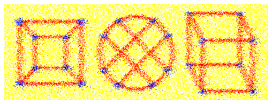
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 13, no. 2, pp. 77–98 (2009)

Algorithm Engineering for Optimal Graph Bipartization

Falk Hüffner

School of Computer Science
Tel Aviv University

Abstract

We examine exact algorithms for the NP-hard Graph Bipartization problem. The task is, given a graph, to find a minimum set of vertices to delete to make it bipartite. Based on the "iterative compression" method introduced by Reed, Smith, and Vetta in 2004, we present new algorithms and experimental results. The worst-case time complexity is improved. Based on new structural insights, we give a simplified correctness proof. This also allows us to establish a heuristic improvement that in particular speeds up the search on dense graphs. Our best algorithm can solve all instances from a testbed from computational biology within minutes, whereas established methods are only able to solve about half of the instances within reasonable time.

Submitted:	Accepted:	Final:	Published:
November 2007	August 2008	September 2008	February 2009
Article type:		Communicated by:	
Regular paper		D. Wagner	

Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4, and the Edmond J. Safra foundation. A preliminary version appeared in *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA '05)*, LNCS vol. 3503, pages 240–252, Springer, 2005.
E-mail address: hueffner@tau.ac.il (Falk Hüffner)

1 Introduction

There has been a long history of exponential-time algorithms for finding optimal solutions to NP-hard problems [37, 15]. Exponential running time at first glance seems to be impractical. This conception has been challenged by the view of *parameterized complexity* [12, 14, 30, 23]. The idea is to accept the seemingly inevitable combinatorial explosion, but to confine it to one aspect of the problem, the *parameter*. If for relevant inputs this parameter remains small, then even large instances can be solved efficiently. Problems for which this confining is possible are called *fixed-parameter tractable*.

The problem we focus on here is Graph Bipartization, also known as Maximum Bipartite Subgraph or Odd Cycle Transversal. By the general results on vertex deletion problems, Vertex Bipartization is NP-hard [25] and MaxSNP-hard [26]. The best known approximation is by a factor of $O(\log n)$ [17]. It has numerous applications, for example in VLSI design [6, 24], linear programming [19], computational biology [34, 31, 38], register allocation [39], and RFID reader networks [9].

In a breakthrough paper, Reed et al. [33] proved that the Graph Bipartization problem on a graph with n vertices and m edges is solvable in $O(4^k \cdot kmn)$ time, where k is the number of vertices to delete. The key idea is to construct size-k solutions from already known size-$(k+1)$ solutions, the so-called *iterative compression*. This is an important theoretical result, since it implies fixed-parameter tractability for Graph Bipartization with respect to k, which was posed as an open question more than five years earlier [27]. But it is also of high practical interest for several reasons:

- The given fixed-parameter complexity promises small running times for small parameter values.
- No intricate algorithmic concepts with extensive implementation requirements or large hidden constants are used as building blocks.
- The method is capable of "compressing" any given nonoptimal solution to a smaller solution. Therefore, it can be used to optimize solutions found by any known or new heuristic.

We note that building on the iterative compression algorithm, Raman et al. [32] gave an algorithm running in $O(1.62^n)$ time. However, this is unlikely to be of practical relevance because of the exponential growth of the running time in the graph size.

Contribution. In Section 3, we give an alternative proof of the result of Reed, Smith, and Vetta. This also prepares the ground for several algorithmic improvements, both heuristic and exact in nature with respect to running time bounds, in Section 4. In Section 5, we present experimental results with real-world data (Section 5.1), simulated application data (Section 5.2), and random graphs (Section 5.3). The results demonstrate that iterative compression is

in fact a worthwhile alternative for solving Graph Bipartization in practice. Thereby, we also shed more light on the potential of iterative compression, which has already lead to novel algorithms for several other problems as well [8, 20, 21, 4, 11, 29, 5, 22].

2 Preliminaries

We consider only undirected graphs $G = (V, E)$ without self-loops or multiple edges, and set $n := |V|$ and $m := |E|$. We use $G[V']$ to denote the subgraph of G induced by the vertices $V' \subseteq V$. For a set of vertices $V' \subseteq V$, we write $G \setminus V'$ for the graph $G[V \setminus V']$. With $N(v)$, we denote the set of *neighbors* of a vertex $v \in V$, that is, $N(v) := \{w \in V \mid \{v, w\} \in E\}$. We extend this notation to sets in the natural way, that is, for $V' \subseteq V$ we let $N(V') := \bigcup_{v \in V'} N(v)$. A *vertex cut* between two disjoint vertex sets in a graph is a set of vertices whose removal disconnects these two sets in the graph.

We frequently use the following characterizations of bipartite graphs from folklore.

Fact 1 *For a graph $G = (V, E)$, the following are equivalent:*

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

Our central object of study is the following NP-hard problem.

Graph Bipartization

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset $X \subseteq V$ of vertices with $|X| \leq k$ such that each odd cycle in G contains at least one vertex from X, that is, the removal of all vertices in X from G results in a bipartite graph. We call X an *odd cycle cover*.

We investigate Graph Bipartization in the context of parameterized complexity [12, 14, 30], a general approach for tackling NP-hard problems. The idea is to determine problem parameters that can be expected to be small in certain applications, and then develop algorithms that are polynomial except for an arbitrary dependence on the parameter. More precisely, a parameterized problem is called *fixed-parameter tractable* if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function solely depending on the parameter k, not on the input size n.

2.1 Algorithms

We describe two conventional approaches and the iterative compression method.

Branch-and-Bound. Wernicke [36] presents an algorithm for Graph Bipartization based on branch-and-bound. A branching decision simply tries for some vertex v the two cases that v is in the odd cycle cover or not. The improvement over the trivial 2^n algorithm comes from the the use of good lower and upper bounds and from data reduction rules. Wernicke [36] presents some experimental results on real-world data.

Integer Linear Program. Integer Linear Programs (ILPs) are frequently used in practice to solve hard problems. The reason is that it is often easy to model the problems as ILP, and that powerful solvers are available, which profit from years of research and engineering experience. We refer to the literature [35, 7] for details.

Graph Bipartization can be formulated as an ILP as follows:

$$
\begin{aligned}
& c_1, \ldots, c_n : \text{binary variables} \quad \textit{(cover)} \\
& s_1, \ldots, s_n : \text{binary variables} \quad \textit{(side)} \\
& \text{minimize} \sum_{i=1}^{n} c_i \\
& \text{s. t. } \forall \{v, w\} \in E : (s_v \neq s_w) \vee (c_v = 1) \vee (c_w = 1)
\end{aligned}
$$

where the constraint can be expressed in canonical ILP form as

$$
\begin{aligned}
\text{s. t. } \forall \{v, w\} \in E : s_v + s_w + (c_v + c_w) \geq 1 \\
\forall \{v, w\} \in E : s_v + s_w - (c_v + c_w) \leq 1
\end{aligned}
$$

Here, a 1 in c_v models that v is part of the odd cycle cover. The variables s_v model the side of the bipartite graph that remains when deleting the vertices from the odd cycle cover. The first set of constraints enforces that for an edge either one endpoint has color 1, or the other has color 1, or one of them is in the cover. In effect, it forbids that both endpoints have color 0 while none of them is in the cover. Analogously, the second set of constraints forbids that both endpoints have color 1 while none is in the cover.

Iterative Compression. Our implementation is based on an algorithm by Reed et al. [33], which we now briefly describe; we give details in Section 3. 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. If we have such a compression routine, an algorithm for Graph Bipartization could then simply start with $X = V$ and iteratively compress the trivial odd

cycle cover X until an optimal one is found. While the basic approach is very easy, the difficulty lies in finding a compression routine with acceptable running time bounds. Reed et al. [33] give such a compression routine with running time $O(4^k \cdot km)$, where k is the size of the odd cycle cover X to be compressed. This can be used to obtain a fixed-parameter algorithm with parameter k. To obtain the desired running time, we do not start with the whole graph, but rather build it up inductively. At each step, we obtain an optimal odd cycle cover for the current partial graph. More precisely, start with $V' = \{v\}$ for some $v \in V$ and $X = \emptyset$; clearly, X is a minimum odd cycle cover for $G[V']$. Now add one vertex $v' \notin V'$ from V to both V' and X. Then X is still an odd cycle cover for $G[V']$, although possibly not a minimum one. We can, however, obtain a minimum one by applying our compression routine. This process is repeated until $V' = V$. The overall running time is then $O(4^k \cdot kmn)$.

As an alternative to the inductive mode of building up a solution, the compression routine can also be employed in a more straight-forward manner by simply trying to compress an initial heuristic solution (e.g., from Abdullah [1] or Wernicke [36]) until it cannot be compressed anymore. However, this leads to a much worse combinatorial explosion: even if there was a factor-c approximation, the running time would be $O(4^{ck}) \cdot n^{O(1)}$.

2.2 Applications

A recent application for Graph Bipartization is in register allocation for processors that, to save wiring, have their register set divided into two banks and require the two operands of an instruction to reside in different banks [39]. Conflicts are modeled by a graph where vertices correspond to operands and edges connect operands that occur together in an instruction. A minimum graph bipartization set then yields the minimum size set of operands that have to be copied into both banks to be able to execute the code.

Another application originates from computational biology. To determine gene sequences, for technical reasons the DNA is first broken into small fragments (*shotgun sequencing*), from which the original sequence is reconstructed by computer. This is complicated by the fact that each gene occurs twice in the human genome. The two copies are mostly identical, but differ at certain sites (so-called SNPs). Given a set of gene fragments, the problem of assigning the fragments to one of these two copies in a consistent manner while dismissing the least number of SNPs as erroneous is called the Minimum Site Removal problem [31, 38]. The Minimum Site Removal problem can be solved using Graph Bipartization algorithms. We evaluate our algorithm in this setting with synthetic data in Section 5.

3 Graph Bipartization by Iterative Compression

In this section, we give an alternative presentation of the algorithm for Graph Bipartization by Reed et al. [33]. As opposed to their proof, our presentation

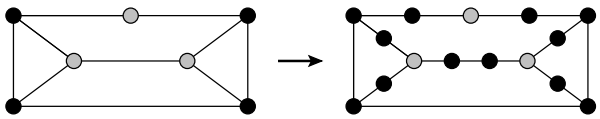

Figure 1: Input transformation for Compress-OCC. The *grey vertices* are the elements of the odd cycle cover X.

does not employ case distinction or contradiction. This also allows us to establish several improvements in Section 4. Our presentation resembles that by Guo et al. [20] for the related problem Edge Bipartization.

The global structure is illustrated by the function Odd-Cycle-Cover. It takes as input an arbitrary graph and returns a minimum odd cycle cover.

```
Odd-Cycle-Cover(G = (V, E))
1  V' ← ∅
2  X ← ∅
3  for each v ∈ V:
4      V' ← V' ∪ {v}
5      X ← X ∪ {v}
6      X ← Compress-OCC(G[V'], X)
7  return X
```

Here, the routine Compress-OCC takes a graph G and an odd cycle cover X for G, and returns a smaller odd cycle cover for G if there is one; otherwise, it returns X unchanged. Therefore, it is a loop invariant that X is a minimum-size odd cycle cover for $G[V']$, and since eventually $V' = V$, we obtain an optimal solution for G.

It remains to implement Compress-OCC. The key idea is to further restrict the search space for a smaller odd cycle cover X' by assuming several additional properties, in particular that X' is disjoint from X. Then, every odd cycle contains both a vertex from X and a vertex from X', a fact that can eventually be used to find X' as a vertex cut.

The properties are:

Property 1 *No two vertices from X are neighbors, that is, $\forall u, v \in X : u \notin N(v)$.*

Property 2 *No vertex in a smaller odd cycle cover X' is neighbor of a vertex in X, that is, $N(X') \cap X = \emptyset$.*

Property 3 *X' is disjoint from X, that is, $X \cap X' = \emptyset$.*

Properties 1 and 2 can be easily obtained by a simple input transformation: subdivide each edge adjacent to a vertex in X by a new vertex (see Figure 1). This is done successively, that is, edges connecting two vertices from X are subdivided by two vertices. This transformation preserves the parity of the

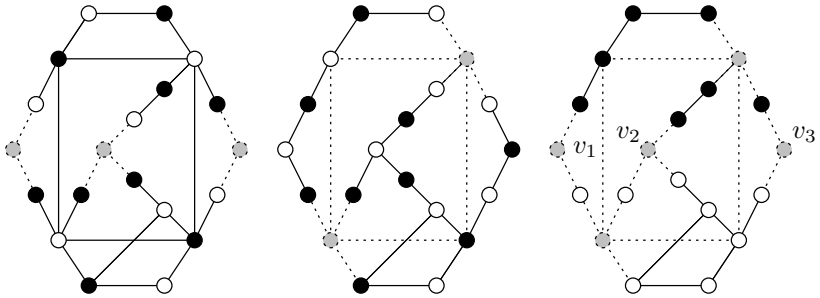

Figure 2: Comparing disjoint odd cycle covers: (a) a graph G with an odd cycle cover X (*grey vertices*); a two-coloring C_X of $G \setminus X$ is marked with *black and white vertices*; (b) another odd cycle cover X' of G with Properties 1–3, and a corresponding two-coloring $C_{X'}$; (c) the comparison function Φ.

length of any cycle C, since for each vertex in C that is in X, two new vertices are inserted into the edges of C. Therefore, after this transformation, X is still an odd cycle cover, and any odd cycle cover for the transformed graph can easily be converted to an odd cycle cover of the same size for the original graph. The transformation allows us to assume without loss of generality that no vertex v in X' is neighbor of a vertex in X (Property 2). This is because any vertex v in an odd cycle cover X' that is neighbor of a vertex in X must be one of the newly inserted degree-2 vertices, and can be replaced by the neighbor of v that is not in X, leading to a solution of the same size.

In contrast, Property 3 comes at a higher price: we use a brute-force enumeration of all 2^k partitions of the given solution X into two sets Y and $X \setminus Y$. For each partition, we then assume that the smaller solution contains all of $X \setminus Y$, but none of Y. Clearly, if we differentiate these $2^{|X|}$ cases, at least once the assumption is correct. Given a case with a particular partition, we can simplify the instance by deleting the vertices in $X \setminus Y$, since they were already determined to be part of the smaller solution. The task then remains to find an odd cycle cover that is smaller than Y, for which we can now assume Property 3.

Intuitively, not allowing the reuse of elements already in a known solution is a quite strong restriction, cutting down the space of possible smaller solutions considerably. This step is also crucial in all other iterative compression based algorithms [8, 20, 4, 11, 29, 5, 22].

At the cost of a factor of 2^k in the running time, the task is thus boiled down to:

Task 1 (Disjoint Compression) *Given a graph G with an odd cycle cover X with Property 1, find a smaller odd cycle cover X' with Properties 2 and 3 for G, or prove that there is no such X'.*

The key to solving Disjoint Compression is to compare the two two-

colorings of G induced by X and the (yet unknown) X' (see Figure 2): some vertices will have the same color in both colorings, and others will get different colors. More precisely, let C_X and $C_{X'}$ be fixed two-colorings of $G \setminus X$ and $G \setminus X'$, respectively, and define the *comparison function*

$$\Phi : V \setminus (X \cup X') \to \{\bigcirc, \bullet\} : v \mapsto \begin{cases} \bigcirc & \text{if } C_X(v) = C_{X'}(v); \\ \bullet & \text{if } C_X(v) \neq C_{X'}(v). \end{cases} \tag{1}$$

The decisive property of Φ is given in the following lemma and illustrated in Figure 2 (c).

Lemma 1 *In the setting of* Disjoint Compression*, the set $X \cup X'$ is a vertex cut between the vertex sets $\bigcirc_\Phi := \Phi^{-1}(\bigcirc)$ and $\bullet_\Phi := \Phi^{-1}(\bullet)$.*

Proof: Consider an edge $\{v, w\} \in E$ with $v, w \in V \setminus (X \cup X')$. Since C_X and $C_{X'}$ are two-colorings, we have $C_X(v) \neq C_X(w)$ and $C_{X'}(v) \neq C_{X'}(w)$. Thus, $\Phi(v) = \Phi(w)$, that is, Φ is constant along any edge that has no endpoint in $X \cup X'$. Consequently, there can be no path between two vertices with different values of Φ that does not contain a vertex from X or X'. □

Lemma 1 naturally suggests obtaining X' from a vertex cut, which is a polynomial-time task. However, we do not know the value of Φ yet, since it depends on X'. But as we will see, it suffices to guess a small part of Φ by brute force.

For this, consider the value of Φ for the neighbors of some vertex $v \in X$. Because of Properties 1 and 2, no neighbor of v is in X or in X', so Φ is defined for all neighbors of v. Further, since neither v (by Property 3) nor its neighbors are in X', the value of $C_{X'}$ is equal for all of v's neighbors. Therefore, there are only two possibilities: for all $w \in N(v) : \Phi(w) = C_X(w)$, or for all $w \in N(v) : \Phi(w) \neq C_X(w)$. Figure 2 (c) shows an example: for all neighbors w of v_1 and v_2, we have $\Phi(w) \neq C_X(w)$, and for all neighbors w of v_3, we have $\Phi(w) = C_X(w)$

This motivates the following definition.

Definition 1 *Consider a graph G and an odd cycle cover X for G, with C_X being a fixed two-coloring of $G \setminus X$. Then a coloring $\Psi : N(X) \to \{\bigcirc, \bullet\}$ is called* valid *when for all $v \in X$ either $\forall w \in N(v) : \Psi(w) = C_X(w)$ or $\forall w \in N(v) : \Psi(w) \neq C_X(w)$.*

Thus, there are $2^{|X|}$ valid colorings. We can now state the central lemma of this section.

Lemma 2 *In the setting of* Disjoint Compression*, for a vertex set $X' \subseteq V$, the following are equivalent:*

(1) X' is an odd cycle cover for G.

(2) There is a valid coloring Ψ of $N(X)$ such that X' is a vertex cut between $\bigcirc_\Psi := \Psi^{-1}(\bigcirc)$ and $\bullet_\Psi := \Psi^{-1}(\bullet)$ in $G \setminus X$.

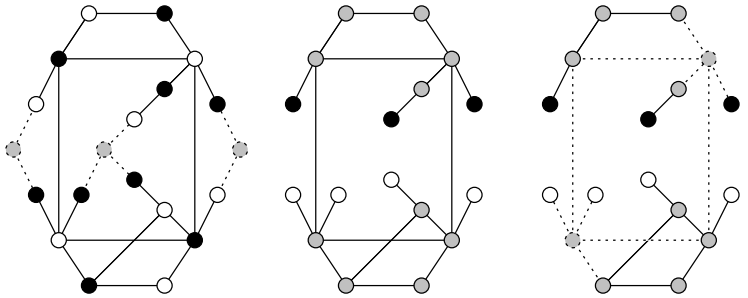

Figure 3: Illustration of the algorithm for solving Disjoint Compression (Compress-OCC): (a) Graph G with odd cycle cover X (*grey vertices*); (b) $G \setminus X$ with a valid coloring Ψ (*black and white vertices*); (c) a vertex cut X' (*dashed vertices*) between the black and the white vertices is an odd cycle cover for G.

Proof: (2) $\Rightarrow$ (1): Consider a vertex set C that induces an odd cycle in G. It suffices to show that $C \cap X' \neq \emptyset$. Since X is an odd cycle cover, there is at least one vertex from X in C. For at least one vertex $v \in C \cap X$, its two cycle neighbors v_l and v_r on C have different colors in C_X, that is, $C_X(v_l) \neq C_X(v_r)$; otherwise, we could two-color the odd cycle C, since no two vertices from X are neighbors by Property 1. By the definition of a valid coloring, this implies $\Psi(v_l) \neq \Psi(v_r)$. Since X' is a vertex cut in $G \setminus X$ between the differently colored vertices v_l and v_r, there must be some $v' \in X'$ with $v' \in C$.

(1) $\Rightarrow$ (2): As argued above, $\Psi := \Phi|_{N(X)}$ (that is, Φ restricted to the neighbors of vertices from X) is a valid coloring, and by Lemma 1 X' is a vertex cut between $\bigcirc_\Psi$ and $\bullet_\Psi$ in $G \setminus X$. □

With Lemma 2, it is now clear that we can solve Disjoint Compression by trying all $2^{|X|}$ valid colorings Ψ and determining a minimum vertex cut between $\bigcirc_\Psi$ and $\bullet_\Psi$. We now have everything in place to present the compression routine.

Compress-OCC(G_0, X_0)
1 subdivide edges around each $v \in X_0$
2 **for each** $X \subseteq X_0$**:**
3 $\quad G \leftarrow G_0 \setminus (X_0 \setminus X)$
4 $\quad$ **for each** valid coloring Ψ of $N(X)$**:**
5 $\quad\quad$ **if** $\exists$ vertex cut D in $G \setminus X$ between $\bigcirc_\Psi$ and $\bullet_\Psi$ with $|D| < |X|$**:**
6 $\quad\quad\quad$ **return** $(X_0 \setminus X) \cup D$
7 **return** X_0

We now explain Compress-OCC in detail. Given is a graph G_0 with an odd cycle cover X_0. First we ensure Properties 1 and 2 by a simple input

transformation (line 1; see Figure 1). We then examine every subset X of the known odd cycle cover X_0 (line 2). For each X, we look for smaller odd cycle covers for G that can be constructed by replacing the vertices of X in X_0 by fewer new vertices from $V \setminus X$ (clearly, for any smaller odd cycle cover, such an X must exist). Since we thereby decided to retain the vertices in $X_0 \setminus X$ in our odd cycle cover, we examine the graph $G = G_0 \setminus (X_0 \setminus X)$ (an example is shown in 3 (a)). After line 3, we have Properties 1 and 3 for G and X. If we now find an odd cycle cover D for G with $|D| < |X|$, we are done, since then $(X_0 \setminus X) \cup D$ is an odd cycle cover smaller than X_0 for G_0. For this, we try all valid colorings for $N(X)$ (Figure 3 (b) shows one example). By Lemma 2, there is some valid coloring where a smaller odd cycle cover forms a vertex cut between $\bigcirc_\Phi$ and $\bullet_\Phi$ in $G \setminus X$; and moreover, any such cut is an odd cycle cover. Therefore, if we find a vertex cut D between $\bigcirc_\Phi$ and $\bullet_\Phi$ that is smaller than X, we are done (see Figure 3 (c)); conversely, if no valid coloring is successful, it is not possible to compress X.

Running Time. Reed et al. [33] state the running time of their algorithm as $O(4^k \cdot kmn)$; a slightly more careful analysis reveals it as $O(3^k \cdot kmn)$. For this, note that in effect the two loops in lines 2 and 4 of Compress-OCC iterate over all possible assignments of each $v \in X_0$ to three roles:

- either $v \in X_0 \setminus X$,
- or $\Psi(w) = C_X(w)$ for all neighbors w of v,
- or $\Psi(w) \neq C_X(w)$ for all neighbors w of v.

Therefore, we solve 3^k minimum vertex cut problems, and since we can solve one minimum vertex cut problem in $O(km)$ time by the Edmonds–Karp algorithm [10, 13, 7], the running time for one invocation of Compress-OCC is $O(3^k \cdot km)$. As Odd-Cycle-Cover calls Compress-OCC n times, we arrive at an overall running time of $O(3^k \cdot kmn)$.

Theorem 1 Graph Bipartization *can be solved in* $O(3^k \cdot kmn)$ *time.*

4 Algorithmic Improvements

In this section, we present several improvements over the algorithm by Reed et al. [33]. We start with two simple improvements that save a constant factor in the running time. In Section 4.2 we then show how to save a factor of k in the running time by exploiting the similarity of the subproblems solved. Finally, in Section 4.3 we present an improvement exploiting the structure of the subgraph induced by the bipartization set. This improvement gave the most pronounced speedups in our experiments presented in Section 5.

4.1 Simple Improvements

It is easy to see that for each valid coloring Ψ there is a symmetric coloring where the value is inverted at each vertex, leading to the same vertex cuts. Therefore we can arbitrarily fix the allocation of the neighbors of one vertex, saving a factor of 2 in the running time.

The next improvement is justified by the following lemma.

Lemma 3 *Consider a graph $G = (V, E)$, a vertex $v \in V$, and a minimum-size odd cycle cover X for $G \setminus \{v\}$ with $|X| = k$. Then no odd cycle cover of size k for G contains v.*

Proof: If X' is an odd cycle cover of size k for G, then $X' \setminus \{v\}$ is an odd cycle cover of size $k-1$ for $G[V \setminus \{v\}]$, contradicting that X is of minimum size. □

With Lemma 3 it is clear that the vertex v we add to X in line 5 of Odd-Cycle-Cover cannot be part of a smaller odd cycle cover, and we can omit the case $v \notin X$ in Compress-OCC, saving a third of the cases.

4.2 Exploiting Subproblem Similarity

In the "inner loop" of Compress-OCC (line 5), we need to find a minimum size vertex cut between two vertex sets in a graph. This is a classic application for maximum flow techniques: The well-known max-flow min-cut theorem [7] tells us that the size of a minimum edge cut is equal to the maximum flow. Since we are interested in vertex cuts, we create a new, directed graph G' for our input graph $G = (V, E)$: for each vertex $v \in V$, create two vertices v_{in} and v_{out} and a directed edge (v_{in}, v_{out}). For each edge $\{v, w\} \in E$, we add two directed edges (v_{out}, w_{in}) and (w_{out}, v_{in}). It is not hard to see that a maximum flow in G' between $Y_1' := \bigcup_{y \in Y_1} y_{in}$ and $Y_2' := \bigcup_{y \in Y_2} y_{out}$ corresponds to a maximum set of vertex disjoint paths between Y_1 and Y_2. Furthermore, an edge cut D between Y_1' and Y_2' is of the form $\bigcup_{v \in V} \{(v_{in}, v_{out})\}$, and $\bigcup_{(v_{in}, v_{out}) \in D} \{v\}$ is a vertex cut between Y_1 and Y_2 in G.

Since we know that the cut is relatively small (less than or equal to k), we employ the Edmonds–Karp algorithm [10, 13, 7]. This algorithm repeatedly finds a shortest augmenting path in the flow network and increases the flow along it, until no further increase is possible. We assume in the rest of this section that the reader is familiar with this algorithm.

The idea is then that the flow problems solved in Compress-OCC are "similar" in such a way that we can "recycle" the flow networks for each problem. For this, after line 3 of Compress-OCC, we merge all white neighbors of each $v \in X$ into a single vertex v_1, and all black neighbors of each $v \in X$ into a single vertex v_2. Clearly, this does not change the minimum cut found in line 5. Then, each flow problem corresponds to one assignment of the vertices in X to the three roles "v_1 source, v_2 target", "v_2 source, v_1 target", and "not present" ($v \notin X$). Using a so-called $(3, k)$-ary Gray code [18], we can enumerate these assignments in such a way that adjacent assignments differ in only one element.

For each of these (but the first one), one can solve the flow problem by adapting the previous flow. One or both of the following actions is needed:

- If the vertex v whose assignment was changed was present previously, drain the flow along the path with end point v_1 and the path with end point v_2 (note that they might be identical). Here, "drain the flow" means to find an augmenting path in the flow network (as opposed to the residual network), and zero the flow along this path.
- If v is present in the updated assignment, find an augmenting path from v_1 to v_2 or from v_2 to v_1, depending on the current role of v.

Lemma 4 *The above procedure correctly updates a maximum flow.*

Proof: Setting the flow to zero along (both directions of) an augmenting path in the flow network clearly does not violate capacity constraints or skew symmetry; since each vertex on the path except for the start- and endvertex has one incoming edge with value 1 and one incoming edge with value -1 cleared, flow conservation is also preserved. Therefore, the draining procedure yields a valid flow where v_1 and v_2 are not source or sink anymore. Augmenting a path is well-known to provide a valid flow where the endpoints are source resp. target. Further, the new flow is maximum: if previously there were k' sources, the flow had value k', since otherwise the algorithm would have terminated. Since we add at most one new source, the new flow can be at most $k' + 1$, and therefore a single augmentation operation suffices to get a maximum flow. □

Since each of these operations can be done in $O(m)$ time, we can perform the update in $O(m)$ time, as opposed to $O(km)$ time for solving a flow problem from scratch. This improves the overall worst case running time to $O(3^k \cdot mn)$. We call this algorithm OCC-Gray.

Theorem 2 Graph Bipartization *can be solved in* $O(3^k \cdot mn)$ *time.*

4.3 Filtering of Valid Colorings

Lemma 2 tells us that for Disjoint Compression, there is a valid coloring for $N(X)$ such that we will find a cut leading to a smaller odd cycle cover. Therefore, simply trying all valid colorings will be successful. However, a more careful examination allows to omit some valid colorings from consideration. For this, consider two vertices $c, d \in X$ that are connected by an edge. After the input transformation, they are connected by a path containing two fresh vertices v_c and v_d. If we now have a valid coloring that assigns different values to v_c and v_d, it is not possible to find a vertex cut that disconnects them, since they are directly connected by an edge. Therefore, any valid coloring that is to be successful must assign them the same colors.

For notational convenience, we now identify a valid coloring Ψ with a coloring C_Ψ of the vertices in X. We identify the choice $\forall w \in N(v) : \Psi(w) = C_X(w)$ with painting v white and the choice $\forall w \in N(v) : \Psi(w) \neq C_X(w)$ with

painting v black. Then, the above observation imposes $C_\Psi(v) \neq C_\Psi(w)$ for all $\{v, w\} \in E$. This means that C_Ψ is a two-coloring of $G[X]$. If $G[X]$ is not bipartite, we can immediately give up trying to find a smaller odd cycle cover; otherwise, we only have to try all two-colorings of $G[X]$ (there can be more than one if $G[X]$ is disconnected). This leads to the following algorithm.

COMPRESS-OCC-ENUM2COL(G_0, X_0)
1 subdivide edges around each $v \in X$
2 **for each** bipartite subgraph B of $G[X]$**:**
3 **for each** two-coloring C_Ψ of B**:**
4 **if** $\exists$ vertex cut D in $G \setminus X$ between $\circ_\Psi$ and $\bullet_\Psi$ with $|D| < |X|$**:**
5 **return** $(X_0 \setminus X) \cup D$
6 **return** X_0

The worst case for COMPRESS-OCC-ENUM2COL is that X is an independent set in G. In this case, every subgraph of $G[X]$ is bipartite and has $2^{|X|}$ two-colorings. This leads to exactly the same number of flow problems solved as for COMPRESS-OCC. In the best case, X is a clique, and $G[X]$ has only $O(|X|^2)$ bipartite subgraphs, each of which admits (up to symmetry) only one two-coloring.

It is easy to construct a graph where any optimal odd cycle cover is independent; therefore the described modification does not lead to an improvement of the worst-case running time. However, at least in a dense graph, it is "unlikely" that the odd cycle covers are completely independent, and already a few edges between vertices of the odd cycle cover can vastly reduce the required computation.

With a simple branching strategy, one can enumerate all bipartite subgraphs of a graph and all their two-colorings with constant cost per two-coloring. This can also be done in such a way that modifications to the flow graph can be done incrementally, as described in Section 4.2. The two simple improvements mentioned at the beginning of this section also can still be applied. We call the thus modified algorithm OCC-ENUM2COL.

It seems plausible that for dense graphs, an odd cycle cover is "more likely" to be connected, and therefore this heuristic is more profitable. Experiments on random graphs confirm this (see Section 5.3). This is of particular interest because other strategies (such as reduction rules [36]) seem to have a harder time with dense graphs than with sparse graphs, making hybrid algorithms appealing.

5 Experiments

We first evaluated the conventional approaches. The ILP performed quite well; when solved by GNU GLPK [28], it consistently outperformed the highly problem-specific branch-and-bound approach by Wernicke [36] on our test data, sometimes by several orders of magnitude. Therefore, we use the ILP as the

comparison point for the performance of our algorithms and do not give details on the branch-and-bound approach.[1]

Implementation Details. The program is written in the C programming language and consists of about 1400 lines of code. The source and the test data are available from http://theinf1.informatik.uni-jena.de/occ/.

Data structures. Over 90% of the time is spent in finding an augmenting path within the flow network; all that this requires from a graph data structure is enumerating the neighbors of a given vertex. The only other frequent operation is "enabling" or "disabling" vertices as determined by the Gray code (see Section 4.2). In particular, it is not necessary to quickly add or remove edges, or query whether two vertices are neighbors. Therefore, we chose a very simple data structure, where the graph is represented by an array of neighbor lists, with a null pointer denoting a disabled vertex.

Since the flow simply models a set of vertex-disjoint paths, it is not necessary to store a complete $n \times n$-matrix of flows; it suffices to store the flow predecessor and successor for each node, reducing memory usage to $O(n)$.

Experimental Setup. We tested our implementation on various inputs. The testing machine is an AMD Athlon 64 3700+ with 2.2 GHz, 1 MB cache, and 1 GB main memory, running under the Debian GNU/Linux 3.1 operating system. The source was compiled with the GNU gcc 3.3.4 compiler with option "-O3". Memory requirements are around 3 MB for the iterative compression based algorithms and up to 500 MB for the ILP.

5.1 Minimum Site Removal

The first test set originates from computational biology (see Section 2.2). The instances were constructed by Wernicke [36] from data of the human genome as a means to solve the so-called Minimum Site Removal problem. We examine these instances to learn about the performance of our algorithms on real-world instances, in particular those modeling a data correction task (the graph should be bipartite, but is distorted, and a most parsimonious reconstruction is sought). The results are shown in Table 1.

As expected, the running times of the iterative compression algorithms mainly depend on the size of the odd cycle cover that is to be found. Interestingly, the ILP also shows this behavior; the probable explanation is that it takes the branch-and-bound part of the solver longer to establish inferiority of an assignment of the integer variables to an upper bound found previously. The observed improvement in the running time from "Reed" to "OCC-Gray" is slightly lower than the factor of k gained in the worst-case complexity, but

[1] We recently found that more sophisticated mathematical programming approaches have been suggested [16], although without provable running time guarantees. It would be interesting to extend our comparison to these.

	n	m	density	$\lvert X\rvert$	ILP	Reed	OCC-Gray	OCC-Enum2Col
Afr. #31	30	51	11.7	2	0.02	0.00	0.00	0.00
Jap. #19	84	172	4.9	3	0.23	0.00	0.00	0.00
Jap. #24	142	387	3.9	4	1.30	0.00	0.00	0.00
Jap. #11	51	212	16.6	5	0.50	0.00	0.00	0.00
Afr. #10	69	191	8.1	6	3.49	0.00	0.00	0.00
Afr. #36	111	316	5.2	7	16.93	0.01	0.00	0.00
Jap. #18	71	296	11.9	9	73.94	0.09	0.02	0.00
Jap. #17	79	322	10.5	10	100.93	0.27	0.04	0.00
Afr. #11	102	307	6.0	11	3790.99	1.10	0.14	0.02
Afr. #54	89	233	5.9	12		5.13	0.61	0.10
Afr. #34	133	451	5.1	13		9.36	0.98	0.02
Afr. #52	65	231	11.1	14		20.95	2.08	0.02
Afr. #22	167	641	4.6	16		318.95	31.24	0.15
Afr. #48	89	343	8.8	17		1269.01	104.20	0.11
Afr. #50	113	468	7.4	18		5287.68	501.15	0.06
Afr. #19	191	645	3.6	19			1288.85	2.23
Afr. #45	80	386	12.2	20			2774.75	0.23
Afr. #29	276	1058	2.8	21				0.38
Afr. #40	136	620	6.8	22				0.98
Afr. #39	144	692	6.7	23				9.11
Afr. #17	151	633	5.6	25				49.27
Afr. #38	171	862	5.9	26				2.60
Afr. #28	167	854	6.2	27				2.43
Afr. #42	236	1110	4.0	30				78.79
Afr. #41	296	1620	3.7	40				175.14

Table 1: Running times in seconds for different algorithms for Wernicke's benchmark instances [36]. Runs were cancelled after 2 hours without result. We show only the instance of median size for each value of $|X|$. The column "ILP" gives the running time of the ILP given in Section 2 when solved by GNU GLPK [28]. The column "Reed" gives the running time of Reed et al.'s algorithm without any of the algorithmic improvements from Section 4 except for the obvious improvement of omitting symmetric valid partitions. The columns "OCC-Gray" and "OCC-Enum2Col" give the running time for the respective algorithms from Sections 4.2 and 4.3.

clearly still worthwhile. The heuristic from Section 4.3 works exceedingly well and allows to solve even the hardest instances within minutes. In fact, for almost all instances that the ILP was able to solve at all, the running time was below the timer resolution of 10 ms.

For both improvements, the savings in running can be completely explained by the reduced number of flow augmentations.

5.2 Synthetic Data from Computational Biology

In this section, we examine solving the Minimum Fragment Removal problem [31] with Graph Bipartization. The motivation is similar as in Section 5.1, except that the goal is to remove the minimum number of fragments (presumably those that contain read errors) to obtain consistent data.

c	$\|V\|$	$\|E\|$	$\|X\|$	ILP	Reed	OCC-Gray	OCC-Enum2Col
2	25	23	1.4	0.02	0.00	0.00	0.00
3	50	61	3.5	2.00	0.00	0.00	0.00
4	82	116	6.1	224.33	0.05	0.00	0.00
5	112	173	8.3		1.17	0.05	0.02
6	143	253	10.4		39.86	1.64	0.16
7	174	321	11.6		7245.40	254.10	1.10
8	211	431	14.8			627.84	4.54
9	243	561	17.9				181.48
10	289	710	21.0				186.36
11	328	839	23.3				2493.82

Table 2: Running times in seconds for different algorithms for synthetic Minimum Fragment Removal instances [31]. Here, c is a model parameter. Each entry is an average over 20 instances.

We generate synthetic Graph Bipartization instances using a model suggested by Panconesi and Sozio [31]. A random binary string h_1 of length n is generated, and h_2 is generated as a copy of h_1 where a proportion of d bits is randomly flipped. These strings represent the two copies of the haplotype. Next, fragments are generated by breaking h_1 and h_2 each into k pieces by selecting $k-1$ breakpoints randomly. The fragment generation process is repeated c times, such that every position in h_1 or h_2 occurs c times in some fragment. Finally, each fragment is mutated by flipping each bit with probability p. From the fragments, the conflict graph is constructed, where the fragments are the vertices and an edge is drawn between two fragments if they differ at some position.

Following Panconesi and Sozio [31], we choose the parameters $n = 100$, $d = 0.2$, $k = 20$, $p = 0.02$, and c varying (see Table 2).

The results are consistent with those of Section 5.1. The ILP is outperformed by the iterative compression algorithms; for OCC-Gray, we get a speedup by a factor somewhat below $|X|$ when compared to "Reed". The speedup from employing OCC-Enum2Col is very pronounced, but still far below the speedup observed in Section 5.1. A plausible explanation is the lower average vertex degree of the input instances; we examine this further in Section 5.3. Note that even with all model parameters constant, running times varied by a factor of up to several orders of magnitude for all algorithms for different random instances.

5.3 Random Graphs

The previous experiments have established OCC-Enum2Col as best performing algorithm. Therefore, we now focus on charting its tractability border. We use the following method to generate random graphs with given number of vertices n, edges m, and odd cycle cover size at most k: Pre-allocate the roles "black" and "white" to $(n-k)/2$ vertices each, and "odd cycle cover" to k vertices; select a random vertex and add an edge to another random vertex consistent with the roles until m edges have been added.

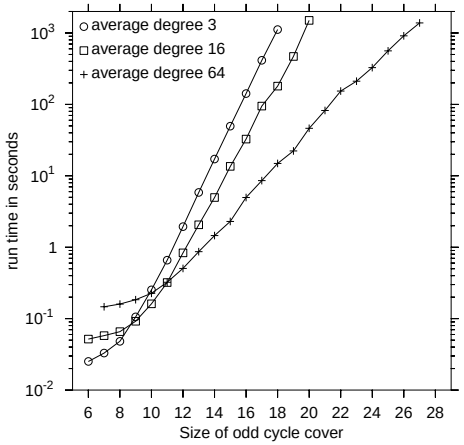

Figure 4: Running time of OCC-Enum2Col (Section 4.3) for random graphs of different density ($n = 300$). Each point is the average over at least 30 runs.

In Figure 4, we display the running time of OCC-Enum2Col for different sizes of the odd cycle cover and different graph densities for graphs with 300 vertices. Note that the actual optimal odd cycle cover can be smaller than the one "implanted" by our model; the figure refers to the actual odd cycle cover size k.

At an average degree of 3, the growth in the measurements closely matches the one predicted by the worst-case complexity $O(3^k)$. For the average degree 16, the measurements fit a growth of $O(2.6^k)$, and for average degree 64, the growth within the observed range is about $O(1.6^k)$. This demonstrates the effectiveness of OCC-Enum2Col for dense graphs, at least in the range of values of k we examined.

6 Conclusions

We evaluated the iterative compression algorithm by Reed et al. [33] for Graph Bipartization and presented several improvements. The implementation performs better than established techniques, and allows to solve instances from computational biology that previously could not be solved exactly. In particular, a heuristic (Section 4.3) yielding optimal solutions performs very well on dense graphs. This result makes the practical evaluation of iterative compression for other applications [8, 20, 21, 4, 11, 29, 5, 22] appealing.

Future Work. The best way to improve the presented programs further for practical applicability seems the incorporation of data reduction rules. In particular, Wernicke [36] reports them to be most effective for sparse graphs. This makes a combination with OCC-Enum2Col (Section 4.3) attractive, since in

contrast, this algorithm displays the worst performance for sparse graphs. It would also be interesting to see whether the quest for better data reduction leads to a problem kernel, that is, a reduced instance whose size depends only on the parameter k, as it was recently achieved for the related problem Feedback Vertex Set [3, 2].

Acknowledgments

The author is grateful to Jens Gramm (Tübingen) and Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke (Jena) for many helpful suggestions.

References

[1] A. Abdullah. On graph bipartization. In *Proc. 1992 IEEE International Symposium on Circuits and Systems (ISCAS '92)*, volume 4, pages 1847–1850, 1992.

[2] H. L. Bodlaender. A cubic kernel for feedback vertex set. In *Proc. 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS '07)*, volume 4393 of *LNCS*, pages 320–331. Springer, 2007.

[3] K. Burrage, V. Estivill-Castro, M. R. Fellows, M. A. Langston, S. Mac, and F. A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *Proc. 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *LNCS*, pages 192–202. Springer, 2006.

[4] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for the feedback vertex set problems. In *Proc. 10th Workshop on Algorithms and Data Structures (WADS '07)*, volume 4619 of *LNCS*, pages 422–433. Springer, 2007. To appear under the title "Improved algorithms for feedback vertex set problems" in *Journal of Computer and System Sciences*.

[5] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proc. 40th ACM Symposium on Theory of Computing (STOC '08)*, pages 177–186. ACM, 2008. To appear in *Journal of the ACM*.

[6] H.-A. Choi, K. Nakajima, and C. S. Rim. Graph bipartization and via minimization. *SIAM Journal on Discrete Mathematics*, 2(1):38–47, 1989.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[8] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory of Computing Systems*, 41(3):479–492, 2007.

[9] V. Deolalikar, M. R. Mesarina, J. Recker, and S. Pradhan. Perturbative time and frequency allocations for RFID reader networks. In *Proc. 2006 Workshop on Emerging Directions in Embedded and Ubiquitous Computing (EUC '06)*, volume 4097 of *LNCS*, pages 392–402. Springer, 2006.

[10] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation (in Russian). *Doklady Akademii Nauk SSSR*, 194(4), 1970. English translation in *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[11] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Proc. 6th Italian Conference on Algorithms and Complexity (CIAC '06)*, volume 3998 of *LNCS*, pages 320–331. Springer, 2006.

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

[13] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

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

[15] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87: 47–77, 2005.

[16] P. Fouilhoux and A. Ridha Mahjoub. Polyhedral results for the bipartite induced subgraph problem. *Discrete Applied Mathematics*, 154(15):2128–2149, 2006.

[17] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP '94)*, volume 820 of *LNCS*, pages 487–498. Springer, 1994.

[18] D.-J. Guan. Generalized Gray codes with applications. *Proc. National Science Council, Republic of China (A)*, 22(6):841–848, 1998.

[19] N. Gülpinar, G. Gutin, G. Mitra, and A. Zverovich. Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, 137(3):359–372, 2004.

[20] 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.

[21] F. Hüffner, N. Betzler, and R. Niedermeier. Optimal edge deletions for signed graph balancing. In *Proc. 6th Workshop on Experimental Algorithms (WEA '07)*, volume 4525 of *LNCS*, pages 297–310. Springer, 2007.

[22] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *Proc. 8th Latin American Theoretical Informatics Symposium (LATIN '08)*, volume 4598 of *LNCS*, pages 711–722. Springer, 2008.

[23] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008.

[24] A. B. Kahng, S. Vaya, and A. Z. Zelikovsky. New graph bipartizations for double-exposure, bright field alternating phase-shift mask layout. In *Proc. Asia and South Pacific Design Automation Conference*, pages 133–138, 2001.

[25] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20 (2):219–230, 1980.

[26] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *Proc. 20th International Colloquium on Automata, Languages and Programming (ICALP '93)*, volume 700 of *LNCS*, pages 40–51. Springer, 1993.

[27] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

[28] A. Makhorin. *GNU Linear Programming Kit Reference Manual Version 4.8*. Department of Applied Informatics, Moscow Aviation Institute, 2004.

[29] D. Marx. Chordal deletion is fixed-parameter tractable. In *Proc. 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG '06)*, volume 4271 of *LNCS*, pages 37–48. Springer, 2006.

[30] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[31] A. Panconesi and M. Sozio. Fast hare: A fast heuristic for single individual SNP haplotype reconstruction. In *Proc. 4th International Workshop on Algorithms in Bioinformatics (WABI '04)*, volume 3240 of *LNCS*, pages 266–277. Springer, 2004.

[32] V. Raman, S. Saurabh, and S. Sikdar. Efficient exact algorithms through enumerating maximal independent sets and other techniques. *Theory of Computing Systems*, 41(3):563–587, 2007.

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

[34] R. Rizzi, V. Bafna, S. Istrail, and G. Lancia. Practical algorithms and fixed-parameter tractability for the single individual SNP haplotyping problem. In *Proc. 2nd International Workshop on Algorithms in Bioinformatics (WABI '02)*, volume 2452 of *LNCS*, pages 29–43. Springer, 2002.

[35] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

[36] S. Wernicke. *On the Algorithmic Tractability of Single Nucleotide Polymorphism (SNP) Analysis and Related Problems*. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 2003.

[37] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Proc. 5th International Workshop on Combinatorial Optimization*, volume 2570 of *LNCS*, pages 185–208. Springer, 2003.

[38] X.-S. Zhang, R.-S. Wang, L.-Y. Wu, and L. Chen. Models and algorithms for haplotyping problem. *Current Bioinformatics*, 1(1):104–114, 2006.

[39] X. Zhuang and S. Pande. Resolving register bank conflicts for a network processor. In *Proc. 12th International Conference on Parallel Architectures and Compilation Techniques (PACT '03)*, pages 269–278. IEEE Press, 2003.