Acta Informatica manuscript No.
(will be inserted by the editor)

An algorithmic study of switch graphs

Bastian Katz · Ignaz Rutter · Gerhard Woeginger

the date of receipt and acceptance should be inserted later

Abstract We derive a variety of results on the algorithmics of switch graphs. On the negative side we prove hardness of the following problems: Given a switch graph, does it possess a bipartite / planar / triangle-free / Eulerian configuration? On the positive side we design fast algorithms for several connectivity problems in undirected switch graphs, and for recognizing acyclic configurations in directed switch graphs.

1 Introduction

What is a switch graph? A *switch* s on an underlying vertex set V is a pair (p_s, T_s) where $p_s \in V$ is the *pivot* vertex and where $T_s \subseteq V$ is a non-empty set of *target* vertices. The vertex set V and some set S of switches on V together form a *switch graph* $G = (V, S)$. A *configuration* of a switch graph is a mapping $c : S \to V$ such that $c(s) \in T_s$ for all $s \in S$. The configuration selects exactly one edge $e_c(s) := \{p_s, c(s)\}$ for every switch $s \in S$, and thus yields a corresponding multiset $E_c := \{e_c(s) : s \in S\}$ of edges. The corresponding multi-graph is denoted $G_c := (V, E_c)$; see Figure 1 for an illustration. Biologically speaking, a switch graph represents the genotype of an entire population of graphs, and every configuration specifies the phenotype of one concrete member in this population.

A preliminary version of this work has appeared as B. Katz, I. Rutter, and G. Woeginger. An Algorithmic Study of Switch Graphs. In: *Proc. 35th Int. Workshop Graph-Theoretical Concepts in Computer Science (WG'09), p. 226–237, LNCS, Springer*

B. Katz and I. Rutter
Faculty of Informatics,
Karlsruhe Institute of Technology(KIT)
E-mail: {katz, rutter}@kit.edu

G. Woeginger
Department of Mathematics and Computer
Science, TU Eindhoven
E-mail: gwoegi@win.tue.nl

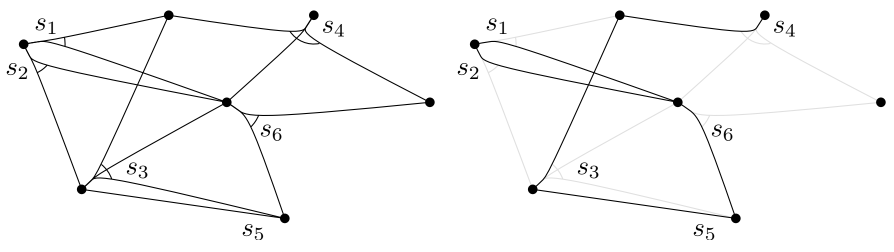


Fig. 1: To the left: A switch graph G with six switches, where s_5 has only a single target. To the right: A configuration yielding a multigraph G_c.

A brief history of switch graphs. Over the last 30 years a huge number of fairly unrelated combinatorial structures has been introduced under the name *switch graph* or *switching graph*; see the introduction of [6] for some pointers to the literature. The switching graph model of Meinel [11] comes very close to the model that is investigated in this paper. Another somewhat restricted type of switch graph has been introduced by Cook [1] who studied cyclic configurations as an abstraction of certain features in Conway's game of life. In Cook's model the vertices are not allowed to have degrees higher than three, and every switch has an obligatory incident edge that must show up in every configuration. Reinhardt [13] essentially studies Cook's model, but drops the constant degree constraint. Reinhardt constructs a polynomial-time $O(|V|^4)$ algorithm that decides whether there exists a configuration that contains a simple path between two prespecified vertices. He also links switch graphs to certain matching problems in computational biology [14]. We note that Cook's and Reinhardt's switch graph models can both easily be emulated by our switch graph model. Huckenbeck [8,9] studied a related but more general problem where the configuration of *valves* at nodes decides which pairs of incoming and outgoing edges of a node may be used by a path.

Groote and Ploeger [6] concentrate on switch graphs with *binary* switches (where every target set contains two elements). Their work is motivated by certain questions around the modal μ-calculus, and among other results they study the complexity of certain graph properties on switch graphs. For instance they show that in directed binary switch graphs, one can decide in polynomial time whether there is a configuration that connects (respectively disconnects) two prespecified vertices. Our current paper was heavily inspired by the conclusions section of [6]; our results in Theorems 3, 5, and 9 answer open questions that have been posed in [6].

Results of this paper. Every graph property $\mathcal{P}$ naturally leads to a corresponding algorithmic problem on switch graphs: Given a switch graph, does there exist a configuration with property $\mathcal{P}$? We will derive a collection of positive and negative results for various graph properties.

– It is NP-hard to decide whether a given switch graph has a configuration that is (a) bipartite, (b) planar, or (c) triangle-free. The three hardness proofs are presented in Section 3.
– We establish a number of matroid properties for switch graphs that possess a connected configuration. This yields a simple $O(|S| + |V|^2)$ time greedy al-

gorithm for finding a configuration that minimizes the number of connected components (and of course also settles the question whether there is a connected configuration); see Section 4.
– We provide a fast algorithm to detect a configuration that connects two given vertices in an undirected switch graph. This substantially improves the time complexity of Reinhardt's result [13]; see Section 5.
– Finding a configuration in which all vertex degrees are even is easy, but finding a configuration with a Eulerian cycle is NP-hard for forward directed switch graphs (see Section 2 for a formal definition), as well as for undirected switch graphs. Moreover, it is NP-hard to find a configuration that is biconnected (for undirected switch graphs) or strongly connected (for forward directed switch graphs); see Section 6.
– Deciding whether a forward directed switch graph allows an acyclic configuration can be done in linear time. In contrast to this, finding a configuration that minimizes the number of directed cycles is NP-hard; see Section 7.

We stress that our negative results hold in the most restricted binary switch model, whereas our positive results apply to the general model.

2 Basic Definitions

Let $G = (V, S)$ be a switch graph. For a subset $S' \subseteq S$ of switches and a configuration c, we denote $E_c(S') := \{e_c(s) : s \in S'\}$ and $G_c(S') := (V, E_c(S'))$. We denote $V(s) := T_s \cup \{p_s\}$ and $V(S') := \bigcup_{s \in S'} V(s)$. For $S' \subseteq S$ and $V' \subseteq V$, we denote by $S'(V') := \{s \in S' \mid V(s) \subseteq V'\}$ the set of *inner switches* of V'. Observe that $V(S'(V')) \subseteq V'$. A switch graph has *fan-out* k if $|T_s| \leq k$ for all $s \in S$. It is called *binary* if $|T_s| \leq 2$ holds for all $s \in S$. Throughout we will use $n := |V|$, $m := |S|$, and $\bar{m} := \sum_s |T_s|$. Note that $m \leq \bar{m} \leq km$ for the fan-out k of G. Throughout the paper we will frequently use the fact that a switch with a single target vertex corresponds to a "fixed" edge that occurs in any configuration of a switch graph. A *contraction* of a switch graph results from identifying several vertices of a switch graph. A contraction of a switch s in a switch graph is defined as the contraction of the switch graph identifying all vertices in $T_s \cup \{p_s\}$. For a switch s we further denote by $E(s) = \{\{p_s, t\} \mid t \in T_s\}$ the set of edges s may potentially create in different configurations.

Although the paper mainly deals with undirected graphs, all definitions easily carry over to directed switches and directed multi-graphs. In a *forward* switch $s = (p_s, T_s)$, arcs must be directed from pivot to target. In a *reverse* switch $s = (T_s, p_s)$, arcs must be directed from target to pivot. A *directed switch graph* may contain both, forward and reverse switches. A *forward directed switch graph* contains only forward switches.

Note that all problems we consider in this paper ask for configurations of a given switch graph with properties that can be tested in polynomial time. Moreover, there exist at most n^m distinct configurations of a switch graph, and thus $O(m \log n)$ bits are sufficient to guess a configuration as a certificate. This implies that all NP-hard problems presented in this paper are also NP-complete.

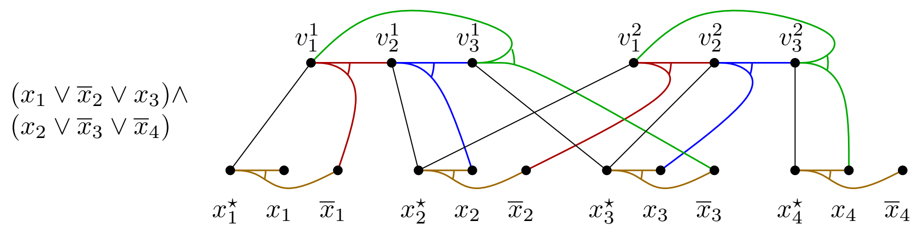


Fig. 2: Reduction of 3Sat to SwitchTriangleFree

3 Bipartite, Planar, Triangle-Free Graphs

In this section, we show hardness of finding configurations that are bipartite, triangle-free or planar.

Theorem 1 *For binary undirected switch graphs, it is NP-hard to decide if there is a bipartite configuration* (SwitchBipartite).

Proof We sketch a reduction from SetSplitting: Given a ground set $X = \{x_1, \ldots, x_n\}$ and a set T of 3-element subsets of X, it is NP-hard to decide whether there is a partition of X into two sets X_1, X_2, such that every $t \in T$ has non-empty intersection with both, X_1 and X_2 [5]. For a given instance of SetSplitting, we construct a switch graph $G = (V, S)$, containing vertices $x_1, \ldots, x_n$ for the elements of X. For each triplet $t_i \in T$ we introduce a switch $s_i = (x_j, t_i - \{x_j\})$ for an arbitrary $x_j \in t_i$.

Every solution X_1, X_2 to SetSplitting yields a bipartite configuration: Color the vertices x_i according to X_1, X_2. Then every triplet t contains both colors, which allows to set the corresponding switch to connect two vertices of distinct colors. Conversely, every bipartition of some configuration G_c induces a bipartition of the x_i. For any triplet in T, the switches prevent the corresponding three vertices from receiving all the same color, and thus the induced partition yields a solution to SetSplitting. □

Theorem 2 *For binary undirected switch graphs, it is NP-hard to decide if there is a triangle-free configuration* (SwitchTriangleFree).

Proof The proof is by reduction from 3Sat. Let φ be an instance of 3Sat. Without loss of generality we assume that each clause contains three different variables. We create a switch graph G_φ as follows. For each variable x_i we create the *variable vertex* $x_i^\star$, the two *literal vertices* x_i and $\bar{x}_i$, and a *variable switch* $s_i = (x_i^\star, \{x_i, \bar{x}_i\})$. For every clause C_j, we add three *clause vertices* v_1^j, v_2^j, v_3^j. If the kth literal in clause C_j corresponds to variable x_i, we introduce an edge $(v_k^j, \{x_i^\star\})$. If it is x_i, we introduce a *clause switch* $(v_k^j, \{v_{k+1}^j, \bar{x}_i\})$, defining $v_4^j := v_1^j$. If it is $\bar{x}_i$, we introduce a clause switch $(v_k^j, \{v_{k+1}^j, x_i\})$. See Figure 2 for an example. Clearly the reduction can be performed in polynomial time. It remains to show that φ admits a satisfying truth assignment if and only if G_φ admits a triangle-free configuration.

Suppose that φ admits a satisfying truth assignment. We define a corresponding triangle-free configuration c of G_φ as follows. For each variable switch s_i

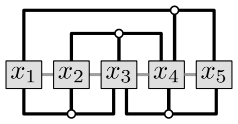


Fig. 3: Planar layout of the graph of the planar monotone 3Sat formula $(x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5)$. Clauses above and below the x-axis contain only positive and negative literals, respectively.

we set $c(s_i) = x_i$ if and only if x_i has the value *true*, for a clause switch $s = (v_k^j, \{v_{k+1}^j, x_i\})$, which corresponds to the literal $\bar{x}_i$ occurring in clause C_j, we set $c(s) = x_i$ if and only if x_i has the value *false*, and for $s = (v_k^j, \{v_{k+1}^j, \bar{x}_i\})$ we set $c(s) = \bar{x}_i$ if and only if x_i has the value *true*. Intuitively, the variable switch picks the satisfied literal, and clause switches whose literal is satisfied pick the literal vertex corresponding to the complementary literal. Correspondingly, a clause switch whose literal is not satisfied picks a clause vertex. We claim that this yields a triangle-free asignment. To see this, observe that each clause contains a satisfied literal, and hence at least one of its clause switches picks a literal vertex. Hence, no three clause vertices form a triangle. A triangle $v_k^j x_i x_i^\star$ ($v_k^j \bar{x}_i x_i^\star$) implies that x_i has the value *true* (*false*) and that C_j contains the literal $\bar{x}_i$ (x_i). But then c picks for the corresponding clause switch the target v_{k+1}^j and not x_i ($\bar{x}_i$), and the triangle does not exist. Since any triangle must fall into one of the categories, it follows that c is triangle-free.

Conversely, assume that c is a triangle-free configuration of G_φ. We define a corresponding truth assignment by setting x_i to *true* if and only if $c(s_i) = x_i$. Consider a clause C_j. Since c is triangle-free, at least one of the clause switches corresponding to C_j must have picked a literal vertex. Let $s = (v_k^j, \{v_{k+1}^j, x_i\})$ be such a switch (the case $s = (v_k^j, \{v_{k+1}^j, \bar{x}_i\})$ can be handled analogously). The existence of switch s implies that C_j contains the literal $\bar{x}_i$. By the choice of s we have $c(s) = x_i$, and since c is triangle-free, it follows that $c(s_i) = \bar{x}_i$. But then by definition of the truth assignment x_i has the value *false* and C_j is satisfied by the literal $\bar{x}_i$. Hence the truth assignment indeed satisfies φ. □

Theorem 3 *For binary undirected switch graphs, it is NP-hard to decide if there is a planar configuration* (SwitchPlanar).

Proof The proof is by reduction from Planar Monotone 3Sat. Planar 3Sat is a well-known NP-hard restriction of 3Sat, where additionally the variable–clause graph is assumed to be planar. The problem Planar Monotone 3Sat is even more restricted: the literals of each clause must be either all positive or all negative. Moreover, the variable clause graph can be drawn in the plane without crossings such that all the variables are on the x-axis, the clauses with positive literals are above the x-axis and the clauses with negative literals are below the x-axis; see Figure 3 for an example. Planar Monotone 3Sat is NP-hard [3].

Now let φ be an instance of Planar Monotone 3Sat together with a corresponding layout. Note that this defines a total ordering (from left to right) of the variables along the x-axis, for each clause a *left*, *middle* and *right* variable, and a

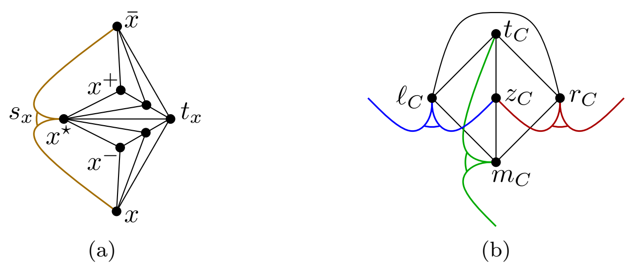
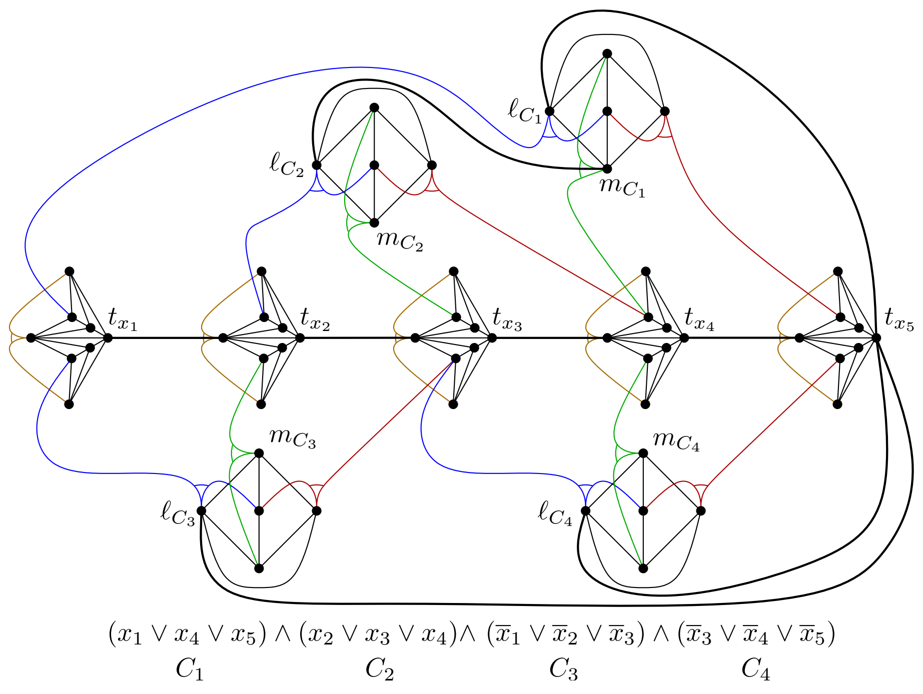


Fig. 4: Gadgets for the reduction of Planar Monotone 3Sat to SwitchPlanar. (a) A variable gadget with literal vertices x^+ and x^-, where literal switches will be attached. The variable gadget contains a switch s_x that either picks x or $\bar{x}$, corresponding to satisfying the corresponding literal. (b) A clause gadget with literal vertices ℓ_C, m_C and r_C that forms a K_5 if all three literal switches choose a target inside the clause gadget.

nesting of some of the clauses above and below the x-axis. For every variable x, we introduce a variable gadget as depicted in Figure 4a with one *variable switch* s_x whose targets are the two *literal vertices* x and $\bar{x}$. For each pair x and y of consecutive variables (y is the successor of x), we connect t_x to $y^\star$. For every clause C, we introduce a clause gadget as depicted in Figure 4b, which basically is a K_5 missing three edges, which may later be enabled or disabled by appropriately setting the *literal switches*, which we will define below. Further, let z denote the right-most variable in the drawing. We connect t_z to the left literal vertex ℓ_C of each outermost clause C, and for each clause C we connect the middle literal vertex m_C to the left literal vertex $\ell_{C'}$ of all clauses C' that are directly enclosed by C. Denote the set of edges added in this way by T. Finally, we add the literal switches. For each variable x occurring positively in a clause C, we define a switch as follows. If x is the left variable of C, we add the switch $s_x^C = (\ell_C, \{x^+, z_C\})$. If x is the right variable of C, we add the switch $s_x^C = (r_C, \{x^+, z_C\})$. If x is the middle variable, we add the switch $s_x^C = (m_c, \{x^+, t_C\})$. For negative literals, we replace x^+ by x^-. Let G_φ denote the resulting switch graph; see Fig. 5 for a complete example. We now claim that G_φ has a planar configuration if and only if φ is satisfiable.

Note that each literal switch has one target belonging to a clause gadget and one target belonging to a variable gadget. Given a configuration, we say that a literal switch s_x^C is *switched into the clause*, if $c(s_x^C)$ is a vertex of a clause gadget, otherwise it is *switched into the variable*. We first consider the variable gadget for a variable x. In a planar configuration, if $c(s_x) = \bar{x}$, then no literal switch s_x^C of a positive clause C may be switched into the variable. Otherwise, we contract all clause gadgets and all variable gadgets except the one of x to a single vertex, each. Now there exists a path from t_x to the now contracted clause C whose edges are in T. Now the upper half of the variable gadget of x together with this path along with the edge $\{\ell_C, x^+\}$ created by switch s_x^C and the edge $\{x^\star, \bar{x}\}$ created by switch s_x form a subdivision of K_5, contradicting planarity. Analogously, in a planar configuration literal switches s_x^C of negative clauses, may not be switched into the variable if $c(s_x) = x$. Thus s_x intuitively picks either x or $\bar{x}$ as satisfied, and all literal switches of the opposite literal must be switched into the clause. A



ℓ_{C_1}
m_{C_1}
ℓ_{C_2}
m_{C_2}
t_{x_1}
t_{x_2}
t_{x_3}
t_{x_4}
t_{x_5}
m_{C_3}
ℓ_{C_3}
m_{C_4}
ℓ_{C_4}
$(x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_3 \vee \overline{x}_4 \vee \overline{x}_5)$
C_1
C_2
C_3
C_4

Fig. 5: Example reduction of Planar Monotone 3Sat to SwitchPlanar for the formula whose variable-clause graph is shown in Fig. 3. The edges of T are drawn as thick black edges.

clause gadget with all three literal switches switched into the clause forms a K_5. Thus, in a planar configuration this does not occur and each clause must contain at least one satisfied variable.

On the other hand, it is not hard to see that a satisfying truth assignment of φ yields a planar configuration of G_φ; the variable switch picks the satisfied literal, and we switch each unsatisfied literal switch into the clause and each satisfied literal switch into the variable. To see that this is planar, observe that there are at most two switches switched into each clause, and the corresponding edges can be drawn inside the clause gadget without crossings and thus do not interfere with the outside drawing. The edges stemming from switches that are switched into the variable can then be drawn as in the original layout of the formula. Finally the edges of T can be added in a planar way since they adhere to the nesting structure of the clauses by construction. □

4 Global Connectivity

In this section we discuss the question whether a given switch graph $G = (V, S)$ has a connected configuration. It turns out that this question has many ties to matroid theory, which allows us to invoke some powerful machinery from mathematical programming.

First, we consider two matroids $(E, \mathcal{I}_1)$ and $(E, \mathcal{I}_2)$ that both have the same ground set E, which is the set of all the multi-edges over V that can possibly result from switches in S. Note that we consider the elements of E to be associated with their corresponding switch, and hence we consider two elements $e_1 \in E(s_1)$ and $e_2 \in E(s_2)$ with $s_1 \neq s_2$ as distinct, even if they have the same endpoints. Formally, the set E is the *disjoint* union $\bigcup_{s \in S} E(s)$ of edges that may be created by switches. The set system $\mathcal{I}_1$ consists of all cycle-free subsets of E. The set system $\mathcal{I}_2$ consists of all subsets of E that contain at most one edge from each switch, that is $E' \subseteq E$ is in $\mathcal{I}_2$ if and only if $|E' \cap E(s)| \leq 1$ for all $s \in S$. Then $(E, \mathcal{I}_1)$ forms a *graphic matroid* and $(E, \mathcal{I}_2)$ forms a *partition matroid.* The fact that the acyclic edge sets of a multi-graph form a matroid is well known [10]. For the partition matroid, we clearly have that $\emptyset \in \mathcal{I}_2$ and the hereditary property holds trivially. For the exchange property assume that $A, B \in \mathcal{I}_2$ and $|A| < |B|$. Then there exists a switch s such that $A \cap E(s) = \emptyset$ but $B \cap E(s) = \{e\}$. It follows that $A \cup \{e\} \in \mathcal{I}_2$, and thus the exchange property holds.

Note that the sets in $\mathcal{I}_1$ are exactly the acyclic edge sets of E, while $\mathcal{I}_2$ contains edge sets that can be realized by configurations, as they contain at most one edge from each switch. Obviously, the switch graph $G = (V, S)$ has a connected configuration, if and only if there exists a spanning tree that can be realized by a configuration of the switches or, equivalently, if there exists a set $E' \subseteq E$ of cardinality $n - 1$ that belongs to both $\mathcal{I}_1$ and $\mathcal{I}_2$. This is a standard matroid intersection problem, which can be solved in polynomial time [4].

It is not hard to see that the intersection $(E, \mathcal{I}_1 \cap \mathcal{I}_2)$ itself does not form a matroid. In the following, we will model the problem in terms of a single matroid, which yields simpler and faster algorithms. Our approach is based on a third structure $(S, \mathcal{I}_3)$ that is defined over the ground set S of switches. A subset $S' \subseteq S$ lies in $\mathcal{I}_3$, if there exists a configuration c such that $E_c(S')$ is cycle-free (or in other words, such that $E_c(S')$ belongs to $\mathcal{I}_1$). The sets in $\mathcal{I}_3$ are called *independent* sets.

Theorem 4 *The structure $(S, \mathcal{I}_3)$ forms a matroid.*

Proof Clearly the set system $\mathcal{I}_3$ contains the empty set and is closed under taking subsets. It remains to show that for two independent sets $A, B \subseteq S$ with $|A| < |B|$, there is an $s \in B - A$ such that also $A \cup \{s\}$ is independent.

Since A and B are independent, there exist configurations a and b for which the corresponding edge sets $E_a(A)$ and $E_b(B)$ are cycle-free. Among all such configurations a and b, we consider a pair that maximizes the number of switches that are in $A \cap B$ and that configure into the same edge both in configuration $E_a(A)$ and in configuration $E_b(B)$; such switches are called *good* switches. Since $E_a(A)$ and $E_b(B)$ are cycle-free, they belong to $\mathcal{I}_1$ in the underlying graphic matroid. Since $|E_a(A)| = |A| < |B| = |E_b(B)|$, by the matroid property of $\mathcal{I}_1$, there exists an edge $e \in E_b(B) - E_a(A)$ such that $E_a(A) \cup \{e\}$ is cycle-free.

Let $s_e \in B$ denote the switch that in configuration b generates edge e. We claim that this switch s_e cannot be in A: Otherwise configuration a would configure this switch s_e into an edge f. Then we can modify configuration a into a new configuration c by switching s_e into e instead of f. The resulting edge set $E_c(A)$ is still cycle-free since even $E_a(c) \cup \{e\}$ is cycle-free, whereas the number of good switches has increased. That is a contradiction. Hence $s_e \notin A$, and $A \cup \{s_e\}$ is an independent set of switches. □

Our next goal is to get a better understanding of independence in $(S, \mathcal{I}_3)$.

Lemma 1 *A set S' of switches is independent if and only if $|T| < |V(T)|$ holds for all non-empty subsets $T \subseteq S'$.*

Proof One direction of the proof is easy: If there is a non-empty $T \subseteq S'$ with $|T| \geq |V(T)|$, then every configuration c induces a cycle on T since $|E_c(T)| = |T| \geq |V(T)|$ holds.

For the other direction of the proof, we show that if $|T| < |V(T)|$ holds for all nonempty sets $T \subseteq S'$, then S' is independent. The proof is by induction on $|S'|$. Clearly, the claim holds for $|S'| = 0$. Now let $|S'| \geq 1$, let $s \in S'$ be an arbitrary switch, and let $S'' := S' \setminus \{s\}$. Since $S'' \subseteq S'$, we have $|T| < |V(T)|$ for all $T \subseteq S''$, and since $|S''| < |S'|$, the induction hypothesis yields that S'' is independent. Thus, there exists a configuration c of S'' such that $E_c(S'')$ is acyclic. Adding an arbitrary edge e from the switch s to $E_c(S'')$ produces a configuration c for S' whose edge set $E_c(S') = E_c(S'') \cup \{e\}$ contains at most one cycle. If $E_c(S')$ is acyclic, then S' is independent and we are done. Hence, assume that $E_c(S')$ contains a single cycle C.

We now describe a procedure that, given a configuration c of a switch set S' such that $E_c(S')$ contains exactly one cycle, either detects a non-empty subset T of switches of S' with $|T| \geq |V(T)|$ or produces from c an acyclic configuration c' of S'. Observe that the existence of such a procedure directly implies the claim of the lemma since the former outcome would contradict our assumption, and thus the procedure gives an acyclic configuration c' of S', showing that S' is independent.

Recall that $E_c(S')$ contains a single cycle C. Let $E := E_c(S')$, let C_0 denote the vertices of the connected component of $(V, E_c(S'))$ that contains C, and let V_0 denote the set of all vertices in $V(S')$ that are *not* in C_0. We work through a number of removal phases. In each phase, we remove a switch from S' and also remove the corresponding edge from E. Moreover, we construct two sets V_i and C_i, maintaining the invariant that after phase i the sets $V_0, \ldots, V_i$ and C_i form a partition of $V(S')$, such that the connected component of (V, E) induced by C_i contains the cycle C and no edge of E connects vertices from distinct sets of the partition. Observe that initially, before phase 1, we have $i = 0$ and V_0 and C_0 form such a partition.

In phase $i \geq 1$, we have as input a partition $V_0, \ldots, V_{i-1}, C_{i-1}$ of $V(S')$. A switch $s = (p, T)$ is *removable*, if it contributes an edge e to the connected component induced by C_{i-1} and its target set T contains a target $t^\star$ outside of C_{i-1}. If no such switch exists, the removal process stops. Otherwise let $s_i = (p_i, T_i)$ be a removable switch with $t_i^\star \in T_i \cap V_\ell$ for some $0 \leq \ell \leq i-1$. We set $\mathrm{ref}(i) := \ell$, remove from E the edge e_i in C_{i-1} corresponding to s_i and consider the connected components of the subgraph of (V, E) induced by C_{i-1} after this removal. We define C_i to be the vertex set of the connected component containing the cycle C. If no such component exists, i.e., e_i is an edge of C, then we set $C_i := \emptyset$. We further set $V_i := C_{i-1} \setminus C_i$. It is not hard to see that this maintains the claimed invariant. Then the $(i+1)$th removal phase starts.

Clearly, the number of removal phases is bounded by the number of switches. Suppose that the process terminates after k successful removals. This means that no removable switch exists in phase $k+1$. Then either (i) the set C_k is empty, i.e., the edge e_k removed in phase k was part of the cycle C, or (ii) we have $C_k \neq \emptyset$, but no switch contributing an edge to C_k has a target outside C_k. In case (ii)

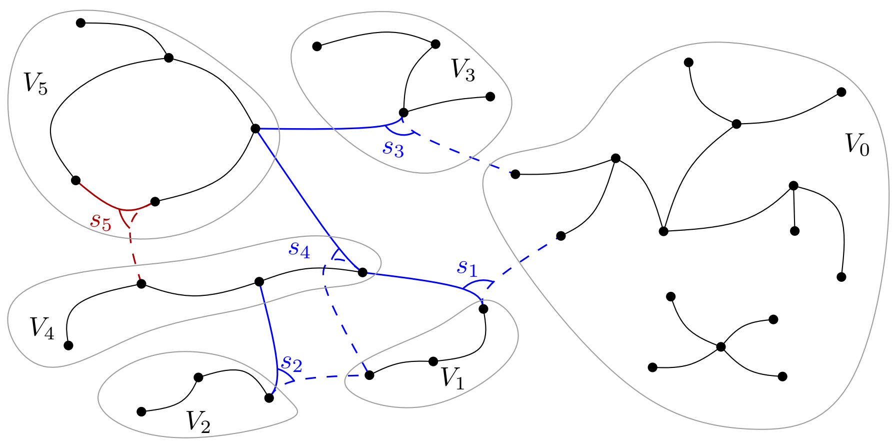


Fig. 6: A proper switch sequence $s_1, \dots, s_5$ that eventually breaks the cycle, the dashed edge of switch s_i connects p_i with the target $t_i^\star$, witnessing that s_i can be removed in step i. Switches are reinserted changing s_5, s_4, s_1.

let T denote the set of switches contributing an edge to C_k. Since no switch in T is removable, we have $V(T) = C_k$. Moreover, since $E_c(T)$ forms on C_k a connected component containing a cycle, we have $|T| \geq |C_k|$. It then follows that $|T| \geq |V(T)|$, and we output the set T. In case (i) we will show how to reinsert and how to reconfigure the removed edges and switches step by step in reverse order $s_k, s_{k-1}, \dots, s_1$ so that the resulting edge set is cycle-free. See Figure 6 for an illustration.

Throughout we will maintain the following invariant: Just after the insertion and reconfiguration of switch s_i $(1 \leq i \leq k)$, there exists an index $\ell(i)$ with $0 \leq \ell(i) < i$, such that the vertex set $V_{\ell(i)} \cup \bigcup_{h \geq i} V_h$ forms a cycle-free connected component with respect to the current edge set. This component is called the *crucial* component; intuitively speaking we will make it grow until it covers all of $V(S')$. We start the growing process with switch s_k, which by definition has a target t_k^* in the set $V_{\mathrm{ref}(k)}$ with $\mathrm{ref}(k) < k$. By reinserting the edge $\{p_k, t_k^*\}$ for switch s_k and by setting $\ell(k) := \mathrm{ref}(k)$, we satisfy the invariant. In handling a switch s_i with $i < k$ we distinguish two cases: First, if $i \neq \ell(i+1)$, then we simply reinsert its old edge e_i and keep $\ell(i) := \ell(i+1)$. This merges the vertices in V_i into the crucial component while maintaining the invariant. In the second case $i = \ell(i+1)$. We reconfigure switch s_i to pick target t_i^*, insert the corresponding edge $\{p_i, t_i^*\}$, and set $\ell(i) := \mathrm{ref}(i)$. This merges the vertices in $V_{\mathrm{ref}(i)}$ into the crucial component, and again maintains the invariant. This reconfiguration process eventually produces a cycle-free configuration for S', and thus completes the proof. □

The statement of Lemma 1 is combinatorial, but its proof is algorithmic and the procedure described in the proofs is a fast independence test for the matroid $(S, \mathcal{I}_3)$: Given an acyclic configuration of an independent set S'' we can check in $O(kn)$ time whether a given switch s can be added to S'' without destroying independence. To see the running time note that the independence of S'' implies that $|S'' \cup \{s\}| \leq n$, which also bounds the number of removal phases. As in the proof of Lemma 1, we denote the set $S'' \cup \{s\}$ by S'. To achieve selection of removable switches within a total of $O(kn)$ time, we direct all edges in $E_c(S')$ that are not part of the cycle to point away from it. Whenever a switch s_i is removed, we use this information to mark all vertices in V_i. Obviously, this only adds $O(n)$ time. A switch s is a candidate if it has a marked target and both p_s and $c(s)$ are unmarked. A set of candidates can be maintained in $O(kn)$ total time.

If the independence test is positive, we obtain a corresponding cycle-free configuration for S'. This already implies that the standard greedy algorithm for matroids, which requires m independence tests, computes a largest independent set of switches in $O(mnk)$ time. However, we can do somewhat better by exploiting that also negative outcomes of the independence tests convey some information.

If the test is negative, we get the final component C_{k-1} that contains the cycle. Let U denote the set of all switches in S'' that contribute an edge to C_{k-1}. Then $|U| = |V(C_{k-1})| - 1$, and none of the switches in U has a target outside of $V(C_{k-1})$. Hence in any cycle-free configuration of S'' the switches in U induce a connected graph on $V(C_{k-1})$; such a set U of switches is called a *tight* set. We exploit this by contracting all switches in the tight set U. The benefit is that now each independence test either increases the size of the independent set of switches or reduces the number of vertices, and thus the total number of independence tests is $O(n)$. These ideas lead to the following theorem, which is the main result of this section. Note that it as a special case yields a polynomial-time algorithm for recognizing switch graphs with connected configurations.

Theorem 5 *For a given switch-graph with fan-out k, we can determine in $O(km + kn^2)$ time a configuration that minimizes the number of connected components.*

Proof Any basis $\mathcal{B}$ of the matroid $(S, \mathcal{I}_3)$ yields a cycle-free configuration with the maximum number of edges, and hence a configuration with the minimum number of connected components; the switches not in $\mathcal{B}$ then can be set arbitrarily. Hence it is enough to determine a basis, and this is done by the standard greedy algorithm for matroids, as follows.

We start with $S' = \emptyset$ and a trivial acyclic configuration c of S', and test the switches one by one to see whether adding them to S' yields an independent set. If the test is positive, we add the switch to S' and update the cycle-free configuration c. If the test is negative, we forget the switch and contract all switches in the corresponding tight set U. These contractions can be done in overall $O(n\alpha(n))$ time by using a union-find data structure, where $\alpha(n)$ denotes the slowly growing inverse of the Ackermann function. Every test on a non-trivial graph costs $O(kn)$ time. Every positive test adds an edge to a cycle-free edge set; hence there are at most $n - 1$ of these tests. Every negative test on a non-trivial graph contracts some vertices; hence there are at most $n - 1$ of these tests. Every negative test on a trivial graph (that has been contracted to a single vertex) costs $O(k)$ time. All in all, this yields the claimed time complexity. □

Global connectivity and bipartite matching. Although not obvious at first sight, there is a strong similarity between global connectivity in switch graphs and bipartite matching. Both problems can be expressed as an intersection of two matroids and the characterization of independent sets of switches is very similar to Hall's theorem [7]. In this section we show that this similarity is no coincidence and that in fact, bipartite matching can be expressed in terms of global connectivity and that in this case the characterization of independent switch sets is exactly Hall's theorem.

Let $G = (A \cup B, E)$ be a bipartite graph with $|A| = |B|$. We construct a switch graph $G' = (V, S)$ as follows. Let V consist of B and a vertex s. Now for each vertex $a \in A$ we create a switch $s_a = (s, N(a))$ where $N(a)$ denotes the neighbors of a in G. The graph G has a perfect matching if and only if G' has a connected configuration. This is the case if and only if all $|A|$ switches of G' are independent, i.e., $|S'| < |V(S')|$ holds for every $S' \subseteq S$. By the construction of G' this is equivalent to the statement that every set $A' \subseteq A$ has at least $|A'|$ neighbors, which is Hall's theorem [7].

SwitchConnect-T. We further briefly analyze a generalization of the global connectivity problem. The problem SwitchConnect-T is defined as follows. Given a switch graph $G = (V, S)$ and a set $T \subseteq V$ of terminals, does there exist a configuration c such that in G_c all vertices of T are in the same connected component?

Theorem 6 SwitchConnect-T *is NP-hard for binary switch graphs.*

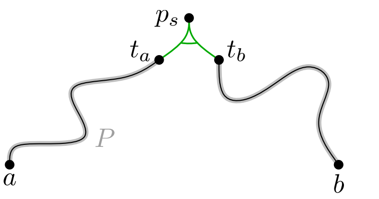
Proof We first show hardness for switch graphs that are not binary. We reduce from 3Sat. Let φ be an instance of 3Sat. We construct a switch graph G_φ as follows. For each variable x we create two literal vertices v_x and $v_{\bar{x}}$. For each clause C we create a vertex v_C and a switch $s_C = (v_C, T_C)$ where T_C is the set of vertices corresponding to the literals that occur in C. Finally, we add one new node s and for each variable x a switch $s_x = (s, \{v_x, v_{\bar{x}}\})$. Let T be the set that contains all clause-vertices and the node s.

We claim that G_φ has a configuration that connects T if and only if φ has a satisfying truth assignment. Given a truth assignment of φ we construct a configuration of G_φ as follows. For each variable x we set $c(s_x) = v_x$ if x has the value *true* and $c(s_x) = v_{\bar{x}}$ otherwise and for each clause C we set $c(s_C) = v$ where v is a vertex that corresponds to a satisfied literal of C. Conversely, from a configuration c that connects T we can find a truth assignment by setting x to *true* if $c(s_x) = v_x$ and *false* otherwise. The correctness of the claim follows from the fact that a clause vertex v_C is connected to s if and only if the corresponding clause C is satisfied by the truth assignment. Since the reduction can be performed in polynomial time, it follows that SwitchConnect-T is NP-hard.

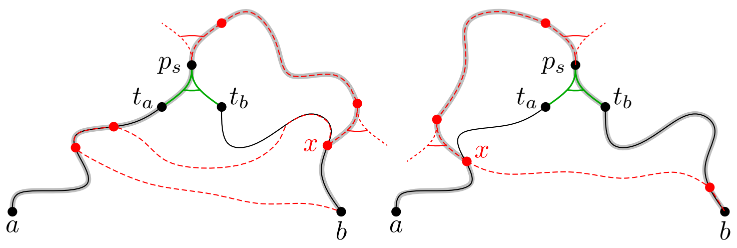
To reduce to binary switches, we replace for each clause C the clause switch $s_C = (v_C, \{x, y, z\})$ by two switches $s'_C = (v_C, \{x, v'_C\})$ and $s''_C = (v'_C, \{y, z\})$, where v'_C is a new vertex. Note that we do not change T, i.e, v_C is contained in T, but v'_C is not. This guarantees that again v_C can be connected to exactly one of the literal vertices occurring in the clause C. The rest of the proof is analogous. □

Although SwitchConnect-T is NP-hard in general, it can be solved in polynomial time for some special cases. We have already shown that it can be solved efficiently in the case $T = V$. In the next section we will show that the problem can be solved in polynomial time if $|T| = 2$.

Moreover, from the polynomial-time algorithm for $T = V$ it follows that the problem is fixed-parameter tractable (FPT) with respect to the parameter $k' := n - |T|$ since we can enumerate all possible subsets V' of $V \setminus T$ and solve the corresponding connectivity instance $G - V'$. It is an open question whether SwitchConnect-T is FPT with respect to $|T|$ or even if it can be solved in polynomial time if $|T| = 3$.



(a) Non-trivial case: P (grey) connects both endpoints a and b with targets of s



(b) The forward path (dashed) from p_s to b hits a first vertex x on P, allowing to bypass either $p_s - t_b$ or $p_s - t_a$ without altering a switch on P.

Fig. 7: A path in $G'_{c'}$ witnesses a path in G_c for some configuration c.

5 Local Connectivity

In this section, we investigate configurations that connect two given vertices a and b by a path. In the following, we call a sequence of switches a *forward path* if every switch's pivot is a target of its predecessor. Recall that a contraction of a switch s in a switch graph is defined as the switch graph identifying all vertices in $T_s \cup \{p_s\}$.

Lemma 2 *Let $G = (V, S)$ be a switch graph and s be any switch such that in $(V, S - s)$, there is a (possibly trivial) forward path from p_s to b. Let G' be the result from contracting s. Then G can be switched to connect a and b if and only if this is possible for G'.*

Proof First, by contracting a switch, it is not possible to lose connectivity. We will thus assume that it is possible to find a configuration c' that connects a and b in G' and show that this witnesses such a configuration c for G. We denote the path in $G'_{c'}$ as a sequence of switches P. Clearly, the switches of P forms either one or two paths in $G_{c'}$. If P forms a single path in $G_{c'}$, or if P connects either a or b to p_s, finding a connecting configuration is trivial. Otherwise, P forms two paths, connecting a to some $t_a \in T_s$ and b to some $t_b \in T_s$. In this situation, depicted in Figure 7, we can make use of the fact that in $S - s$, there is a forward path from p_s to b. Its first switch is not part of P, and we simply follow the forward path until we hit some vertex x on P. Now, switching the forward path from p_s to x gives us a bypass either from p_s to t_a or from p_s to t_b and switching s accordingly connects a and b. □

This lemma provides a simple test for a-b-connectivity: If there is a configuration c that connects a and b in G_c, there either is a forward path from b to a or there

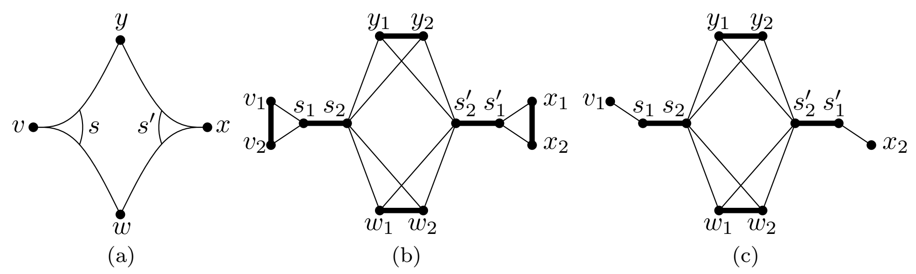


Fig. 8: Example for the reduction of local connectivity to finding an augmenting path. A switch graph (a) and corresponding graph H with matching M drawn as bold edges (b). The augmenting path $v_1s_1s_2w_1w_2s'_2s'_1x_2$ in $H\setminus\{v_2, x_1\}$ corresponds to the path vwx in G (c). The fact that $H \setminus \{y_2w_1\}$ does not admit a perfect matching shows that there is no path from y to w in G.

is a contractable switch, since there must be a first switch that is used "forward" on the path from a to b in G_c. The proof of Lemma 2 is constructive, naively implemented, it yields an $O(n^2 + n\bar{m})$ time algorithm to test the existence of and compute a connecting configuration by storing a forward path for each contraction. It is not difficult to improve the running time for the problem of deciding whether an ab-path exists to almost linear time by using a Union-Find data structure. However, it is not as easy to also provide a corresponding path if it exists. Instead we show that the local connectivity problem is in fact equivalent to the problem of finding an augmenting path with respect to a matching. This yields a fast algorithm for solving the local connectivity problem.

Theorem 7 *For a given switch-graph $G = (V, S)$ and two vertices $a, b \in V$, we can determine in $O(\bar{m}+n)$ time a configuration that connects a and b, if such a configuration exists.*

Proof Let $G = (V, S)$ be a switch graph and let a and b two vertices of G. We construct a new graph $H = (V_H, E_H)$ and a matching of H as follows. For each vertex $v \in V$ we create two *node vertices* v_1 and v_2 in V_H and add to E_H the edge $e_v = \{v_1, v_2\}$. We call these edges *node edges*. For each switch $s = (p, T) \in S$ we create two *switch vertices* s_1 and s_2 in V_H and we add the edges $\{s_1, p_1\}$ and $\{s_1, p_2\}$, which we call *pivot connector edges*, the *switch edge* $\{s_1, s_2\}$, and for each of its targets $t \in T$ the edges $\{s_2, t_1\}$ and $\{s_2, t_2\}$, called *target connector edges*. We choose the matching $M = \{\{v_1, v_2\} \mid v \in V\} \cup \{\{s_1, s_2\} \mid s \in S\}$. We now claim that there is a configuration that connects a and b in G if and only if there exists an augmenting path from a_1 to b_2 in $H\setminus\{a_2, b_1\}$ with respect to $M' := M\setminus\{a_1a_2, b_1b_2\}$. See Figure 8 for an example.

Let c be a configuration such that a and b are connected. Then a simple path between a and b in G_c can be described by an alternating sequence of vertices and switches $v^1s^1v^2s^2 \dots v^{k-1}s^{k-1}v^k$ with $a = v^1$, $b = v^k$ and such that all switches and all vertices are distinct and for $i = 1, \dots, k-1$ it holds that $e_c(s^i) = \{v^i, v^{i+1}\}$. To compute an alternating path between a_1 and b_2 in $H \setminus \{a_2, b_1\}$ we drop all vertices of this sequence and replace each switch s^i for $i = 1, \dots, k-1$ as follows. If v^i is the pivot of s^i we replace s^i by $v_1^i s_1^i s_2^i v_2^{i+1}$, otherwise we replace it by $v_1^i s_2^i s_1^i v_2^{i+1}$. This substitution results in an alternating a_1b_2-path in H with respect to M'.

Conversely assume that we have an alternating a_1b_2-path in $H \setminus \{a_2, b_1\}$ with respect to M'. A first observation is that any alternating path in H that contains a node vertex v_1 or v_2 stemming from a vertex $v \in V \setminus \{a, b\}$ must also contain the corresponding node edge $\{v_1, v_2\}$ and thus both node vertices since $\{v_1, v_2\}$ is in M'. The same holds for switch vertices s_1 and s_2 for all switches $s \in S$. Second, the matching edges that are contained in an alternating path must alternate between switch edges and node edges since by construction of H no two node vertices and no two switch vertices are connected by an edge that is not in M'. Further, a node vertex and a switch vertex that are adjacent in H are incident in G. Hence, the alternating a_1b_2-path in $H \setminus \{a_2, b_1\}$ yields an ab-path in the underlying multigraph of G, i.e., the graph that contains all edges that can possibly result from any configuration of G. As the path in H can contain at most one target connector edge of each switch this path contains at most one edge of each switch and hence can be realized by a configuration. This proves the claim.

It is not hard to see that the reduction can be performed in linear time and that also the resulting path can be translated back in linear time. The claim follows since the existence of an augmenting path can be checked in linear time [15]. □

6 Even Degrees and Eulerian Graphs

In this section we study the problems of finding a Eulerian or a biconnected configuration and several related problems. A graph is Eulerian (that is it admits a cycle that uses each edge exactly once) if and only if it is connected and all vertices have even degrees. As we have seen in Section 4, a connected configuration of a switch graph can be found efficiently, if it exists. It turns out that a configuration for which all vertex degrees are even can be found efficiently, too. We call such a configuration *even* and denote the corresponding problem SwitchEven.

We use the results of Cornuéjols [2] on the general factor problem: Let (W, E) be an undirected graph, and for every $v \in W$ let $D(v)$ be a subset of $\{1, \ldots, |W|\}$. Does there exist a subset $F \subseteq E$, such that in the graph (W, F) every vertex has its degree in $D(v)$? Cornuéjols [2] shows that this problem can be decided in $O(n^4)$ time on any n-vertex graph, as long as the sets $D(v)$ do not contain any gap of length 2. (A set D of integers contains a gap of length 2, if it contains two elements d_1 and d_2, such that $d_2 \geq d_1 + 3$ and such that none of the numbers $d_1 + 1, \ldots, d_2 - 1$ is in D.)

To solve SwitchEven with this result, construct a bipartite auxiliary graph between the set of switches and the set of vertices in the switch graph. Put an edge between any switch s and all targets in T_s. For any switch $s \in S$ set $D(s) = \{1\}$. For any vertex $v \in V$ that is pivot of an even number of switches set $D(v) = \{0, 2, 4, \ldots\}$, and for any vertex $v \in V$ which is pivot of an odd number of switches set $D(v) = \{1, 3, 5, \ldots\}$. Note that none of these sets contains a gap of length 2. It can be seen that the auxiliary graph has a factor obeying the degree constraints if and only if the graph G has a configuration in which all vertex degrees are even.

While this settles the membership of SwitchEven in $\mathcal{P}$, the algorithm is not very efficient, as it takes time $O((n + m)^4)$. We therefore also provide a more elementary algorithm that is faster than the one described above.

Theorem 8 *For an undirected switch graph $G = (V, S)$,* SwitchEven *can be decided in $O(n \cdot (\bar{m} + n))$ time.*

Proof Let $G = (V, S)$ be a switch graph and let c be a configuration. We define a helper switch graph $H(c) = (V, S_c)$ that contains one switch for each switch of G in the following manner. For every $s \in S$ we define a corresponding switch $(c(s), T_s \setminus \{c(s)\})$ that has the current target $c(s)$ of s as pivot and as targets all alternative targets of s. For ease of use we identify corresponding switches in the graphs G and $H(c)$.

Assume that two vertices a and b that are odd in G_c can be connected by a path in $H(c)$ with a configuration h. Let $s_1, \dots, s_k$ be the set of switches in this path. We define a new configuration c' of G as follows: We set $c'(s) := h(s)$ if $s \in \{s_1, \dots, s_k\}$ and $c'(s) := c(s)$ otherwise. Now the even vertices of $G_{c'}$ are exactly the even vertices of G_c plus a and b. By the definition of c' the degree of a vertex changes from G_c to $G_{c'}$ by 1 for each edge on the path between a and b in $H(c)_h$. Since all interior vertices of the path have an even number of incident edges only the parity of a and b changes. This suggests a very simple strategy for finding even configurations: Start with any configuration c of G and as long as there exists an odd vertex a find a path in $H(c)$ that connects a to an odd vertex b and change the configuration accordingly.

It remains to show that if the strategy does not succeed in finding an even configuration then there is none. We prove that if there exists an even configuration c^* of G, then for any non-even configuration c and any odd vertex a of G_c it is possible to switch a path in $H(c)$ that connects a to another odd vertex b. Consider the graph $H' = (V, E_{H'})$ with $E_{H'} = \{\{c(s), c^*(s)\} \mid s \in S \text{ with } c(s) \neq c^*(s)\}$. Note that this graph can by definition be obtained as a subgraph of $H(c)$ with a suitable configuration h. Each edge in H' represents a change of the degree of its incident vertices by 1 when changing c to c^*. Therefore, even (odd) vertices of G_c have even (odd) degree in H'. Hence a must have odd degree in H'. Since the connected component of H' that contains a must have an even number of odd vertices, there is an odd vertex b in this component and hence we can switch a path between a and b in $H(c)$ as claimed. The running time follows from at most n applications of the algorithm from Theorem 7. □

As we have seen, we can efficiently check whether a given switch graph admits a connected configuration and whether it admits an even configuration. A Eulerian configuration is one that satisfies both properties simultaneously. Interestingly, this combined problem is much more difficult than the two individual problems, and in fact NP-hard as we show in the next theorem. Moreover, higher degrees of connectivity, such as finding a biconnected configuration or a strongly connected configuration in the case of directed switch graphs is NP-hard as well.

Theorem 9 *For binary undirected switch graphs it is NP-hard to decide if there is a Eulerian or a biconnected configuration. For binary forward directed switch graphs it is NP-hard to decide if there is a Eulerian or a strongly connected configuration.*

Proof We reduce from DirectedHamiltonianCycle, which is known to be NP-hard for directed graphs with out-degree bounded by 2 [12]. Obviously out-degree 1 is a necessary lower bound for the existence of a directed Hamiltonian cycle.

Let $G = (V, E)$ be a directed graph with out-degrees 1 and 2. We define a switch graph $H = (V, S)$ as follows. For each vertex $v \in V$ we add a switch $s_v = (v, N(v))$

where $N(v) = \{u \in V \mid (v, u) \in E\}$. Let further $\overrightarrow{H}$ denote the directed switch graph obtained from H by replacing each switch by a corresponding forward switch with the same pivot and the same target set. Obviously a configuration of H can be interpreted as a configuration of $\overrightarrow{H}$ and vice versa. Now, since for every configuration c the multi-graphs H_c and $\overrightarrow{H}_c$ have n vertices and n edges, the following properties are equivalent:

(i) G has a directed Hamiltonian cycle.
(ii) $\overrightarrow{H}$ has a directed Eulerian configuration.
(iii) $\overrightarrow{H}$ has a strongly connected configuration.
(iv) H has a biconnected configuration.
(v) H has a Eulerian configuration.

The claim follows since the reduction can be performed in linear time. □

7 Acyclic and Almost Acyclic Graphs

This section deals with forward directed switch graphs (as defined in Section 2): We check in polynomial time whether a forward directed switch graph has a DAG configuration. We also show that finding a configuration with the minimum number of directed cycles is NP-hard for binary forward directed switch graphs. By contrast, we show that for general binary directed switch graphs, which may contain forward and reverse switches, even testing the existence of a DAG configuration is NP-hard.

Theorem 10 *For a forward directed switch graph, it can be decided in $O(n + \bar{m})$ time if it has an acyclic configuration. If an acyclic configuration exists, it can be found within the same time complexity.*

Proof Let $G = (V, S)$ be a forward directed switch graph, and observe the following. First: The out-degree of every vertex in G_c is independent of the chosen configuration c. Second: If all vertices in a digraph have out-degree at least 1, then the graph contains a directed cycle. Third: If G contains a sink v (that is, a vertex v with out-degree 0), then it is safe to configure all switches s with $v \in T_s$ towards this sink when looking for a configuration minimizing the number of directed cycles. These three observations suggest a simple procedure: As long as the graph contains a sink v, we first set $c(s) := v$ for all switches s with $v \in T_s$, and then remove v together with all these switches. The procedure either stops with an empty graph (and an acyclic configuration), or with a non-empty subgraph of G in which all vertices have out-degree at least 1 (in which case there is no acyclic configuration). The algorithm can easily be implemented to run in linear time. □

In contrast, finding an acyclic configuration in general directed switch graphs and minimizing the number of cycles in forward directed switch graphs is hard:

Theorem 11 *For binary directed switch graphs, it is NP-hard to decide whether an acyclic configuration exists* (SwitchDirectedAcyclic).

Proof The proof is by reduction from 3Sat. Let φ be an instance of 3Sat with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. We construct a switch graph G_φ as follows: We start with two vertices z and w and the arc (w, z). For each variable x_i we create two corresponding vertices $x_i, \bar{x}_i$ and a reverse switch $s_i = (\{x_i, \bar{x}_i\}, w)$. For each clause C_i we add a vertex v_i and the arc (z, v_i). Let x_u, x_v, x_w be the variables occurring in clause C_i. We set $\ell_u := x_u$ if x_u occurs negated in C_i and $\ell_u := \bar{x}_u$ otherwise. We define ℓ_v, ℓ_w analogously. We then add a *clause switch* $s_i^C = (v_i, \{\ell_u, \ell_v, \ell_w\})$. See Figure 9 for an example.

A satisfying truth assignment for φ yields an acyclic configuration c of G_φ: For each variable x_i we set $c(s_i) := x_i$ if x_i is assigned the value true and $c(s_i) := \bar{x}_i$ otherwise. Since each clause of φ is satisfied in this configuration at least one target of every clause switch has out-degree 0. Hence every clause switch can easily be configured to avoid all cycles.

Moreover, an acyclic configuration c of G_φ yields a satisfying truth assignment for φ: We set variable x_i to true if $c(s_i) = x_i$ and to false otherwise. As the configuration is acyclic every clause switch must have a sink as target, and this sink represents a satisfied literal in the corresponding clause.

Note that although the clause switches have fan-out 3, the result also holds for binary switch graphs. To see this, we simply replace each switch $s = (p, \{x, y, z\})$ by two switches $s_1 = (p, \{x, p'\})$ and $s_2 = (p', \{y, z\})$, where p' is a new vertex. This replacement does not affect the number of cycles with respect to any configuration since p' has either degree 1 (and thus is not contained in any cycle) if s_1 picks x, or degree 2 if s_1 picks p'. In the latter case p' belongs to exactly the same cycles as p. □

Theorem 12 *For a binary forward directed switch graphs G and an integer $k > 0$, it is NP-hard to decide if there is a configuration with at most k cycles* (SwitchMinimumDirectedCycles).

Proof We show how to simulate binary reverse switches with usual binary forward switches at the cost of one cycle per reverse switch. Let G' be an instance of SwitchDirectedAcyclic with k binary reverse switches. We construct a directed switch graph G by replacing each reverse switch $s = (\{x, \bar{x}\}, w)$ by the following construction. We add four vertices $x_{\text{out}}, \bar{x}_{\text{out}}, x_{\text{in}}, \bar{x}_{\text{in}}$ along with the arcs $(x_{\text{in}}, \bar{x})$, $(\bar{x}_{\text{in}}, x)$, (x_{out}, x), $(\bar{x}_{\text{out}}, \bar{x})$, (x_{in}, w), $(\bar{x}_{\text{in}}, w)$ and the two forward switches $s_x = (x, \{x_{\text{in}}, x_{\text{out}}\})$, $s_{\bar{x}} = (\bar{x}, \{\bar{x}_{\text{out}}, \bar{x}_{\text{in}}\})$; see Figure 9.

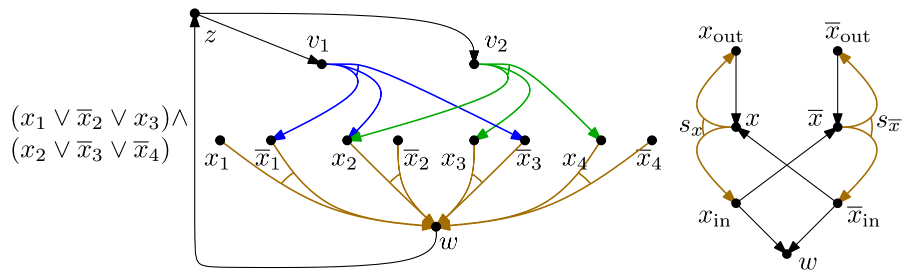


Fig. 9: Reduction of 3Sat to SwitchDirectedAcyclic with reverse switches (left). Replacement of reverse switches for reduction of 3Sat to SwitchMinimumDirectedCycles (right)

As each replacement creates at least one cycle every configuration of G has at least k cycles. Each replacement has four distinct configurations. Two of them directly correspond to a configuration of the original reverse switch, namely the ones where one of the vertices $x, \bar{x}$ is connected to its in- and the other one to its out-vertex. We say that a configuration of G is *good* for the replacement in this case. There is a bijection between the acyclic configurations of G' and the configurations of G with k cycles that are good for each replacement.

Let c be a configuration of G with k cycles. We can modify c such that it is good for each replacement without increasing the number of cycles: The case $c(s_x) = x_{\text{out}}, c(s_{\bar{x}}) = \bar{x}_{\text{out}}$ can be excluded, as it would induce two cycles. In case $c(s_x) = x_{\text{in}}, c(s_{\bar{x}}) = \bar{x}_{\text{in}}$ we can change $c(s_x) := x_{\text{out}}$ without increasing the number of cycles thus making c good for the replacement. This operation does not increase the number of cycles, as we introduce at most one new cycle, namely xx_{out} but at the same time remove at least the cycle $xx_{\text{in}}\bar{x}\bar{x}_{\text{in}}$. Hence G admits a configuration with at most k cycles if and only if G' has an acyclic configuration. Since G can be constructed in linear time from G', the problem to decide whether such a configuration exists is NP-hard. □

8 Conclusion

In this paper we have studied the complexity of fundamental problems on switch graphs. While finding a configuration of a switch graph such that the resulting graph satisfies a certain property $\mathcal{P}$ is NP-hard for many properties, we gave efficient algorithms for certain connectivity problems. In particular, we have shown how to test efficiently, whether all vertices of a switch graph can be connected and how to check whether two given vertices can be connected. We leave open the question whether it is possible to check in polynomial time if three given vertices can be connected simultaneously and, more generally, whether the problem SwitchConnect-T is FPT with respect to $|T|$.

Acknowledgments We thank Alexander Wolff for pointing us to switch graph problems. We thank the anonymous referees for helpful comments.

References

1. M. Cook. Still Life Theory. In C. Moore and D. Griffeath, editors, *New Constructions in Cellular Automata*, volume 226, pages 93–118. Oxford University Press, 2003.
2. G. Cornuéjols. General factors of graphs. *Journal of Combinatorial Theory, Series B*, 45:185–198, 1988.
3. M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *Proc. 16th International Computing and Combinatorics Conference (COCOON'2010)*, volume 6196 of *LNCS*, pages 216–225. Springer, 2010.
4. J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications*, pages 69–87, Calgary, 1969.
5. M. R. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.
6. J. Groote and B. Ploeger. Switching graphs. *International Journal of Foundations of Computer Science*, 20(5):869–886, 2009.
7. P. Hall. On representatives of subsets. *Jour. London Math. Soc.*, 10:26–30, 1935.

8. U. Huckenbeck. On paths in networks with valves. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, volume 665 of *LNCS*, pages 90–99, 1993.
9. U. Huckenbeck. On valve adjustments that interrupt all s-t-paths in a digraph. *Journal of Automata, Languages and Combinatorics*, 2(1):19–45, 1997.
10. B. Korte and J. Vygen. *Combinatorial Optimization, Theory and Algorithms.* Springer New York, Berlin, fourth edition edition, 2008.
11. C. Meinel. Switching graphs and their complexity. In *Proceedings of the 14th Conference on Mathematical Foundations of Computer Science (MFCS'1989)*, volume 379 of *LNCS*, pages 350–359. Springer, 1989.
12. J. Plesńik. The NP-completeness of the Hamiltonian Cycle Problem in planar digraphs with degree bound two. *Information Processing Letters*, 8(4):199–201, 1979.
13. K. Reinhardt. The simple reachability problem in switch graphs. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'2009)*, volume 5404 of *LNCS*, pages 461–472. Springer, 2009.
14. R. Sharan, J. Gramm, Z. Yakhini, and A. Ben-Dor. Multiplexing schemes for generic SNP genotyping assays. *Journal of Comp. Biology*, 15:514–533, 2005.
15. R. E. Tarjan. *Data structures and network algorithms.* SIAM, Philadelphia, 1983.