S$ and add $n_e - n_d$ edges labeled $\neq$ between $u$ and $v$. Otherwise, remove $C$ and 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.** *The described gadget replacement yields an equivalent instance.*

**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

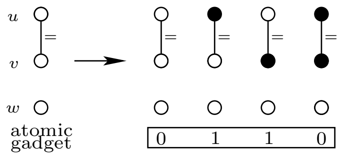


**Fig. 2.** Example for an atomic gadget with its cost vector.

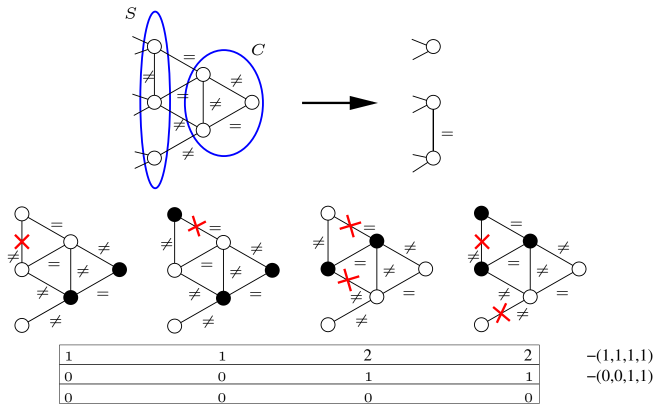
construct the final gadget from *atomic gadgets*, which can be added independently until in total they have the desired effect. To characterize the effect of an atomic gadget, we introduce the concept of a *cost vector*. In the case of 3-cuts, up to symmetry, we have four possibilities to color the separator vertices from $S$. 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 a $=$-edge between $u$ and $v$ corresponds to the cost vector $(0, 1, 1, 0)$ (see Fig. 2): 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 the two one-entries in the cost vector. Generalizing this to the five other possibilities of putting one labeled edge, we thus 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 $(=, \neq, \neq)$.

**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* $(1, 1, 1, 0)$.



| | | | | |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 | −(1,1,1,1) |
| 0 | 0 | 1 | 1 | −(0,0,1,1) |
| 0 | 0 | 0 | 0 | |

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

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 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 $S$. Even further, 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 $(1, 1, 1, 1)$. 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.
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.

We show an example in Fig. 3. On top we show the induced subgraph $G[S \cup C]$ and the gadget by which it can be replaced. In the middle we show optimal solutions for the (up to symmetry) four possible colorings of $S$ and mark by a cross the edges that have to be deleted. Cost vectors are displayed below these figures. The cost vector associated with $G[S \cup C]$ is shown in the top box. We then normalize by subtracting (1,1,1,1) and then subtracting the atomic cost vector (0,0,1,1) that corresponds to the gadget given on top, ending up with the zero-vector.

**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.*

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 and all separators with $|S| = 3$ and $|C| \geq 2$ are subject to data reduction.*

**Gadget construction for 4-cuts and outlook.** 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, $2^4$ atomic gadgets with one extra vertex, and $2^9$ atomic gadgets with two extra vertices. We then filtered out those that can be obtained by combining cheaper ones, and arrived after about five minutes of computation time 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 $S$ 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.

## 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.

We can prove the fixed-parameter tractability with respect to the parameter $k$ of Balanced Subgraph by giving a parameter-preserving reduction from Balanced Subgraph to its special case Edge Bipartization.

**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.*

*Proof.* Edge Bipartization can be solved in $O(2^k m^2)$ time [8]. It is the special case of Balanced Subgraph where all edges are labeled $\neq$. The following simple replacement transforms a Balanced Subgraph instance into an equivalent instance for Edge Bipartization: Replace every $=$-edge $\{u, v\}$ by two edges $\{u, x_{uv}\}$ and $\{x_{uv}, v\}$, where $x_{uv}$ is a new vertex. It is straightforward to show that the transformed instance has a solution of size $k$ iff the original instance has a solution of size $k$, and that from a solution of the transformed instance we can easily reconstruct a solution of the original instance. The transformation is computable in linear time and at most doubles the size of the the new instance. Hence, we obtain the same asymptotic running time as for Edge Bipartization. □

Theorem 2 improves an $O(n^{2L} \cdot (nm)^3)$ time exact algorithm by DasGupta et al. [3, 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 [14]. We inferred speedups with this trick up to a factor of about $10^{12}$. We refer to the full version of this paper for details.

## 4 Experimental Results

We applied our data reduction combined with the improved iterative compression routine to gene-regulatory networks and randomly generated graphs.

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 types. These reductions can be seen as special cases of our data reduction scheme from Sect. 2 with $|C| = 0$ and $|S| = 1$ and 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 experimental 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 in the next two sections that this cut-off value is sufficient for the considered networks.

As a further comparison point, we implemented an integer linear programming (ILP)-based approach (we omit the details). However, it was consistently outperformed by the iterative compression approach as soon as the heuristic speedup mentioned in Sect. 3 was employed.

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 the Objective Caml 3.08.3 compiler and the GNU gcc 3.3.5 compiler with options "-O3 -march=athlon". For the approximation algorithm by DasGupta et al. [3], we used MATLAB version 7.0.1.24704 (R14). Our source code is available as free software from http://theinf1.informatik.uni-jena.de/bsg/.

*Biological Networks.* We started our experimental investigations with gene regulatory networks up to the size of about 700 vertices and more than 7000 edges.

We begin with comparing our algorithm to the approximation algorithm of DasGupta et al. [3]. The authors considered the regulatory networks of yeast and human epidermal growth factor (EGFR). We additionally examine a macrophage network [17]. The results of both algorithms are given in Table 1. Apart from giving an optimal solution instead of an approximative one, we can 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

**Table 1.** Comparison of approximation [3] 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 |

**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 $c$.

| | Yeast | | | EGFR | | | Macrophage | | |
|---|---|---|---|---|---|---|---|---|---|
| $c$ | $n$ | $m$ | $t$ | $n$ | $m$ | $t$ | $n$ | $m$ | $t$ |
| 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 | ca. 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 |

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 (Theorem 2) cannot be applied directly. Furthermore, here it is remarkable that the network breaks up into several smaller components of up to 70 vertices that have to 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 $c$ of the separator $S$, we investigated stepwise for $c$ the results for the yeast, EGFR, and macrophage networks. The results are given in Table 2, where setting $c$ to $4r$ 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 $i$ by *ci-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. Regarding the macrophage network, the use of 4-cuts reduces running time severely.

We now investigate $c4$-reduction with and without cut-off value. For all networks, we could achieve the best data reduction results by using $c4r$-reduction: As mentioned above, for the yeast network the "normal" $c4$-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 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 more reduced than by applying $c3$-reduction alone. The reason that we still cannot achieve a better overall running time is the running time for the $c4r$-reduction itself. For the EGFR network, the size of the largest component does hardly change going from $c4$- to $c4r$-reduction, indicating that we do not lose much by the cut-off; in fact, we achieve a better overall running time for $c4r$. Applying $c4r$-reduction instead of $c4$-reduction to the macrophage network does not change the size of the remaining largest component, but decreases the running time from hours to seconds.

Note 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, consisting of about 100 vertices and up to 200 edges. With $c3$-reduction we could compute optimal solutions ranging from 20 to 28 in split seconds.

To end the section of regulatory networks, we describe our results for two larger networks that yet cannot be solved optimally with our method. Regarding the regulatory network for a toll-like receptor [18], 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* [5], we were less successful. The number of vertices was decreased from 628 to 500 and the number of edges from 7302 to 6845 in 25 minutes. This could be a hint that the very 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 [21]. Thereby, we tried to model the yeast network by choosing the following settings: the cluster coefficient is set to 0.016, the distribution of node-degrees is set to power-law with $\alpha = -2.2$, and the probability to assign $\neq$ to an edge is set to 0.205.

We generated 5 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 received with setting the cut-off parameter for the $c4$-rule again to 25. Redoing the test with a higher threshold of 50 did in

**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 |

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 $k = 91$. Note that 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. [3].

## 5 Outlook

There are numerous avenues for future research. DasGupta et al. [3] also introduced a directed version of the Balanced Subgraph problem. The approximation results are worse than for the undirected case, which is probably why there is no implementation yet [3]. Fortunately, the directed case can be reduced to the Vertex Bipartization problem, which can be solved in $O(3^k \cdot mn)$ time [14]. Again, this opens the route for experimental studies.

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 gadgets constructions for problems other than Balanced Subgraph. Further, it would be nice to have a formal characterization of graphs for which our separation-based data reduction scheme is useful.

*Acknowledgement.* We thank the authors of [3] for making their source code available. Further, we thank our students Tamara Steijger and Thomas Zichner for help with the experiments and Jiří Matoušek (Prague) for helpful references for the integer linear combination problem.

## References

1. A. Avidor and M. Langberg. The multi-multiway cut problem. In *Proc. 9th SWAT*, volume 3111 of *LNCS*, pages 273–284. Springer, 2004.
2. C. Chiang, A. B. Kahng, S. Sinha, X. Xu, and A. Z. Zelikovsky. 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, 2007.
3. 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*, volume 4007 of *LNCS*, pages 253–264. Springer, 2006. Extended version to appear in *Biosystems*.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
5. A. M. Feist, J. C. M. Scholten, B. Ø. Palsson, F. J. Brockman, and T. Ideker. Modeling methanogenesis with a genome scale metabolic reconstruction of *Methanosarcina barkeri*. *Molecular Systems Biology*, 2(2006.0004), 2006.
6. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
7. H. N. Gabow. Path-based depth-first search for strong and biconnected components. *Information Processing Letters*, 74(3–4):107–114, 2000.
8. J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.
9. J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, Mar. 2007.
10. C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Proc. 8th GD*, volume 1984 of *LNCS*, pages 77–90. Springer, 2000.
11. F. Harary. On the notion of balance of a signed graph. *Michigan Mathematical Journal*, 2(2):143–146, 1953.
12. M. R. Henzinger, S. Rao, and H. N. Gabow. Computing vertex connectivity: New bounds from old techniques. *Journal of Algorithms*, 43(2):222–250, 2000.
13. J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
14. F. Hüffner. Algorithm engineering for optimal graph bipartization. In *Proc. 4th WEA*, volume 3503 of *LNCS*, pages 240–252. Springer, 2005.
15. S. Khot. On the power of unique 2-prover 1-round games. In *Proc. 34th STOC*, pages 767–775. ACM Press, 2002.
16. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
17. K. Oda, T. Kimura, Y. Matsuoka, A. Funahashi, M. Muramatsu, and H. Kitano. Molecular interaction map of a macrophage. Research Report, August 2004. http://www.systems-biology.org/001/010.html.
18. K. Oda and H. Kitano. A comprehensive map of the toll-like receptor signaling network. *Molecular Systems Biology*, 2(2006.0015), 2006.
19. T. Polzin and S. Vahdati Daneshmand. Practical partitioning-based methods for the Steiner problem. In *Proc. 5th WEA*, volume 4007 of *LNCS*, pages 241–252. Springer, 2006.
20. V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.
21. E. Volz. Random networks with tunable degree distribution and clustering. *Physical Review E*, 70:056115, 2004.
22. T. Zaslavsky. Bibliography of signed and gain graphs. *Electronic Journal of Combinatorics*, DS8, 1998.