Elsevier Editorial System(tm) for Journal of Discrete Algorithms
Manuscript Draft

Manuscript Number:

Title: Finding kernels or solving SAT

Article Type: Full Length Article

Keywords: propositional logic, satisfiability, digraph kernel

Corresponding Author: Mr. michal walicki,

Corresponding Author's Institution: Institute of informatics

First Author: michal walicki

Order of Authors: michal walicki; Sjur Dyrkolbotn

Abstract: We begin by offering a new, direct proof of the equivalence between the problem of the existence of kernels in digraphs, KER, and satisfiability of propositional theories, SAT, giving linear reductions in both directions. Having introduced some linear reductions of the input graph, we present new algorithms for KER, with variations utilizing solvers of boolean equations. In the worst case, the algorithms try all assignments to either a feedback vertex set, F, or a set of nodes E touching only all even cycles. Hence KER is fixed parameter tractable not only in the size of F, as observed earlier, but also in the size of E. A slight modification of these algorithms leads to a branch and bound algorithm for KER which is virtually identical to the DPLL algorithm for SAT. This suggests deeper analogies between the two fields and the probable scenario of KER research facing the challenges known from the work on SAT.

Finding kernels or solving SAT

Michał Walicki and Sjur Dyrkolbotn
Department of Informatics, University of Bergen
michal@ii.uib.no

Abstract

We begin by offering a new, direct proof of the equivalence between the problem of the existence of kernels in digraphs, KER, and satisfiability of propositional theories, SAT, giving linear reductions in both directions. Having introduced some linear reductions of the input graph, we present new algorithms for KER, with variations utilizing solvers of boolean equations. In the worst case, the algorithms try all assignments to either a feedback vertex set, F, or a set of nodes E touching only all even cycles. Hence KER is fixed parameter tractable not only in the size of F, as observed earlier, but also in the size of E. A slight modification of these algorithms leads to a branch and bound algorithm for KER which is virtually identical to the DPLL algorithm for SAT. This suggests deeper analogies between the two fields and the probable scenario of KER research facing the challenges known from the work on SAT.

1 Introduction

The concept of a kernel of a digraph (an independent set reachable from every outside node by an edge) was introduced in [28] as a generalization of a solution of a cooperative game and has since then found applications in both positional and cooperative game theory as well as in logic. Determining the existence of a kernel has become a problem of independent interest in graph theory, starting with the classical results of Richardson, [24, 25], and followed in the last decades by several publications, e.g., [22, 13, 14, 1, 16, 11], with a recent overview [4].

The problem of the existence of kernels in digraphs, KER, is NP-complete, [6], so in a trivial sense it is equivalent to the satisfiability of propositional theories, SAT. The equivalence has been applied, e.g., in [20] for representing finitely branching dags as consistent propositional theories, in [10, 11] for studying default logic, in [12] for correlating models of logic programs and kernels of appropriate digraphs and in [29] for analysing circularity in logical paradoxes. But it has not received a separate treatment, independent from particular applications. From an algorithmic perspective, it is natural to ask for a more fine-grained analysis of the exact relationship between SAT and KER. An answer should provide an indication both as to whether kernel theory can contribute to SAT-solving, and as to how techniques developed by SAT-solvers can be employed to increase efficiency of deciding KER. Equivalence of the two problems with respect to some complexity class does not suffice to answer such questions because, in order for a reduction to be useful in practice, even constant factors may matter, requiring a more detailed analysis of the actual choices and possible heuristics.

In this article we focus on KER, showing that the reducibility of KER to SAT has a practical, algorithmic content. This is found not so much in the direct application of SAT-solvers, although this too is a viable approach for some cases, but rather in the similarities between the problems encountered while trying to solve KER (directly) and those faced by SAT-solvers. We present a series of novel algorithms for KER, utilizing new observations of graph-theoretical nature but also the possibility of solving SAT at appropriate places. These can be very efficient for some classes of graphs, but are hardly optimal in general. We then present our final algorithm for KER, which is very similar to the central SAT-algorithm DPLL, [9, 8]. We review several issues which, arising from earlier experiences with SAT, are likely to affect future work on KER.

The question of how kernel theory can be used to solve SAT more effectively is left for future work, but we hope that the connection we demonstrate here indicates strongly that SAT-solvers might indeed have something to gain from utilizing the graphical nature of KER.

Section 2 introduces the basic definitions and establishes the equivalence of KER and SAT, giving new linear reductions in both directions, simpler than previously available. The problem of finding a kernel is formulated in terms of assigning boolean values to the nodes of the graph, an assignment is a *solution* when it determines a kernel and a graph is *solvable* if it has a solution. Section 3 presents some linear (or low polynomial) graph reductions which preserve and reflect solvability and are later used by the discussed algorithms. Section 4 presents several results relating solvability to various conditions on feedback vertex sets. In Subsection 4.1 we also show how to solve KER by constructing a dag from a digraph. This is essentially the technique used in the algorithms from [10, 11]. In our case, however, a single dag suffices for either finding a kernel or concluding its non-existence. In the worst case, it tries all assignments to a feedback vertex set, and thus the complexity of the trivial brute-force $\mathcal{O}^*(2^{|G|})$ is reduced to $\mathcal{O}^*(2^{|F|})$, where $F \subseteq G$ is a feedback vertex set.[1] Following that, we show that one can reduce this factor even further to the number of even cycles only. Subsection 4.2 gives an algorithm which, for each assignment to a subset of nodes E touching all even cycles, determines in linear time if the resulting, induced assignment is a solution, thus giving the complexity $\mathcal{O}^*(2^{|E|})$. Both these algorithms show that the problem is fixed parameter tractable, FPT, taking the size of F, respectively E, as the parameter. We also discuss a variation which, instead of inducing the values along the obtained dag, decides solvability of the appropriate system of $|E|$ boolean equations over $|E|$ variables. Section 5 introduces the main, recursive algorithm, based on the simplifications introduced in Section 3. It subsumes the algorithm from [11] as a special case and allows to show the complexity bound $\mathcal{O}^*(1.325^{|G|})$ for oriented graphs (with no 2-cycles). It turns out that, except for the fact that it works on digraphs and not on CNFs, it is exactly the DPLL algorithm – the basis of most modern SAT-solvers. This brings a new aspect of the relationship to SAT, and we conclude listing a series of conjectures and hypotheses on the expected issues and choices in the further development of the algorithms for KER, originating from the experiences with SAT-solvers.

2 Background

A *digraph* (*directed graph*) is a pair $\mathsf{G} = \langle G, E\rangle$, where G is a finite set of nodes and $E \subseteq G \times G$ is a binary relation that describes the directed edges of G.[2]

For a vertex $x \in G$, we denote by $E(x) = \{y \in G \mid E(x, y)\}$ the set of *successors of* x, and by $E^{\smile}(x) = \{y \in G \mid E(y, x)\}$ the set of *predecessors of* x with respect to the directed-edge relation of G. Letting E^* denote the transitive closure of E, we use $[x) = \{y \mid y \in E^*(x)\}$ to denote the set of vertices reachable from x and $(x] = \{y \mid x \in [y)\}$ to denote the set of vertices from which x is reachable. These notational conventions are extended to subsets of vertices, for example, for all $X \subseteq G$, we let $E^{\smile}(X) = \bigcup_{x\in X} E^{\smile}(x)$. For an $X \subseteq G$, we also write $\mathsf{G} \setminus X$ to denote the subgraph of G induced by the subset $G \setminus X$.

A *walk* p is a sequence of vertices $\langle x_0, x_1, x_2, ..., x_n\rangle$ such that $\forall 0 \leq i < n : x_{i+1} \in E(x_i)$ and such that all edges traversed are distinct, i.e. whenever $x_i = x_j$ for $0 \leq i \neq j < n$, we have $x_{i+1} \neq x_{j+1}$. The length of a walk is the number of edges it uses, $l(p) = n$. A walk is a *path* if it is also a sequence of distinct vertices. A *cycle* is a walk $\langle x_0, ..., x_{n-1}, x_n\rangle$ such that $\langle x_0, ..., x_{n-1}\rangle$ is a path and $x_n = x_0 \in E(x_{n-1})$.

A *sink* in G is a vertex $x \in G$ without successors and $sinks(\mathsf{G}) = \{x \in G \mid E(x) = \varnothing\}$ denotes the set of sinks of G. A vertex which is not a sink is *internal*, $int(\mathsf{G}) = G \setminus sinks(\mathsf{G})$. A *root* of G is a vertex $x \in G$ such that every other vertex is reachable by a path from x. The degree of $x \in G$, $d(x)$, is the number of edges incident to x in G.

[1] The notation $\mathcal{O}^*()$ suppresses polynomial factors from exponential functions

[2] Some results presented below apply to the infinite digraphs and infinitary propositional logic. However, in the present context of algorithm design, we assume all involved sets to be finite. Also, unless stated otherwise, by a graph we always mean a digraph.

A subset of vertices $S \subseteq G$ is *strongly connected* if (*): $\forall x, y \in S : x \in [y) \wedge y \in [x)$. Such an S is a strongly connected *component* if there is no set $S' \supset S$ such that $(*)$ holds. A strongly connected component S is *final* whenever $E(S) = S$. Since this will be of relevance for some algorithms, we remind the reader that it is possible, for instance by using Tarjan's algorithm [27], to decompose a graph into its strongly connected components in linear time.

For a digraph G, $\underline{\mathsf{G}}$ denotes the undirected graph obtained by turning every directed edge $\langle x, y\rangle$ into an undirected one $\{x, y\}$. An *oriented* graph is a digraph G obtained from $\underline{\mathsf{G}}$ by giving every undirected edge some direction. Such a graph does not contain any cycles of length 2.

A *kernel* of a digraph $\mathsf{G} = \langle G, E\rangle$ is a subset of vertices $K \subseteq G$ such that:

(i) $G \setminus K \supseteq E^{\smile}(K)$ (K is an independent set in G) and

(ii) $G \setminus K \subseteq E^{\smile}(K)$ (from every non-kernel vertex there is at least one edge to a kernel vertex).

Any kernel of G is an independent and dominating set in $\underline{\mathsf{G}}$. These two properties are equivalent to K being a maximal independent subset of $\underline{\mathsf{G}}$. Conversely, given a maximal independent subset K, we can determine if it is a kernel of G by verifying that every vertex $x \in G \setminus K$ has a directed edge into K (a $\underline{\mathsf{G}}$-edge in G might be only to x.)

Consequently, a possible (if not most efficient) algorithm for finding the kernels would unorient the input digraph G, find $\underline{\mathsf{G}}$'s maximal independent subsets, and for each such check if every node outside it has a directed edge to the subset. The number of maximal independent subsets of any G is limited by Moon and Moser's $3^{\frac{|G|}{3}}$ bound, [21]. It follows that there is an algorithm that finds all kernels in a graph in time $\mathcal{O}^*(3^{\frac{|G|}{3}})$. This running time is in fact tight for the problem of finding all kernels, as can be seen considering G that is a collection of disjoint symmetric cycles of length 3, i.e. the reversal of every edge is also present. For such a graph every maximal independent subset of $\underline{\mathsf{G}}$ is a kernel and there are $3^{\frac{|G|}{3}}$ of them. Even though for most digraphs only a proper subset of the maximal independent sets will be kernels, finding all kernels is not a computationally feasible problem. We consider only the problem of determining the existence of a kernel which, when one exists, amounts usually to producing it.

The problem is addressed using the equivalence between the existence of kernels and the satisfiability of propositional theories, arising from an equivalent definition of kernels. For a digraph $\mathsf{G} = \langle G, E\rangle$, an assignment $\alpha \in \{\mathbf{0}, \mathbf{1}\}^G$ (of truth-values to the vertices of G) is *correct* at a vertex $x \in G$ if $\alpha(x) = \mathbf{1} \Leftrightarrow \alpha(E(x)) \subseteq \{\mathbf{0}\}$ or equivalently, if:

$$\big(\alpha(x) = \mathbf{1} \ \wedge \ \alpha(E(x)) \subseteq \{\mathbf{0}\}\big) \quad \vee \quad \big(\alpha(x) = \mathbf{0} \ \wedge \ \mathbf{1} \in \alpha(E(x))\big) \tag{2.1}$$

An $\alpha \in \{\mathbf{0}, \mathbf{1}\}^G$ is a *solution* for G, $\alpha \in sol(\mathsf{G})$, if α is correct at every vertex of G, and if such an α exists G is *solvable*. For any $\alpha \subseteq G \times \{\mathbf{0}, \mathbf{1}\}$, we denote $\alpha^{\mathbf{1}} = \{x \in G \mid \langle x, \mathbf{1}\rangle \in \alpha\}$ and $\alpha^{\mathbf{0}} = \{x \in G \mid \langle x, \mathbf{0}\rangle \in \alpha\}$. For all graphs G and all assignments $\alpha \in \{\mathbf{0}, \mathbf{1}\}^G$ it holds that α is a solution iff $\alpha^{\mathbf{1}}$ is a kernel:

$$\alpha \in sol(\mathsf{G}) \Longleftrightarrow \alpha^{\mathbf{1}} = G \setminus E^{\smile}(\alpha^{\mathbf{1}}) \Longleftrightarrow \alpha^{\mathbf{1}} \text{ is a kernel of } \mathsf{G} \tag{2.2}$$

A possible algorithm for finding kernels is then based on the fact that every digraph G induces a propositional theory $\mathcal{T}(\mathsf{G})$ by taking, for each $x \in G$, the formula

$$x \leftrightarrow \bigwedge_{y \in E(x)} \neg y, \tag{2.3}$$

with the convention that $\mathbf{1} = \bigwedge_{y \in \varnothing} y$.[3] Then, letting $mod(\mathsf{T})$ denote all models of a theory T, the following equality holds:

$$sol(\mathsf{G}) = mod(\mathcal{T}(\mathsf{G})). \tag{2.4}$$

[3]Satisfiability of such a theory is equivalent to the existence of solutions for the corresponding system of boolean equations. This motivates the use of the name "solution", which was also used in the early days of kernel theory, e.g., in [28], p.588, or [25].

Since determining kernels is a special case of determining the models of propositional theories, we can feed the equations (2.3) together with $z = \mathbf{1}$ for all $z \in sinks(\mathsf{G})$ to a solver of systems of boolean equations, to determine if G has a kernel. Alternatively, we can feed the problem to a clausal SAT-solver. First, each equation (2.3) is equivalent to

$$\mathbf{1} \quad = \quad (x \vee \bigvee_{y \in E(x)} y) \quad \wedge \bigwedge_{y \in E(x)} (\neg y \vee \neg x). \tag{2.5}$$

Collecting now the right-hand-sides of these new equations and adding the requirement for all $z \in sinks(\mathsf{G})$, yields the formula in CNF:

$$CNF(\mathsf{G}) = \bigwedge_{x \in int(\mathsf{G})} \left((x \vee \bigvee_{y \in E(x)} y) \quad \wedge \bigwedge_{y \in E(x)} (\neg y \vee \neg x) \right) \wedge \bigwedge_{z \in sinks(\mathsf{G})} z. \tag{2.6}$$

Satisfiability of $CNF(\mathsf{G})$ is equivalent to the solvability of the system of equations (2.5) for all internal nodes, with all sinks assigned $\mathbf{1}$ which, in turn, is equivalent to the existence of a kernel in G, by (2.4).[4]

The above reduction and the resulting $CNF(\mathsf{G})$ are essentially the same as in [7]. The linear reduction in the opposite direction used there 3-Colorability, so we give a direct reduction from SAT: every propositional theory T can be transformed in linear time into a digraph $\mathcal{G}(\mathsf{T})$ such that $mod(\mathsf{T}) = sol(\mathcal{G}(\mathsf{T}))$. Many different graphs can satisfy these requirements, so we give only one example. First, assume a theory T given as a set of equivalences of the form

$$x \leftrightarrow \bigwedge_{i \in I_x} \neg y_i, \tag{2.7}$$

where all y, x_i are variables, and where every variable occurs at most once on the left of $\leftrightarrow$. The digraph $\mathcal{G}(\mathsf{T})$ is obtained by taking variables as vertices and, for every formula, introducing edges $x \to y_i$ for all $i \in I_x$. In addition, for every variable z *not* occurring on the left of any $\leftrightarrow$, we add a new vertex $\overline{z}$ and two edges $z \to \overline{z}$ and $\overline{z} \to z$. This last addition ensures that each variable z of T which would become a sink of $\mathcal{G}(\mathsf{T})$, and hence could only be assigned $\mathbf{1}$ by any solution of $\mathcal{G}(\mathsf{T})$, can be also assigned $\mathbf{0}$ (when the respective $\overline{z}$ is assigned $\mathbf{1}$). Letting $V(\mathsf{T})$ denote all variables of T, and $sol(\mathsf{X})|_Y$ restriction of assignments in $sol(\mathsf{X})$ to the variables in Y, we have that

$$mod(\mathsf{T}) = sol(\mathcal{G}(\mathsf{T}))|_{V(\mathsf{T})} \tag{2.8}$$

Now, an arbitrary theory T can be transformed into the above form. To simplify the transformation, assume T to be given as a set of clauses, each clause $C = \langle C^+, C^- \rangle$ consisting of the set of positive, $C^+ = \{x_p \mid p \in P\}$, and negative, $C^- = \{\neg x_n \mid n \in N\}$, literals. First, let a_C be a new variable. The formula $C' : a_C \leftrightarrow \neg a_C \wedge \neg C$ is equisatisfiable with C, with models related by the equation $mod(C') = mod(C) \times \{\langle a_C, \mathbf{0} \rangle\}$. Substituting for $\neg C$, we obtain a more explicit form of $C' : a_C \leftrightarrow \neg a_C \wedge \bigwedge_{p \in P} \neg x_p \wedge \bigwedge_{n \in N} x_n$. We introduce for every variable in the initial theory $x \in V(\mathsf{T})$, a new variable $\overline{x}$. For every such pair of variables we introduce the formulae (i), and for every clause C the formula (ii):

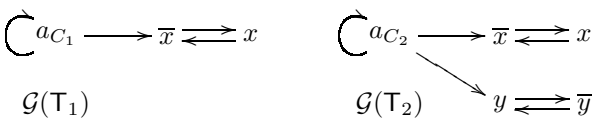
(i) $x \leftrightarrow \neg\overline{x}$ and $\overline{x} \leftrightarrow \neg x$.

(ii) $a_C \leftrightarrow \neg a_C \wedge \bigwedge_{p \in P} \neg x_p \wedge \bigwedge_{n \in N} \neg\overline{x}_n$

The theory C'' containing formulae (i) and (ii) is equisatisfiable with C and $mod(C) = mod(\mathsf{C}'')|_{V(C)}$. Defining $\mathsf{T}' = \bigcup_{C \in \mathsf{T}} \mathsf{C}''$ and letting $\mathcal{G}(\mathsf{T}) = \mathcal{G}(\mathsf{T}')$, the equality (2.8) remains valid.

[4] Assuming the adjacency list representation of the argument $\mathsf{G} = \langle G, E \rangle$, CNF is linear in the number of vertices, $|G|$, and edges, $|E|$ (each edge $\langle x, y \rangle$ giving rise to two pieces of data: $\neg y \vee \neg x$ and the element y in the disjunction for $x : x \vee ... \vee y \vee ...$).

Example 2.9 *For* $\mathsf{T}_1 = \{\neg x\}$*, respectively,* $\mathsf{T}_2 = \{\neg x \vee y\}$*, we obtain the digraphs:*



As a digression, we note that $\mathcal{G}(\mathsf{T})$ can be defined so that it is oriented and has no loops. In addition to a_C, add two more nodes in a 3-cycle $\langle a_C, b_C, c_C, a_C\rangle$, and for every $x \in V(\mathsf{T})$, introduce in (i) two more new nodes, replacing the 2-cycle by the 4-cycle: $\langle x, \overline{x}, x', x'', x\rangle$.

Both equations (2.4) and (2.8) hold for arbitrary digraphs but when they have infinite branchings, the corresponding theory is in infinitary propositional logic. In this paper, we are concerned exclusively with usual propositional logic and finite graphs, so "graph" and "arbitrary graph" mean here only a finite digraph.

3 Preprocessing

This section presents some simplifications reducing the input graph, which will be later combined with different algorithms. In Subsection 3.1, we show that we can consider only the problem for graphs without sinks, since kernels of an arbitrary graph G are determined by the kernels of its appropriate, sinkless subgraph which can be obtained from G in linear time. Subsection 3.2 presents some further simplifications of a graph which are based on local dependencies and are of linear, or low polynomial, complexity.

3.1 Forcing values

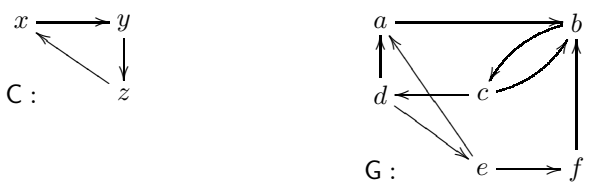
The obvious brute-force approach, simply checking the condition (2.1) for every possible assignment, can be improved by observing consequences of a given partial assignment. The following definition captures some such consequences that are recognizable locally in the graph.

Definition 3.1 *A* partial assignment *to a graph* G *is an* $\alpha \in \{\mathbf{0},\mathbf{1}\}^X$ *for any* $X \subseteq G$*. Given such an* α*, we define inductively its extension to the nodes which obtain forced values:*

$$\begin{aligned}
\alpha^{\mathbf{1}}_1 &= \alpha^{\mathbf{1}}\\
\alpha^{\mathbf{0}}_1 &= \alpha^{\mathbf{0}}\\
\alpha^{\mathbf{0}}_{i>1} &= E(\alpha^{\mathbf{1}}_{i-1}) \cup E^{\smile}(\alpha^{\mathbf{1}}_{i-1}) \cup \alpha^{\mathbf{0}}_{i-1}\\
\alpha^{\mathbf{1}}_{i>1} &= \mathit{sinks}(\mathsf{G} \setminus \alpha^{\mathbf{0}}_i) \cup \alpha^{\mathbf{1}}_{i-1} \cup \{x \in E(y) \mid y \in \alpha^{\mathbf{0}}_i \wedge \{x\} = E(y) \setminus \alpha^{\mathbf{0}}_i\}
\end{aligned}$$

Fixed-point is reached when $\alpha^{\mathbf{1}}_k = \alpha^{\mathbf{1}}_{k-1}$*, no later than for* $k = |G|$*. We then let* $\overline{\alpha}^{\mathbf{1}} = \bigcup \alpha^{\mathbf{1}}_i$*,* $\overline{\alpha}^{\mathbf{0}} = \bigcup \alpha^{\mathbf{0}}_i$ *and set* $\overline{\alpha} = \{(n,\mathbf{1}) \mid n \in \overline{\alpha}^{\mathbf{1}}\} \cup \{(n,\mathbf{0}) \mid n \in \overline{\alpha}^{\mathbf{0}}\}$.

Example 3.2 *Consider the following two graphs:*



In C*,* $\alpha = \{\langle x,\mathbf{1}\rangle\}$ *gives* $\alpha^{\mathbf{0}}_2 = \{z,y\}$*,* $\alpha^{\mathbf{1}}_2 = \{x,z\}$ *and then* $\alpha^{\mathbf{0}}_3 = \{x,y,z\}$*,* $\alpha^{\mathbf{1}}_3 = \{x,y,z\}$*, i.e.,* $\overline{\alpha} = \{x,y,z\} \times \{\mathbf{0},\mathbf{1}\}$.

In G*, from* $\alpha = \{\langle c,\mathbf{0}\rangle\}$ *we obtain* $\overline{\alpha} = \{\langle c,\mathbf{0}\rangle, \langle b,\mathbf{1}\rangle, \langle a,\mathbf{0}\rangle, \langle f,\mathbf{0}\rangle, \langle e,\mathbf{1}\rangle, \langle d,\mathbf{0}\rangle\}$*, while* $\beta = \{\langle c,\mathbf{1}\rangle\}$ *leads to* $\overline{\beta} = \{\langle c,\mathbf{1}\rangle, \langle d,\mathbf{0}\rangle, \langle b,\mathbf{0}\rangle, \langle a,\mathbf{1}\rangle, \langle f,\mathbf{1}\rangle, \langle e,\mathbf{0}\rangle\}$*. In both cases, the resulting assignment is a a solution of* G*. Starting with* $\gamma = \{\langle e,\mathbf{0}\rangle\}$ *does not induce any values, i.e.,* $\overline{\gamma} = \gamma$.

C shows that $\overline{\alpha}$ may happen to be a (non-functional) relation, i.e., $\overline{\alpha}^{\mathbf{1}} \cap \overline{\alpha}^{\mathbf{0}} \neq \varnothing$. If this is the case, then we cannot find a correct assignment that extends α. However, if $\overline{\alpha}$ is a function, then for all $x \in dom(\overline{\alpha})$ we have the following weaker form of correctness:

$$\big(\overline{\alpha}(x) = \mathbf{1} \ \wedge \ \overline{\alpha}(E(x)) \subseteq \{\mathbf{0}\}\big) \quad \vee \quad \big(\overline{\alpha}(x) = \mathbf{0} \ \wedge \ \exists y \in E(x) : y \notin \overline{\alpha}^{\mathbf{0}}\big) \tag{3.3}$$

We say that $\overline{\alpha}$ is *consistent* in this case. Given a consistent partial assignment $\overline{\alpha}$, it might, but need not, be possible to extend it to a solution for G. This depends on the solvability of the subgraph yet to be assigned, but also on the possibility of finding a solution for the remaining graph such that each vertex assigned $\mathbf{0}$ by $\overline{\alpha}$ is eventually justified by the assignment of $\mathbf{1}$ to one of its successors. In particular, we have to meet this constraint on the *border* of α, defined as follows:

Definition 3.4 *Given a partial assignment α to a graph G, the* border *of α is the set $bord(\alpha) = \{x \in dom(\alpha) \mid \alpha(x) = \mathbf{0} \wedge \mathbf{1} \notin \alpha(E(x))\}$.*

The formula (3.3) implies that a consistent partial assignment is correct everywhere with the possible exception of its border.

Remark 3.5 *When a partial assignment α is correct on its whole domain, i.e., $\alpha \in sol(dom(\alpha))$, then $\alpha^{\mathbf{1}} \subseteq G$ is called a* local kernel *(sometimes* semi-kernel*) in kernel theory. Local kernels are used in inductive proofs of sufficient conditions for the existence of kernels in digraphs from certain classes, e.g. in [2, 14, 13, 16]. Deciding if a graph has a local kernel is NP-complete, [12].*

Any $\beta \in sol(\mathsf{G})$ must be such that its restriction to any subset $B \subseteq G$ is consistent on the subgraph induced by B. Also, every solution respects all values induced by its own restrictions, in particular, induced from the empty assignment. Consequently, the values induced from the empty assignment are the same in all solutions (if any). These observations are gathered in the following lemma. $\mathsf{G}^{\circ}_{\alpha}$ denotes the subgraph $\mathsf{G} \setminus dom(\overline{\alpha})$, $\varnothing$ denotes the empty assignment, and we abbreviate $\mathsf{G}^{\circ} = \mathsf{G}^{\circ}_{\varnothing}$.

Lemma 3.6 *For an arbitrary G:*

1. *$bord(\overline{\varnothing}) = \varnothing$;*
2. *for any partial assignment $\alpha : sinks(\mathsf{G}^{\circ}_{\alpha}) = \varnothing$;*
3. *$\forall \beta \in sol(\mathsf{G}) \ \forall B \subseteq G : \overline{\beta|_B} = \beta|_{dom(\overline{\beta|_B})}$;*
4. *$sol(\mathsf{G}) = \{\beta \cup \overline{\varnothing} \mid \beta \in sol(\mathsf{G}^{\circ})\}$.*

Proof. 1. It follows by induction that each $\varnothing_i$ satisfies (2.1), i.e., $E(\varnothing_i^{\mathbf{1}}) \subseteq \varnothing_i^{\mathbf{0}} \wedge \forall x \in \varnothing_i^{\mathbf{0}} : E(x) \cap \varnothing_i^{\mathbf{1}} \neq \varnothing$. This holds trivially at the start with $\varnothing_1 = \varnothing$, and after first iteration when $\varnothing_2^{\mathbf{1}} = sinks(\mathsf{G})$ and $\varnothing_2^{\mathbf{0}} = \varnothing$. Assuming (2.1) as IH for $\varnothing_i$, then
for each new $x \in \varnothing_{i+1}^{\mathbf{0}} : x \in E^{\smile}(\varnothing_i^{\mathbf{1}})$, because $E(\varnothing_i^{\mathbf{1}}) \subseteq \varnothing_i^{\mathbf{0}}$ by IH
for each new $x \in \varnothing_{i+1}^{\mathbf{1}} : x \in sinks(\mathsf{G} \setminus \varnothing_{i+1}^{\mathbf{0}})$ – the last component does not apply, since for any $y \in \varnothing_i^{\mathbf{0}}$ there is a $z \in E(y) \cap \varnothing_i^{\mathbf{1}}$ by IH.
2. $\overline{\alpha} = \alpha_i = \alpha_{i+1}$ for some $i \geq 0$ and assume $x \in sinks(\mathsf{G}^{\circ}_{\alpha})$, i.e., $E(x) \subseteq dom(\overline{\alpha})$. If $E(x) \cap \overline{\alpha}^{\mathbf{1}} \neq \varnothing$, then $x \in \alpha_{i+1}^{\mathbf{0}}$ and otherwise $x \in \alpha_{i+1}^{\mathbf{1}}$. In either case $x \in dom(\alpha_{i+1}) = dom(\overline{\alpha})$. Contradiction.
3. By induction on the steps used in the construction of $\overline{\beta|_B}$, Definition 3.1, we show that for all $i : (\beta|_B)_i = \beta|_{dom((\beta|_B)_i)}$. The basis is trivial since $(\beta|_B)_1 = (\beta|_B) = \beta|_{dom((\beta|_B))}$. For the induction step, any $x \in (\beta|_B)_{i+1}$ gives one of the following cases:
(0) $x \in (\beta|_B)_{i+1}^{\mathbf{0}}$ i.e., either

- $x \in (\beta|_B)_i^{\mathbf{0}}$ which, by IH, means that $x \in \beta^{\mathbf{0}}$ or
- $x \in E((\beta|_B)_i^{\mathbf{1}}) \cup E^{\smile}((\beta|_B)_i^{\mathbf{1}})$ which, by IH and correctness of β, means that $x \in E(\beta^{\mathbf{1}}) \cup E^{\smile}(\beta^{\mathbf{1}}) \subseteq \beta^{\mathbf{0}}$, or

(1) $x \in (\beta|_B)_{i+1}^{\mathbf{1}}$ i.e., either

- $x \in (\beta|_B)^{\mathbf{1}}_i$ which, by IH, means that $x \in \beta^{\mathbf{1}}$ or
- $x \in \mathit{sinks}(\mathsf{G} \setminus (\beta|_B)^{\mathbf{0}}_{i+1})$, i.e., $E(x) \subseteq (\beta|_B)^{\mathbf{0}}_{i+1} \subseteq \beta^{\mathbf{0}}$ by point (0), and $x \in \beta^{\mathbf{1}}$ by correctness of β, or
- $\{x\} = E(y) \setminus (\beta|_B)^{\mathbf{0}}_{i+1} : y \in (\beta|_B)^{\mathbf{0}}_{i+1}$, i.e. by point (0) we have $y \in \beta^{\mathbf{0}}$ and $E(y) \setminus \{x\} \subseteq \beta^{\mathbf{0}}$. Then by correctness of β we must have $\{x\} = E(y) \setminus \beta^{\mathbf{0}}$ with $x \in \beta^{\mathbf{1}}$.

4. For every $x \in G^\circ : E(x) \cap \overline{\varnothing}^{\mathbf{1}} = \varnothing$ and, by 2, $E(x) \cap G^\circ \neq \varnothing$. Hence, every $\beta \in \mathit{sol}(\mathsf{G}^\circ)$ can be combined with $\overline{\varnothing}$ into a correct solution for G. But the values on $\mathit{dom}(\overline{\varnothing})$ can not be chosen otherwise since, by 3, $\forall \alpha : \alpha \in \mathit{sol}(\mathsf{G}) \to \alpha|_{\mathit{dom}(\overline{\varnothing})} = \overline{\varnothing}$. □

The construction from Definition 3.1, together with Lemma 3.6, will provide the basic simplification mechanism used in all our algorithms. According to point 4, we can first (in linear time) induce all values from the sinks of G, removing $\mathit{dom}(\overline{\varnothing})$ from the graph. Then, trying various partial assignments σ to the remaining, sinkless subgraph G°, point 3 ensures that it suffices to consider only the induced assignment $\overline{\sigma}$, thus reducing the search space.

In the following subsection, we identify some particular, structural patterns allowing local simplifications of the graph.

3.2 Simplification

The number of possible simplifications, preserving and reflecting solvability, can be unlimited. In practice, one has to choose some which can be expected to occur frequently and can be performed cheaply. Two such simplifications are given, providing also some information about the structural properties of kernels. The first one concerns a special type of paths.

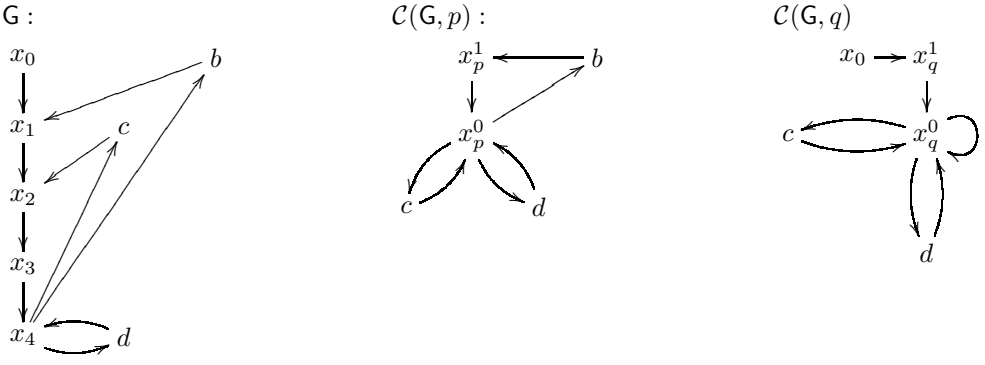
Definition 3.7 *A path $p = \langle x_0, x_1, ..., x_{l(p)} \rangle$ is* isolated *if $\forall 0 \leq i < l(p) : E(x_i) = \{x_{i+1}\}$.*

It follows from Definition 3.1 that any assignment, of $\mathbf{0}$ or $\mathbf{1}$, to any vertex on an isolated p will induce values to every other vertex on the path. So the vertices on isolated paths do not contribute anything to the structural properties of G determining its kernels. They can be removed.

Definition 3.8 *For an isolated path $p = \langle x_{0,p}, ..., x_{l(p),p} \rangle$ with $l(p) \geq 2$, let $P \subseteq G$ denote all nodes $x_{i,p}$ on p. The graph $\mathcal{C}(\mathsf{G}, p)$, the* contraction *of G on p, is defined by a mapping $f : G \to \mathcal{C}(G, p)$:*

- $\mathcal{C}(G, p) = G \setminus \{x_{i,p} \mid x_{i,p} \in P\} \cup \{x^0_p, x^1_p\}$
- $f : G \to \mathcal{C}(G, p)$ *is defined by $f(x) = x$ when $x \in \mathcal{C}(G, p)$, $f(x_{i,p}) = x^0_p$ when $i + l(p)$ is even and $f(x_{i,p}) = x^1_p$ otherwise*
- $\mathcal{C}(E, p) = \{\langle x, y \rangle \mid \exists \langle x', y' \rangle \in (f^-(x) \times f^-(y)) \cap E : x' = x \vee x' = x_{l(p),p} \vee y' = x_{l(p),p}\}$

Example 3.9 *We contract the isolated path $p = \langle x_0, x_1, x_2, x_3, x_4 \rangle$ in the digraph G, obtaining the digraph $\mathcal{C}(\mathsf{G}, p)$ where f is defined on p by $f(x_0) = f(x_2) = f(x_4) = x^0_p, f(x_1) = f(x_3) = x^1_p$. Also shown is the digraph $\mathcal{C}(\mathsf{G}, q)$ obtained by contracting $q = \langle b, x_1, x_2, x_3, x_4 \rangle$ with $f(b) = f(x_2) = f(x_4) = x^0_q, f(x_1) = f(x_3) = x^1_q$.*



Contraction of isolated paths preserves and reflects solutions as stated in the following Fact. ($f;g$ denotes function composition in diagrammatic order: f followed by g.)

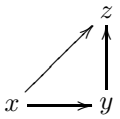
Fact 3.10 *For any isolated path p with $l(p) \geq 2$ in any G :*
$sol(\mathsf{G}) = \{\alpha \mid \exists \beta \in sol(\mathcal{C}(\mathsf{G}, p)) : \alpha = f;\beta\}$.

PROOF. $\supseteq$) For a $\beta \in sol(\mathcal{C}(\mathsf{G}, p))$ define $\alpha \in \{\mathbf{0}, \mathbf{1}\}^G$ by $\forall x \in G : \alpha(x) = \beta(f(x))$. To show $\alpha \in sol(\mathsf{G})$ it suffices, by definition of f, to show that α is correct on p. For $x_{i,p}$ such that $i+l(p)$ is odd or $i = l(p)$, correctness follows since by definition of f and the fact that p is isolated we have $f(E(x_{i,p})) = E^{\mathcal{C}(\mathsf{G},p)}(f(x_{i,p}))$. All other $x_{i,p}$'s are such that $i + l(p)$ is even and $i > 0$, and since p is isolated we have $f(E(E(x_{i,p}))) = f(x_{i,p}) = f(x_{l(p),p})$. So correctness follows from correctness of $\alpha(x_{l(p),p})$
$\subseteq$) Assume $\alpha \in sol(\mathsf{G})$. By definition of f and the fact that p is an isolated path it follows that $\forall x : \forall y_1, y_2 \in f^-(x) : \alpha(y_1) = \alpha(y_2)$. Then we define β for every $x \in \mathcal{C}(G, p)$ by choosing arbitrary $y \in f^-(x)$ and taking $\beta(x) = \alpha(y)$. Then β is correct and it satisfies $\forall x \in G : \alpha(x) = \beta(f(x))$ □

As the second simplification we remove basic contradictions.

Definition 3.11 *An $x \in G$ is a* basic contradiction *if $\exists y \in E(x) : E(y) \subseteq E(x)$.*

Important special cases include the predecessors of sinks, loops, and triangles such as the following graph:

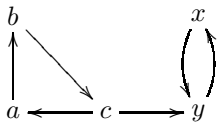


The following fact is obvious:

Fact 3.12 *If x is a basic contradiction in G then $\forall \alpha \in sol(\mathsf{G}) : \alpha(x) = \mathbf{0}$.*

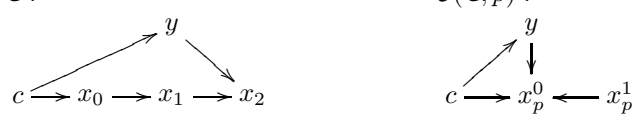
PROOF. Let $y \in E(x)$ be such that $E(y) \subseteq E(x)$ and $\alpha \in sol(\mathsf{G})$. If $\alpha(y) = \mathbf{1}$ then $\alpha(x) = \mathbf{0}$, while if $\alpha(y) = \mathbf{0}$ then, for some $z \in E(y) : \alpha(z) = \mathbf{1}$. But then also $\alpha(x) = \mathbf{0}$ since $z \in E(y) \subseteq E(x)$. □

The notion of basic contradiction is motivated by the fact that a (general) contradiction, i.e. an x such that $\forall \alpha \in sol(\mathsf{G}) : \alpha(x) = \mathbf{0}$, may not be identifiable as such locally by inspecting its fixed neighbourhood. For instance, in the following graph, x is a contradiction since $x = \mathbf{0}$ is necessary (and sufficient) for the existence of a correct assignment to the rest of the graph.



The contraction of isolated paths can turn a contradiction into a basic one, as the following example illustrates.

Example 3.13 *The graph $\mathcal{C}(\mathsf{G}, p)$ results from contracting the isolated path $p = \langle x_0, x_1, x_2\rangle$ in G. After contraction, c becomes a basic contradiction, revealing that it is a contradiction in G:*



The specific case of Fact 3.10, covered by the following fact, characterizes the contradictions which become basic after contraction of isolated paths. (Basic contradiction is a special case when $l(p) = 0$ and $l(q) = 1$.)

Fact 3.14 *Given isolated paths* $p = \langle x_{0,p}, x_{1,p}, ..., x_{l(p),p}\rangle$, $q = \langle x_{0,q}, x_{1,q}, ..., x_{l(q),q} = x_{l(p),p}\rangle$ *such that* $l(p) + l(q)$ *is odd. If* $c \in G$ *is such that* $x_{0,p}, x_{0,q} \in E(c)$, *then* $\forall \alpha \in sol(\mathsf{G}) : \alpha(c) = \mathbf{0}$.

Proof. Assume arbitrarily that p has odd length and that we start by contracting p to obtain $\mathsf{H} = \mathcal{C}(\mathsf{G}, \mathsf{p})$. Then we have $x_p^1 \in E^{\mathsf{H}}(c)$, and there is an isolated path $q = \langle x_{0,q}, x_{1,q}, ..., x_{l(q),q} = x_p^0\rangle$ in H. Contracting q to obtain $\mathsf{K} = \mathcal{C}(\mathsf{H}, \mathsf{q})$ we obtain a graph where $x_q^0 = x_p^0 \in E^{\mathsf{K}}(c)$ and $E^{\mathsf{K}}(x_p^1) = \{x_q^0\}$. So by facts 3.10 and 3.12 it follows that $\forall \alpha \in sol(\mathsf{G}) : \alpha(c) = \mathbf{0}$. □

Similar facts can be proven for other situations, where contracting some collection of paths reveals a basic contradiction (for instance in the case of isolated cycles of odd length, or with two paths p, q as in Fact 3.14 but admitting also outgoing edges at nodes with even indices x_{2i}.) We do not attempt to give a complete classification, however.

Towards an algorithm for KER, we gather the two rules for isolated paths and basic contradictions into the simplification procedure $simp(\mathsf{G})$ as shown in Algorithm 3.15. The algorithm returns the error value $\perp$ if it discovers the non-existence of solutions. Otherwise, by Facts 3.10, 3.12 and Lemma 3.6, every solution to the input graph G can be obtained from a solution to the returned graph. [5]

Algorithm 3.15 $simp(\mathsf{G})$

```
if there is an isolated path p with l(p) ≥ 2 then
  return simp(C(G, p))
else if there is a basic contradiction x ∈ G then
  α := {⟨x, 0⟩}
  if ᾱ is a function then
    return simp(G \ dom(ᾱ))
  else
    return ⊥
  end if
else
  return G
end if
```

4 Breaking cycles

According to Richardson's theorem [25], every finitely branching (in particular, finite) graph not containing odd cycles has a kernel. Consequently, a possible approach to KER is to try *breaking* the odd cycles. Below, we reduce the number of cycles to consider and give a general treatment of this approach utilizing the following concept.

Definition 4.1 *For a graph* G, *we define* $\mathcal{B}(\mathsf{G}) = \{X \subseteq G \mid \forall \beta \in sol(\mathsf{G}) : \exists \alpha \in \{\mathbf{0}, \mathbf{1}\}^X : \overline{\alpha} = \beta\}$. *An* $X \in \mathcal{B}(\mathsf{G})$ *is called a* basis *for* $sol(\mathsf{G})$.

Thus, for any $X \in \mathcal{B}(\mathsf{G})$, any solution for G can be obtained by inducing from some assignment to X, reducing the complexity of the brute-force approach to $2^{|X|}$. It remains to be proven that suitable $X \in \mathcal{B}(\mathsf{G})$ exists. Below we provide two types of bases, guaranteed to exist for any graph. In algorithmic terms this means that KER, when parameterized by the size of either of these bases, is FPT. It should be noted here that a more obvious choice of parameter for KER, the size of the kernel we are looking for, does not make the problem FPT for general graphs unless

[5] Inducing and checking the existence of isolated paths can be done in linear time. The trivial search for basic contradictions would visit, for every node x, each of its successors $y \in E(x)$, checking if $E(y) \subseteq E(x)$. The worst case $|G|^2$ hardly ever obtains and, in practice, even this trivial procedure is sub-quadratic.

collapses, deemed unlikely, occur among the parameterized complexity classes.[6] So the result in Subsection 4.2, admitting as a basis any set of vertices touching all even cycles, appears to be the best currently available regarding the parameterized complexity of KER.

4.1 Feedback Vertex Sets

A *feedback vertex set* for a graph G is a subset $F \subseteq G$ such that $\mathsf{G} \setminus F$ is acyclic (a dag).

Proposition 4.2 *For any graph* G*, if* F *is a feedback vertex set for* G *then* $F \in \mathcal{B}(\mathsf{G})$*.*

Proof. Let F be a feedback vertex set for G and consider arbitrary $\beta \in sol(\mathsf{G})$. Then by lemma 3.6 we have $\overline{\beta|_F} = \beta|_{dom(\overline{\beta|_F})}$. All we need to prove is $dom(\overline{\beta|_F}) = G$. So consider $\mathsf{G} \setminus dom(\overline{\beta|_F})$. By Lemma 3.6.2, this graph has no sinks, and as F is a feedback vertex set, it has no cycles. Since G is finite, it follows that $\mathsf{G} \setminus dom(\overline{\beta|_F}) = \varnothing$, as desired. □

This observation gives a simple algorithm for KER: find some feedback vertex set F and try all possible assignments to its nodes, verifying if the induced assignments are correct on the whole graph. More cleverly, proposition 4.2 can be used to construct a branch and bound algorithm that only branches at vertices from F. An algorithm based on this idea is presented in [11]. We will return to branch and bound algorithms in Section 5, but note here that as the success of such an approach depends on finding *small* feedback vertex sets, we can not expect it to be optimal for all graphs. It will be good enough, though, for solving KER effectively on graphs that admit small feedback vertex sets. This follows from the recent work in [5], showing that the problem of finding a minimum feedback vertex set is FPT in the size of such a set. In particular, KER is FPT in the size of a *minimum* feedback vertex set.

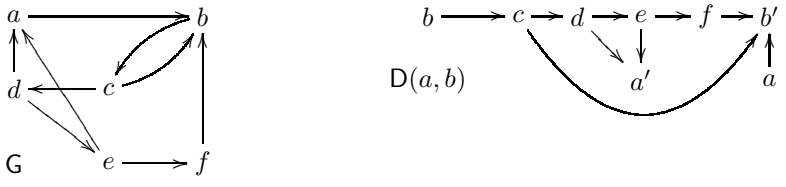
Feedback vertex sets are useful tools when graphs are viewed algebraically as systems of boolean equations. In this context they allow for a systematic substitution of equals for equals that both preserves and reflects solutions, allowing us to represent G more compactly than the system $\mathcal{T}(\mathsf{G})$ from (2.3). In the rest of this subsection we present this construction, linking substitution in systems of boolean equations with feedback vertex sets of graphs. We do this by introducing labeled dag's that are nice in their own right in that they provide a visualization of the bases originating from feedback vertex sets. [7]

F denotes such a set and given it, we represent G as a (labeled) dag $\mathsf{D}(F) = \langle D_F, E_F \rangle$, where $F' = \{x' \mid x \in F\}$ is a set of new elements and:

$$\begin{array}{rcl} D_F & = & G \cup F' \\ E_F & = & \Big(E^{\mathsf{G}} \setminus \{\langle y, x\rangle \mid x \in F\}\Big) \cup \{\langle y, x'\rangle \mid x' \in F' \wedge x \in E(y)\} \end{array} \tag{4.3}$$

The new vertices are exactly the new sinks $F' = sinks(\mathsf{D}(F)) \setminus sinks(\mathsf{G})$ and $|F'| \leq$ number of cycles in G. The labeling, defined by $l(x) = x$ for $x \in G$ and $l(x') = x$ for the new $x' \in F'$, serves establishing a unique correspondence between solutions of $\mathsf{D}(F)$ and of G.

Example 4.4 $\mathsf{D}(a, b) \in \mathsf{dag}(\mathsf{G})$ *is obtained from the feedback vertex set* $\{a, b\}$*.*



[6] In [17] it is shown that using the size of the kernel as parameter does make the problem FPT for planar digraphs. ([23] provides an introduction to parameterized complexity.)

[7] The question whether this correspondence could be applied for solving more general systems of boolean equations seems an interesting research challenge in its own right

The double 'a' would disappear if we constructed the dag $\mathsf{D}(b)$ *from the feedback set* $\{b\}$*. It would have an edge from* a' *(which became* a*) to* b' *and no extra* a *without incoming edges.*

Let $\mathsf{dag}(\mathsf{G})$ denote the set of so obtained dags from a given G. Given a $\mathsf{D}(F) \in \mathsf{dag}(\mathsf{G})$, we can use inductive definitions over this representation. In particular, any assignment to the new sinks, $\beta \in \{\mathbf{0},\mathbf{1}\}^{F'}$, induces an assignment $\overline{\beta}$ to the whole D, in linear time. We only verify that the values assigned to the new sinks $x' \in F'$ are the same as the values induced at the respective $x \in F$. In the above $\mathsf{D}(a,b)$, trying $a' = \mathbf{1} = b'$ fails inducing $a = \mathbf{0}$. Trying $b' = \mathbf{1}$ and $a' = \mathbf{0}$ induces the same values at b and a, allowing to conclude the existence of a kernel for G.

$sol(\mathsf{G})$ becomes thus captured by a new system of equations requiring the values assigned to F' to agree with the values induced in F. The system is defined as follows. For every vertex $x \in G \setminus sinks(\mathsf{G}) = int(\mathsf{D}(F))$, divide the set of its successors $E_F(x)$ into two disjoint subsets: $E_L(x) = E_F(x) \cap F'$ and $E_R(x) = E_F(x) \setminus E_L(x)$.

Definition 4.5 *For* $x \in sinks(\mathsf{G})$ *let* $FRM_{\mathsf{D}(F)}(x) = \mathbf{1}$ *and for* $x \in int(\mathsf{D}(F))$ *define:*

$$FRM_{\mathsf{D}(F)}(x) = \bigwedge_{y \in l(E_L(x))} \neg y \wedge \bigwedge_{z \in E_R(x)} \neg FRM_{\mathsf{D}(F)}(z).$$

The reduced system is $EQU_{\mathsf{D}(F)}(\mathsf{G}) = \{FRM_{\mathsf{D}(F)}(x) = x \mid x \in F\}$.

Example 4.6 (4.4 continued) *The reduced system* $EQU_{\mathsf{D}(a,b)}(\mathsf{G})$ *has two equations:* $a = \neg b$ *and* $b = \neg(\neg b \wedge \neg(\neg a \wedge \neg(\neg a \wedge \neg\neg b)))$.

The dag $\mathsf{D}(b) \in \mathsf{dag}(\mathsf{G})$ *would give the corresponding reduced system with only one equation (equivalent to the one obtained by substituting* $a = \neg b$ *in the above system), namely:* $b = \neg(\neg b \wedge \neg(\neg\neg b \wedge \neg(\neg\neg b \wedge \neg\neg b)))$*. Simplifying its right-hand side, we gradually obtain the trivial equation* $b = \neg(\neg b \wedge \neg(\neg\neg b \wedge \neg\neg\neg b))) = \neg(\neg b \wedge \neg\mathbf{0})) = b$.

Each $FRM_{\mathsf{D}(F)}(x)$ contains only variables from F, so an $\alpha \in \{\mathbf{0},\mathbf{1}\}^F$ can be extended to $\alpha^* \in \{\mathbf{0},\mathbf{1}\}^G$ as follows ($\alpha[\phi]$ denotes the usual evaluation of the formula ϕ under the assignment α):

$$\alpha^*(x) = \begin{cases} \alpha(x) & \text{if } x \in F \\ \alpha[FRM_{\mathsf{D}(F)}(x)] & \text{otherwise} \end{cases} \tag{4.7}$$

This makes α^* a function consistent with $\overline{\alpha}$ induced according to Definition 3.1, i.e., $\alpha^* \subseteq \overline{\alpha}$. Every solution for G is, in fact, such an α^* obtained from a solution for $EQU_{\mathsf{D}(F)}(\mathsf{G})$.

Proposition 4.8 *For any* $\mathsf{D}(F) \in \mathsf{dag}(\mathsf{G})$*:*

$sol(\mathsf{G}) = \{\alpha^* \mid \alpha \in \{\mathbf{0},\mathbf{1}\}^F \wedge \forall x \in F : \alpha(x) = \alpha[FRM_{\mathsf{D}(F)}(x)]\}$.

Proof. $\supseteq$) If the equality holds for F, then (4.7) makes it hold also for all other nodes. Then, for every $x \in G$, we have that (*) $\alpha^*(x) = \mathbf{1} \Leftrightarrow \alpha[FRM_{\mathsf{D}(F)}(x)] = \mathbf{1}$, and hence

$$\begin{aligned}
\alpha^*(x) = \mathbf{1} &\Leftrightarrow \Big(\bigwedge_{y \in l(E_L(x))} \neg\alpha^*(y) \wedge \bigwedge_{z \in E_R(x)} \neg\alpha[FRM_{\mathsf{D}(F)}(z)]\Big) = \mathbf{1} && (*) \\
&\Leftrightarrow \Big(\bigwedge_{y \in l(E_L)(x)} \neg\alpha^*(y) \wedge \bigwedge_{z \in E_R(x)} \neg\alpha^*(z)\Big) = \mathbf{1} && (*) \\
&\Leftrightarrow \bigwedge_{y \in E(x)} \neg\alpha^*(y) = \mathbf{1} && E(x) = l(E_L(x)) \cup E_R(x) \\
&\Leftrightarrow \forall y : y \in E(x) \to \alpha^*(y) = \mathbf{0}
\end{aligned}$$

$\subseteq$) For an arbitrary $\beta \in sol(\mathsf{G})$, let $\alpha = \beta|_F$. Since $F \in \mathcal{B}(\mathsf{G})$, so $\overline{\alpha} = \beta$. But since $\overline{\alpha}$ and α^* both are functions and $\alpha^* \subseteq \overline{\alpha}$, so $\alpha^* = \overline{\alpha} = \beta$. □

In Example 4.6, the reduced system simplified to one trivial equation $b = b$, so the graph G has exactly two solutions, each induced from a solution to this equation.

Expressing this proposition in terms of the assignment $\overline{\alpha}$, induced in the dag $\mathsf{D}(F) \in \mathsf{dag}(\mathsf{G})$ from the assignment $\alpha \in \{\mathbf{0},\mathbf{1}\}^{F'}$ to its new sinks F', gives the following claim:

$$sol(\mathsf{G}) = \{\overline{\alpha}|_G \mid \alpha \in \{\mathbf{0},\mathbf{1}\}^{F'} \wedge \forall x' \in F' : \alpha(x') = \overline{\alpha}(x)\}.$$

The above algorithms, whether utilizing the reduced system of equations $EQU_{\mathsf{D}(F)}(\mathsf{G})$ or merely inducing values directly in $\mathsf{D}(F)$, rely on finding an arbitrary feedback vertex set. The following subsection presents an algorithm for which it suffices to find a subset of nodes breaking only the even cycles.

4.2 Breaking Even Cycles

Dually to Richardson's theorem, we have the following fact.

Lemma 4.9 *If $\mathsf{G} \neq \varnothing$, $sinks(\mathsf{G}) = \varnothing$, and G has no even cycles, then $sol(\mathsf{G}) = \varnothing$.*

Proof. Assume towards contradiction that $\alpha \in sol(\mathsf{G})$. Clearly, $\alpha^{\mathbf{1}} \neq \varnothing$. So choose $a \in \alpha^{\mathbf{1}}$ and consider a sequence of sets $V : \mathsf{N} \to \mathcal{P}([a])$ such that

$V_0 = \{a\}$
$V_{2i+1} = \bigcup_{x \in V_{2i}} E(x)$
$V_{2i+2} = \bigcup_{x \in V_{2i+1}} \{y_x\}$, where $y_x \in E(x)$ is such that $\alpha(y_x) = \mathbf{1}$ (if it exits).

By correctness of α, such a sequence satisfies $\bigcup_i V_{2i} \subseteq \alpha^{\mathbf{1}}$ and $\bigcup_i V_{2i+1} \subseteq \alpha^{\mathbf{0}}$ and, as $sinks(\mathsf{G}) = \varnothing$ so $\forall i \in \mathsf{N} : V_i \neq \varnothing$. Also, it is easy to see that for every $n \in \mathsf{N}$ and every $a_n \in V_n$ there is a sequence of edges $\langle a, a_1, a_2, ..., a_n \rangle$ such that $\forall i : a_i \in V_i \cap E(a_{i-1})$. So there is an infinite sequence of edges $p = \langle a, a_1, a_2, ... \rangle$ such that $\forall i : a_i \in V_i \cap E(a_{i-1})$. Since G is finite this is only possible if $\exists j > i : a_i = a_j$. Let i, j be a pair satisfying this condition and such that for all $i \leq k < l < j : a_k \neq a_l$. The sequence of edges $C = \langle a_i, a_{i+1}, ..., a_j \rangle$ must be of even length since otherwise $a_i \in \alpha^{\mathbf{1}} \cap \alpha^{\mathbf{0}}$, i.e., an even cycle, contradicting our assumption about G. □

From an algorithmic point of view, the observation that odd cycles are the only obstacle to the existence of kernels suggests (not very efficient) algorithms based on breaking the odd cycles. The above observation suggests that we can restrict attention to even cycles, and the following proposition makes this suggestion precise. A subset of vertices $X \subseteq G$ is an even cycle transversal, if $\mathsf{G} \setminus X$ contains no even cycles.

Proposition 4.10 *If $X \subseteq G$ is an even cycle transversal, then $X \in \mathcal{B}(\mathsf{G})$.*

Proof. For an arbitrary $\beta \in sol(\mathsf{G})$, Lemma 3.6 gives that $\overline{\beta|_X} = \beta|_{dom(\overline{\beta|_X})}$. Assume towards contradiction that $\mathsf{G}' = \mathsf{G} \setminus dom(\overline{\beta|_X}) \neq \varnothing$. By Lemma 3.6.2, G' has no sinks, and also $\forall x \in G' : \forall y \in E(x) \cap dom(\overline{\beta|_X}) : \beta(y) = \mathbf{0}$. This implies that $\beta = \beta|_{dom(\overline{\beta|_X})} \cup \beta'$ for some $\beta' \in sol(\mathsf{G}')$. However, as G' is a graph with no sinks and no even cycles we have $sol(\mathsf{G}') = \varnothing$ by Lemma 4.9. This is our contradiction. □

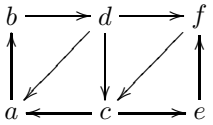
Clearly, for many graphs this represents a significant improvement over the algorithms from the previous subsection, reducing the worst case exponent from the number of cycles to the number of even cycles. Even though an implementation seeking to take advantage of this encounters the problem of finding an even cycle transversal (finding a minimum such is NP-hard, and not known to be FPT), one can argue also for the practical relevance of Proposition 4.10, besides the merely theoretical improvement. In some situations, it can happen that an even cycle transversal can be easily obtained from the input. In general, it often suffices to find a small – and not a minimum – such set and this can be done relatively efficiently.[8]

Example 4.11 (3.2, 4.4 continued) *The graph G has two even cycles: $\langle b, c, b \rangle$ and $\langle b, c, d, a, b \rangle$. Trying $b = \mathbf{1}$ (respectively, $\mathbf{0}$), induces the assignment $\overline{\alpha}$ (respectively, $\overline{\beta}$) as in Example 3.2. The induced assignments are functions and hence solutions by Proposition 4.10 and Definition 4.1.*

[8] Finding a minimum feedback vertex set is shown to be FPT in [5]. Given a graph with vertices G, and such a subset $V \subseteq G$, one can try moving, one at a time, a vertex x from V back to the induced subgraph $G \setminus V$, checking if the resulting, induced subgraph $G \setminus V \cup \{x\}$ has an even cycle. This last problem is in P by the recent result from [26]. If no even cycle appears, we continue with $V \setminus \{x\}$ and the induced subgraph extended with x, while if some does, x remains in V. What remains in V, after trying all its vertices, is an even cycle transversal.

The above example might be insufficient since b is, in fact, a feedback vertex set, so the conclusion follows already by Proposition 4.2. The following example, shows the difference.

Example 4.12 *In the following graph* G



the only even cycle is $C = \langle a, b, d, c, a\rangle$*. Breaking it at, say* a*, leads to two trials:*
$a = \mathbf{1}$ *induces* $b = c = d = \mathbf{0}$ *but then* $d = \mathbf{0}$ *gives a conflict inducing* $b = \mathbf{1}$*, and*
$a = \mathbf{0}$ *induces* $b = \mathbf{1}, d = \mathbf{0}$*, but no more nodes obtain induced value.*
Neither assignment induces a solution, so Definition 4.1 and Proposition 4.10 imply $sol(\mathsf{G}) = \varnothing$*.*
In the graph $\mathsf{G} \setminus \{e\}$*, we have the same even cycle. Trying* $a = \mathbf{1}$ *gives a conflict as above, but from* $a = \mathbf{0}$*, we obtain* $b = \mathbf{1} = c$ *and* $d = \mathbf{0} = f$*, yielding a solution.*

This concludes the first set of our algorithms for KER. Except for the obvious algorithm using $CNF(\mathsf{G})$ from (2.6), testing SAT (of boolean equations) is of use here only as a possible enhancement. The algorithms from the present section can be very efficient when applied to graphs with few (even) cycles and, particularly, when cycles or feedback vertex sets are easily read from the input. We do not think, however, that they will be optimal for all kinds of instances. Their likely shortcoming will arise from the comparison to the algorithm proposed in the following section, which also shows much tighter connections between KER and SAT.

5 KER and SAT

Algorithms in the previous section perform the initial simplification, Algorithm 3.15, extract a relevant subset X of vertices and then answer KER solving a system of equations or trying blindly assignments to X, which induce the assignments to the whole graph. The following, recursive Algorithm 5.1 performs simplification and induction at each recursive call, returning an element of $sol(\mathsf{G})$, if such exists, and $\perp$ otherwise. It takes an additional argument, the partial assignment α, and constructs its extension to a complete solution, if possible, or returns $\perp$ if not.

Algorithm 5.1 $sol(\mathsf{G}, \alpha)$

Input: A digraph G and a partial assignment α (initially $\alpha = \varnothing$).
Output: $\beta \in sol(\mathsf{G})$ with $\alpha \subseteq \beta$ if it exists, $\perp$ otherwise.

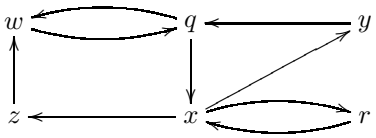
1: $\alpha := \overline{\alpha}$ ..... // Definition 3.1
2: **if** α is not a function **then return** $\perp$ **endif**
3: $\mathsf{G} := \mathsf{G} \setminus dom(\alpha)$
4: **if** $G = \varnothing$ **then return** α **endif**
5: $\mathsf{G} := simp(\mathsf{G})$ ..... // Algorithm 3.15
6: **if** $\mathsf{G} = \perp$ **then return** $\perp$ **endif**
7: Choose some $x \in G$
8: **return** $sol(\mathsf{G}, \alpha \cup \{\langle x, \mathbf{1}\rangle\}) \oplus sol(\mathsf{G}, \alpha \cup \{\langle x, \mathbf{0}\rangle\})$

Strictly speaking, this is not *an* algorithms but *a class* of algorithms. For instance, the simplification of a graph, as well as inducing of values from a given partial assignment, could be defined otherwise and replace those used here. Also, several minor issues are left for more detailed decisions. For instance, α is only a solution to the reduced graph obtained after a possible series of contractions at line 5. The solution for the actual input graph has to be reconstructed from it by a corresponding series of applications of Fact 3.10. The polynomial (in the size of the original graph) time, spent in each recursive call on the computation of $\overline{\alpha}$, can be improved since, once it starts going, it only needs to consider border vertices from $dom(\alpha)$, Definition 3.4. A conflict

(a vertex assigned two values so that $\overline{\alpha}$ is no longer a function) must occur at these vertices, and once it is detected, the algorithm can return the failure value at line 2.

Two more central decisions are left open. The operation $\oplus$ denotes angelic choice, ignoring the possible argument $\bot$, i.e., $x \oplus \bot = \bot \oplus x = x$, and $x \neq \bot \wedge y \neq \bot \rightarrow (x \oplus y) \in \{x, y\}$. An implementation has to decide how to perform the choice of the first value tried. Finally, we have not specified how to choose an $x \in G$ for branching at line 7. A specific instance of the above algorithm was presented in [11]. It performs no simplification and branches only from maximal degree cyclic vertices. Proposition 4.2 guarantees sufficiency of branching only from cyclic vertices, and choosing maximal degree is often sound.[9] However, it is not always the best choice, as evidenced by the following example.

Example 5.2 *Consider the graph* G*:*



The minimal degree is $2 = deg(y) = deg(z) = deg(r)$*. Branching from* y *gives two cases:*
$y = \mathbf{1}$ *induces the solution with* $x = q = z = \mathbf{0}$ *and* $y = w = r = \mathbf{1}$
$y = \mathbf{0}$ *induces the solution with* $q = z = r = \mathbf{1}$ *and* $x = y = w = \mathbf{0}$.
Similarly, each branching from z *induces a solution. So, Algorithm 5.1 branching first on some nodes with minimal degree, terminates successfully in just one recursive call. This need not happen when branching on* x*, the only vertex with maximal degree. Inducing from* $x = \mathbf{1}$ *gives* $y = q = \mathbf{0}$ *which yields a conflict at* y*, while* $x = \mathbf{0}$ *induces only* $r = \mathbf{1}$*, i.e., requires further recursive calls.*

It is probably too much to ask for an algorithm that always makes the optimal choice of branching vertex. As a simple and general rule for arbitrary graphs, choosing a vertex with maximal degree is probably a good solution. It is possible, however, to specify the choice of branching vertex in particular situations more carefully and thereby improve running times for certain classes of graphs. This is done for the class of oriented graphs in the following Subsection 5.1.

5.1 Oriented graphs

The worst case complexity of Algorithm 5.1 is $\mathcal{O}^*(2^{|G|})$, but many possibilities of improvements exist in the interplay of the different choices involved. An interesting, related question is how much the complexity can be improved when attention is restricted to special classes of graphs. We show that it is possible, by choosing $x \in \mathsf{G}$ for branching in a certain way, to obtain a better bound for the oriented graphs. The following lemma is the main tool for showing this improvement.

Lemma 5.3 *If* G *has no sinks then either all the final strongly connected components of* G *are cycles or there is some final strongly connected component* $S \subseteq G$ *with* $x \in S$ *such that* $|E^{\smile}(x)| \geq 2$*.*

Proof. Assume towards contradiction that G has a final strongly connected component S which is not a cycle and such that $\forall x \in S : |E^{\smile}(x)| \leq 1$. Clearly, since S is not a cycle there is some $x \in S$ such that $|E(x)| \geq 2$. Let y and z be two successors of some such x. Then since S is a final strongly connected component, there must be paths $\langle y = y_0, ..., y_{n-1}, y_n = x\rangle$ and $\langle z = z_0, ..., z_{m-1}, z_m = x\rangle$ in S. Since $y_n = z_m = x$ it follows that there is some maximal i such that $y_{n-i} = z_{m-i} = w$. Then we have $|E^{\smile}(w)| \geq 2$ □

Proposition 5.4 *For oriented* G *:* $|sol(\mathsf{G})| \leq 1.325^{|G|}$ *and Algorithm 5.1 can run in* $\mathcal{O}^*(1.325^{|G|})$*.*

[9] Whether degree refers to the in-degree, out-degree or their sum may depend on the implementation and, in particular, on the way of inducing. In our case, inducing happens both along and against the direction of the edges, so it is natural to take the degree of a node to be the number of all incoming and outgoing edges.

PROOF. We run Algorithm 5.1 and consider different recurrences depending on which vertex we branch on. First we make sure that we always branch on a vertex from a final strongly connected component of the current G. (Using Tarjan's algorithm, this can be ensured in linear time.) This component cannot be a cycle of odd length since then it is contracted, revealed to be a basic contradiction, and removed by Algorithm 3.15. It also follows by Algorithm 3.15 that if it is a cycle of even length then it is presented to us as a cycle of length 2. However, since G is oriented this cycle has been obtained as a result of contracting some isolated paths of length ≥ 3. Since this gave us an immediate reduction of the input size of at least 2, we are justified in writing the recurrence as $T(|G|) = T(|G| - 2)$. Assignment of $\mathbf{1}$ or $\mathbf{0}$ to any vertex in a cycle of length 2 with no outgoing edges forces a value to it's successor, and so we obtain the recurrence $T(|G|) = 2T(|G| - 4)$. If S is not a cycle then we branch on a vertex $x \in S$ such that $|E^{\smile}(x)| \geq 2$, guaranteed to exist by lemma 5.3. Then if $|E(x)| = 1$ we will force $\mathbf{1}$ to it's successor on assignment of $\mathbf{0}$, and $\mathbf{0}$ to all three neighbours on assignment of $\mathbf{1}$. So we obtain a worst case recurrence of $T(|G|) = T(|G| - 2) + T(|G| - 4)$. If $|E(x)| \geq 2$ we force values to four vertices when assigning $\mathbf{1}$ to x, and so we obtain $T(|G|) = T(|G| - 1) + T(|G| - 5)$. This latter bound serves as global worst case. Since the largest real root of the characteristic polynomial $\lambda^5 - \lambda^4 - 1 = 0$ is $\lambda \approx 1.325$ it follows that $|sol(\mathsf{G})| \leq 1.325^{|G|}$ and that Algorithm 5.1 can decide their (non)existence in time $\mathcal{O}^*(1.325^{|G|})$. □

Notice that the proof of Proposition 5.4 does not specify completely how to choose a branching vertex, but only narrows the choice down to sets of vertices with some desired properties. The question about the exact choice is still highly relevant for an implementation.

5.2 DPLL

If, in Algorithm 5.1, we take G to be a CNF formula, the algorithm turns out to be exactly the pseudo-code for the DPLL algorithm for satisfiability, [9, 8], which is the basis of virtually all modern SAT-solvers (for a relatively recent overview, one can consult e.g., [19]). Inducing in the first line amounts then, typically, to the unit propagation and the condition 'α is not a function' amounts to the 'conflict' in the SAT-solving parlance. An α satisfying all clauses in G is returned, line 4. Otherwise, the remaining problem is preprocessed for the next recursive call, line 5. Simplification may include elimination of clauses with pure literal (occurring only positively or only negatively), as well as learning and many other heuristics depending on the implementation. We suggested, similarly, a wide range of possible choices in Section 3. Choosing then wisely the branching literal x is one of the crucial aspects of successful SAT-solvers.

The coincidence of Algorithm 5.1 and DPLL goes beyond the mere fact of both instantiating the general branch and bound schema. It involves also the fact that kernels can be seen as solutions, (2.2), and that during their gradual construction, partial assignments induce values to the neighbourhoods, in analogy to unit propagation and other constraint propagation techniques in SAT. One may therefore expect the lessons from SAT-solving to be relevant for KER-solving. The crucial aspects of SAT-solvers concern the efficiency and range of inducing values from a given, partial assignment, line 1, and the choices of the branching point and its value required to get the most out of the propagation of constraints implied by the performed choices, line 7. These two elements occupy the critical position, as SAT-solvers spend around 80% of time on this phase. It is reasonable to expect a similar situation in KER-solving. The importance of this aspect has been illustrated by the improved complexity bound $\mathcal{O}^*(1.325^{|G|})$ for oriented graphs, which was obtained due to such graphs enabling, at each recursive call, some minimal extension of the current partial assignment, thus reducing the remaining search space.

There seems to be no general guidance in actually performing the choice of branching vertex. High degree may often work well but, as we saw in example 5.2, is not necessarily optimal. It might be too much to ask for a strategy working best in all cases but uncertainty at this point may also reflect the lack of experience and overview of the problem instances. In SAT-solvers, the choice is performed depending on the subclass of instances for which the solver is designed. Choice

of the branching literal in solvers for random-SAT uses a lookahead procedure, which determines the reduction in the search space effected by each choice. Solvers for industrial-SAT can use the results of learning from the earlier encountered conflicts.

We have thus mentioned another important aspect: a SAT-solver is designed for a specific category of instances. A solver deciding SAT quickly on instances from industrial, or other rational and systematic contexts, may perform poorly on random instances. For random instances, "local search" heuristics for merely finding a solution may be extremely efficient but remain incomplete, being unable to conclude unsatisfiability. The winner of several categories of the SAT-competition last years, SATzilla, is actually a collection of various algorithms, which are only chosen appropriately depending on the analysis of the actual instance. The lack of one, uniform approach and the need to adjust solutions and heuristics to appropriately limited subclasses of instances is a general lesson from SAT. One can expect KER to face the same challenge of identifying such relevant subclasses. It is likely, however, that just as the DPLL schema is at the core of virtually all efficient procedures for solving SAT, so does Algorithm 5.1 express the core structure of efficient approaches for solving KER.

An important case of subclasses are those for which the problem becomes tractable. For instance, 2-SAT is NL-complete and Horn-SAT is P-complete. Search for sufficient conditions for kernel existence is an active research field, e.g., [1, 13, 14, 16, 3], with a recent overview in [4]. Further research should, in our opinion, consider also the problem of finding classes of graphs which may not admit kernels but have complexity bounds for the KER-problem below NP-completeness.[10]

Finally, let us mention an interesting SAT phenomenon – phase transition. When the clausal density (the ratio of number of clauses to the number of variables) is below 4, the theory is, with high probability satisfiable, while when it exceeds 4.5, the theory is almost certainly unsatisfiable. The instances with the clausal density around the transition value, 4.25, are the most difficult to solve. It is not obvious how to translate this into the graph language. Graph density (average degree) seems to be a relative of the clausal density, so one might conjecture that sparse graphs should be solvable with high probability (as are, e.g., all trees, dags and 50% of all cycles.) Very dense graphs might be expected to be relatively easy (e.g., kernels in a weakly complete digraph G (one with a complete underlying graph $\underline{\mathsf{G}}$) are exactly nodes x satisfying $E^{\smile}(x) = G \setminus \{x\}$) but should be expected to be unsolvable. A naive guess might expect the most difficult problems somewhere in the middle between these two extremes. This is partially confirmed by the tests of the algorithm presented in [11]. According to them, sparse graphs and graphs with density over 50% are relatively easy, while those with density around 15-20% are most difficult. On the other hand, it has been shown in [15] that the kernel problem is NP-complete for planar digraphs of degree at most 3, so the "easy" instances of KER can certainly be difficult enough. It remains to be seen if phase transition from SAT has a counterpart in KER and, if so, under what measure of graph density.

6 Conclusions

We have studied the problem, KER, of solvability of digraphs or, in the more standard language, of determining if a given digraph has a kernel. We began by observing its equivalence to the problem of satisfiability of propositional formulae, whether in usual or infinitary propositional logic. Seeing different applications of digraph kernels, in areas such as game theory and non-classical logics, it is conceptually rewarding in itself to see that kernels can be expressed – equivalently and naturally – as models of propositional theories.

We have proposed a series of graph reductions which preserve and reflect solvability and, being linear (or low polynomial), can be incorporated into the algorithms for KER-solving. In Section 4,

[10] One non-trivial result of this kind follows from the work done on stable matchings. An algorithm presented in [18] decides existence of stable matchings for the roommates problem in polynomial time. This solves KER in polynomial time for any digraph that is an orientation of a line graph and for which every weakly complete subgraph is acyclic.

we gave two such algorithms: one based on the extraction of a feedback vertex set, F, and another reducing the complexity even to $\mathcal{O}^*(2^{|E|})$, where E is an even cycle transversal. As a consequence, KER is FPT not only in the size of F but also of E. (As an instance of reducing KER to SAT, we gave a variant of the first algorithm where solving a reduced system of boolean equations replaced blind trials of all assignments to the sinks of a labeled dag, representing the input graph.) These algorithms can be expected to perform well on graphs with few (even) cycles and, especially, when even cycle transversal or, at least, feedback vertex set can be easily obtained from the input.

The question about a general algorithm for arbitrary instances of KER, led in Section 5 to another, new algorithm, which turns out to be virtually identical to the well-known DPLL algorithm, underlying modern SAT-solvers. From this we dare draw a series of conjectures for further development of the research on KER. It suggests that this final algorithm may outperform others on the large, practical instances of KER. This, however, will depend on more detailed decisions, because the presented sketch gives only a class of algorithms. It leaves open the possibility for further choices and improvements at points were such possibilities were realised or are still investigated in the context of SAT-solving. Experience with SAT-solving suggests that one will have to adjust choices and heuristics to specific subclasses of instances. As a particular case, we showed that, with a specific branching strategy, oriented digraphs guarantee a certain minimum of inducing during the recursive trials, allowing to reduce their worst case bound to $\mathcal{O}^*(1.325^{|G|})$.

We have shown that SAT-solving can be, to some extent, incorporated in KER-algorithms. More importantly and generally, however, solving KER appears to pose the same kind of choices and challenges, as met earlier in the design of SAT-algorithms. One can therefore expect that issues known from SAT, like those exemplified in Section 5.2, have graph-theoretic counterparts that will come up in the design of KER-algorithms. This itself may provide an independent motivation, and a specific direction, for the further study of KER. On the other hand, it does not seem unreasonable to expect that SAT-solvers may eventually benefit from KER-algorithms. The fact that KER can be formulated just as naturally in the language of graphs as in the language of logic or of game-theory, suggests that the problem can act as a useful point of reference for the exchange of ideas between these different fields. A better understanding of KER might very well foster a better understanding of the relationship between different problems that, apart from being computationally demanding, often appear to have little in common.

Having seen several new algorithms, the reader might expect also a report of their implementation and performance in practice. However, the analogy to SAT suggests that one should not rely here on any simple statements of the kind "algorithms perform well in practice". More precisely, any such statement should be qualified by a careful description of the instances and actual performance measures. Experimentation with various implementations seems to be, in the case of KER as it is in the case of SAT, an independent and extensive field of work, not to be dismissed in a few sentences. We leave this important aspect for future work.

References

[1] Martine Anciaux-Mendeleer and Pierre Hansen. On kernels in strongly connected graphs. *Networks*, 7(3):263–266, 1977.

[2] Claude Berge and Pierre Duchet. Recent problems and results about kernels in directed graphs. *Discrete Mathematics*, 86:27–31, 1990.

[3] Mostafa Blidia. A parity digraph has a kernel. *Combinatorica*, 6(1):23–27, 1986.

[4] Endre Boros and Vladimir Gurvich. Perfect graphs, kernels and cooperative games. *Discrete Mathematics*, 306:2336–2354, 2006.

[5] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of ACM*, 55(5):1–19, 2008.

[6] Vašek Chvátal. On the computational complexity of finding a kernel. Technical Report CRM-300, Centre de Recherches Mathématiques, Univeristé de Montréal, 1973. `http://users.encs.concordia.ca/~chvatal`.

[7] Nadia Creignou. The class of problems that are linearly equivalent to satisfiability or a uniform method for proving NP-completeness. *Theoretical Computer Science*, 145:111–145, 1995.

[8] Martin Davis, Georgs Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[9] Martin Davis and Hillary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[10] Yannis Dimopoulos and Vangelis Magirou. A graph theoretic approach to default logic. *Information and Computation*, 112:239–256, 1994.

[11] Yannis Dimopoulos, Vangelis Magirou, and Christos H. Papadimitriou. On kernels, defaults and even graphs. *Annals of Mathematics and Artificial Intelligence*, 20:1–12, 1997.

[12] Yannis Dimopoulos and Alberto Torres. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science*, 170(1-2):209 – 244, 1996.

[13] Pierre Duchet. Graphes noyau-parfaits, ii. *Annals of Discrete Mathematics*, 9:93–101, 1980.

[14] Pierre Duchet and Henry Meyniel. Une géneralization du théorème de Richardson sur l'existence de noyaux dans les graphes orientés. *Discrete Mathematics*, 43(1):21–27, 1983.

[15] Aviezri S. Fraenkel. Planar kernel and grundy with d$\leq$3, dout$\leq$2, din$\leq$2 are NP-complete. *Discrete Applied Mathematics*, 3(4):257–262, 1981.

[16] Hortensia Galeana-Sánchez and Victor Neumann-Lara. On kernels and semikernels of digraphs. *Discrete Mathematics*, 48(1):67–76, 1984.

[17] Gregory Gutin, Ton Kloks, Chuan Min Lee, and Anders Yeo. Kernels in planar digraphs. *Journal of Computer and System Sciences*, 71(2):174 – 184, 2005.

[18] Robert W. Irving. An efficient algorithm for the stable roommates problem. *J. Algorithms*, 6(4):577–595, 1985.

[19] Inês Lynce and João P. Marques-Silva. An overview of backtrack search satisfiability algorithms. *Annals of Mathematics and Artificial Intelligence*, 37:307–326, 2003.

[20] Eric C. Milner and Robert E. Woodrow. On directed graphs with an independent covering set. *Graphs and Combinatorics*, 5:363–369, 1989.

[21] John W. Moon and Leo Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:2328, 1965.

[22] Victor Neumann-Lara. Seminúcleos de una digráfica. Technical report, Anales del Instituto de Matemáticas II, Universidad Nacional Autónoma México, 1971.

[23] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, 2006.

[24] Moses Richardson. On weakly ordered systems. *Bulletin of the American Mathematical Society*, 52:113–116, 1946.

[25] Moses Richardson. Solutions of irreflexive relations. *The Annals of Mathematics, Second Series*, 58(3):573–590, 1953.

[26] Neil Robertson, Paul D. Seymour, and Robin Thomas. Permanents, pfaffian orientations and even directed cycles. *Annals of Mathematics (2)*, 150(3):929–975, 1999.

[27] Robert Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121, Oct. 1971.

[28] John von Neumann and Oscar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944 (1947).

[29] Michał Walicki and Sjur Dyrkolbotn. The graphical structure of paradox and inconsistency. 2010. [submitted].