Technische Universität Berlin
Fakultät IV: Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Algorithmik und Komplexitätstheorie (AKT)

Non-Binary Opinion Dynamics in Social Networks

Bachelorarbeit

von Lilian Jacobs

Matrnr. 362943

zur Erlangung des Grades „Bachelor of Science“ (B.Sc.)
im Studiengang Informatik

Erstgutachter:	Prof. Dr. Rolf Niedermeier
Zweitgutachter:	Prof. Dr. Markus Brill
Betreuer:	Dr. Robert Bredereck

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittle angefertigt habe.

Die selbstständige und eigenständige Anfertigung versichert an Eides statt:

Berlin, den _____

Unterschrift

Abstract

Within the realm of graph theory and algorithmics, we investigate opinion diffusion on influence networks. Agents of an influence network update their opinions such that they are in agreement with the majority of their neighborhood. Recent research on binary influence networks did reveal that finding an update sequence maximizing the occurrence of an opinion in an influence network can be done in polynomial time. We study the computational complexity of this problem for non-binary influence networks. In this thesis, we prove that finding such an opinion maximizing sequence for three opinions is NP-complete. Further, this is shown for the case of influence networks with a maximum degree of at least seven as well as planar influence networks.

Kurzzusammenfassung

Im Rahmen der Graphentheorie und Algorithmik untersuchen wir die Meinungsverbreitung auf Einflussnetzwerken. Ein Agent eines Einflussnetzwerkes aktualisiert seine Meinung, sodass er in Übereinstimmung mit der Mehrheit seiner Nachbarschafft steht. Kürzliche Untersuchungen auf binärer Einflussnetzwerken ergaben, dass das Finden einer Updatesequenze, welche das Vorkommen eine Meinung in einem Einflussnetzwerkes maximiert, in polynomieller Zeit erreicht werden kann. Wir untersuchen die Komplexität dieses Problems für nicht-binäre Einflussnetzwerke. In dieser Arbeit beweisen wir, dass das Finden einer meinungsmaximierenden Sequenz für drei Meinungen NP vollständig ist. Ebenso zeigen wir dies für Einflussnetzwerke mit einem maximalen Kantengrad von sieben sowie für planare Einflussnetzwerke.

Contents

1 Introduction

Today, the importance of communication increases rapidly over time. With the rise of social networking over the last few decades exchanging information, ideas or opinions over large distances and short time periods was never that easy. With big companies like Twitter and Facebook we have created massive digital social networks. However, social networks are not limited to the digital infrastructure. Analog social networks do exist, where humans living in a city can be interpreted as vertices and there is an edge between two people when they are neighbors, friends or colleagues. For social networks we are interested in the opinion on certain topics of the members of the network. When making a decision, people tend to consider the opinion of like-minded individuals (see e.g. the survey of Grandi [Gra17]). We investigate a model for opinion diffusion, where agents may update their opinion such that the opinion is in agreement with the majority of their neighborhood. This model was first investigated for a binary set of opinions by Goles and Olivos [GO80]. In this thesis, we investigate opinion diffusion for non-binary sets of opinions. An example for a binary set of opinions on a social network is, whether someone likes the operation system Linux. In comparison, for a set of opinions of size three, one may choose from a list of operation systems including the the choices of Linux, iOS or Windows. For someone interested in a specific opinion, the maximum number of agents with a specific opinion in a network may be important. It is known that one can compute the best-case outcome for the presented model of opinion diffusion in polynomial time for a binary set of opinions [BE17]. In this thesis, we investigate the computational complexity for a set of opinions of size three.

1.1 Related Work

For opinion diffusion there is a lot of literature on different aspects and for different fields of science (see [Gra17] for a recent survey). The work of Goles and Olivos [GO80] shows that a synchronous update sequence converges to a stable state with period of two for a binary set of opinions. Yildiz et al. [Yil+13] describes *stubborn agents* influencing there neighbors. These are agents that are ignorant about the opinions of their neighbors and do never change their opinion. The concept of stubborn agents is closely related to *permanently stable* agents, which are described in Section 2.2.1. Also, the work of Chierichetti, Kleinberg, and Oren [CKO13] is closely related to our thesis. They describe *discrete preference games* consisting of agents in a finite metric space, where these agents need to decide on their opinion. The edges between these agents are weighted, thus our setup can be seen as a special case of discrete preference games. In a somewhat related paper, opinion diffusion on preference profiles is described by Faliszewski et al. [Fal+18]. In a paper by Katona, Zubcsek, and Sarvary [KZS11], the influential power of agents in a social network is described. The influence of an agent in a social network is depending on the degree of the corresponding vertex as well as the size of the surrounding cluster. Opinion manipulation in weighted networks for non-discrete opinions is described in the paper of Silva [Sil16]. The concept is somewhat similar to the non-discrete majority update rule described in Section 2.1.1. The opinion diffusion of a binary set of opinions

with constrains is described in the paper of Botan, Grandi, and Perrussel [BGP]. The main problem in this paper is to minimize the amount of exchanged information in the influence network.

1.2 Results of the Thesis

In this thesis, we prove that finding an update sequence maximizing the number of vertices with a specific opinion is NP-complete for three opinions. We do this by reducing 3Sat to this problem. Further, we show that this problem remains NP-complete for planar graphs and graphs such that each vertex has at most seven neighbors. On the positive side, we prove that finding such an update sequence can be done in polynomial time for paths and cycles.

2 Preliminaries

In this chapter we formally introduce the main concepts and definitions used in the thesis: influence networks, update rules and update sequences.

2.1 Model and Notation

An (*undirected*) *influence network*—InfNet for short—is a pair $(G, \circ)$ with $G = (V, E)$ being an (*undirected*) graph such that $|V| = n$ and $|E| = m$. Also, $\circ : V \to O$ is an *opinion function* describing the initial opinion of each vertex for a set of opinions O. If $|O| = 2$, then we use the set of opinions $O := \{0, 2\}$. We say a vertex v is *black* if $\circ(v) = 2$ and *white* if $\circ(v) = 0$. If $|O| = 3$, then we use the set of opinions $O := \{0, 1, 2\}$. We say a vertex v is *black* if $\circ(v) = 2$, *gray* if $\circ(v) = 1$ and *white* if $\circ(v) = 0$.
We denote the *neighborhood* of a vertex $v \in V$ by $N(v) := \{u \in V \mid \{u, v\} \in E\}$. The *degree* of v—the number of neighbors—is denoted with $\deg(v)$, which can also be denoted by $|N(v)|$. The maximum degree in the graph G is denoted by $\triangle(G) := \max_{v \in V}(\deg(v))$. We use $V_{i,\circ} = \{v \in V \mid \circ(v) = i\}$ for the set of vertices mapped to opinion $i \in O$ for the opinion function $\circ$. Note, that each vertex has exactly one opinion, thus for $|O| = 3$ we have $V_{0,\circ} \dot\cup V_{1,\circ} \dot\cup V_{2,\circ} = V$.

2.1.1 Update Rules

There are different ways to consider *updating* the opinion of vertices in an influence network. We declare two update rules.

For the **discrete majority update** rule a vertex updates its opinion to the opinion shared by the majority of the neighborhood. The vertex keeps the current opinion if there is no majority opinion. For this update rule we introduce the function $\mathrm{maj}(v, \circ)$:

$$\mathrm{maj}(v, \circ) = \begin{cases} q, & \text{if } |\{w \in N(v) \mid \circ(w) = q\}| > \frac{|N(v)|}{2} \\ \circ(v), & \text{otherwise.} \end{cases}$$

For the **non-discrete majority update** rule we compute the mean value of the opinions of the neighborhood of v. The mean value is

$$\frac{\sum_{w \in N(v)} \circ(w)}{|N(v)|}.$$

We choose the opinion closest to this value. However, when there are two closest opinions, then we choose the opinion closest to the opinion of v. Afterwards, we update v to this opinion. An example for both update rules can be found in Figure 1.

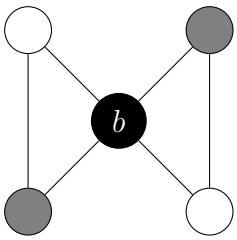


Figure 1: An example such that the discrete majority update rule and the non-discrete majority update rule behave different for the vertex b. For the discrete majority update rule the vertex b remains black, because there is no opinion shared by the majority of the neighborhood. For the non-discrete majority update rule the vertex b updates to gray. The mean value of the opinions of the neighbors of b is 0.5. The opinion white (0) and gray (1) have the same distance to the value, but gray is closer to black (2). For that reason b updates to gray.

2.1.2 Update Process

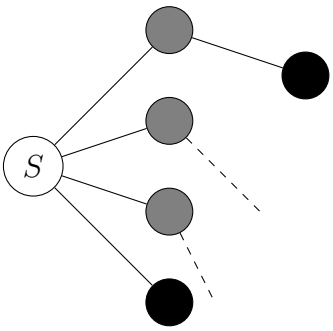
During an update process a set of vertices gets updated according to a chosen update rule. The opinion $\circ(v)$ of a vertex v may change during an *update*. We refer to the ordering of updating vertices in an influence network $(G, \circ)$ as *update sequence*. An update sequence σ is a sequence of subsets of V where the i-th subset corresponds to the set of vertices getting updated at step i. We denote by $\circ[G, \sigma, z]$ the opinion function resulting from $\circ$ after z update steps following the sequence σ. Further, we denote by $\circ[G, \sigma]$ the opinion function after $|\sigma|$ steps. We declare three properties for update sequences.

An update sequence is a **synchronous update sequence**, when every vertex updates simultaneously in each update step. Therefore, we denote a synchronous update sequence σ with $\sigma = (V, ..., V)$.

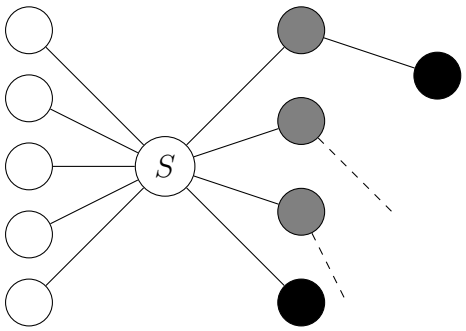
An update sequences is an **asynchronous update sequence**, when in each update step only one vertex updates. Each element of an asynchronous update sequence σ is a singleton of an element of V.

An update sequence is a **balanced asynchronous update sequence**, when the sequence is an asynchronous update sequence and every vertex v can only update, when all other vertices of the influence network did update at least as often as v. In order to create such an update sequence, we create a list, which contains all vertices in an arbitrary order. This list dictates the order of the update sequence. We iterate through the list and append a singleton of the current vertex to the update sequence. After one iteration, we create a permutation of this list and iterate through this list all over again. We allow a balanced asynchronous update sequence to end at any point of this list. Hence, an update sequence σ does not need to fulfill the property that $|\sigma| \bmod |V| = 0$.

For a given InfNet $(\mathrm{G}, \circ)$ we say the vertex v is stable when $\circ(v) = \mathrm{maj}(v, \circ)$. The graph G is stable if and only if all $v \in V$ are stable. Further, we say that the update sequence σ is stable if and only if for all $v \in V$ the property $\circ[G, \sigma](v) = \mathrm{maj}(v, \circ[G, \sigma])$ holds.



(a) An example of a permanently stable vertex S in an influence network. The vertex S does not update to gray, because S shares an edge with at least five white vertices, which are hidden in the illustration.



(b) The same permanently stable vertex S such that the white vertices in his neighborhood are shown in the illustration.

Figure 2: A permanently stable vertex S with hidden majority neighborhood on the left side and with visible majority neighborhood on the right side. We use the symbol S in illustrations to reduce the number of vertices and increase readability.

2.2 Graph Notation

In this section, we define some graph notations. We introduce permanently stable vertices to improve *readability* and planar graphs as an important graph property.

2.2.1 Permanent Stable

We introduce a type of vertices whose opinion does not change for any update sequence—*permanently stable* vertices. A vertex is permanently stable when there is no update sequence σ such that $\circ(v) \neq \circ[G, \sigma, z](v)$ for any $z \in [0, |\sigma|]$. An arbitrary vertex v in an influence network can be declared permanently stable when the majority of his neighbors have the same opinion and these vertices are only connected to v or each other.
In our illustrations, we label permanently stable vertices by S. For that reason, we can hide part of the neighborhood of a permanently stable vertices to highlight other aspects of an illustration.
For the discrete majority update rule an arbitrary vertex v can be easily made permanently stable by adding a set of $|N(v)| + 1$ vertices with opinion $\circ(v)$ to the graph and connect each vertex of the cluster to v. The size of this cluster exceeds the initial size of $N(v)$, thus $\text{maj}(v, \circ[G, \sigma, z]) = \circ(v)$ for every sequence σ and $z \in [0, |\sigma|]$. Also, the majority of $v' \in N(v)$ fulfills the property $\text{maj}(v', \circ[G, \sigma, z]) = \circ(v') = \circ(v)$, because these added vertices are stable by definition. An illustration describing the usage of the label S is shown in Figure 2.

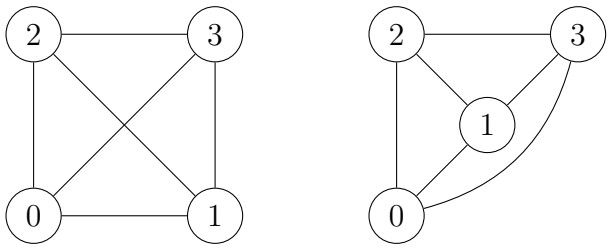


Figure 3: On the left side a K_4 is illustrated such that the graph is not planar embedded. On the right side the same K_4 is illustrated such that the graph is planar embedded.

2.2.2 Planar Graph

We introduce planarity of graphs which is an important graph property. A planar graph is a graph which can be embedded in the plane such that there are no edges crossing. An example of a planar embedded graph can be found in Figure 3.

2.3 Problems

In the following we declare the definitions of the problems used in this paper. Particular, we will introduce 3Sat (Garey and Johnson [GJ99, LO2]) and Planar 3Sat (Lichtenstein [Lic77]).

2.3.1 3SAT

The problem 3Sat is a variant of the boolean satisfiability problem SAT, where the instances contain only clauses—disjunctions of literals—with exactly three literals each. A literal is either a variable (also referred as *positive literal*) or the negation of a variable (also referred as *negative literal*).

3Sat

Input: A set U of variables and a collection C of clauses over U such that every clause contains three literals.

Question: Is there a boolean assignment $\delta : U \to \{\top, \bot\}$ for every $x \in U$ such that every clause $c \in C$ evaluates to $\top$.

Without loss of generality, we assume that our 3Sat instances do not contain a variable appearing either only positively or only negatively. Setting such a variable to true (or false respectively) will always be part of a satisfying assignment if such an assignment exists. Therefore, we simply remove these variables and clauses containing those variables.

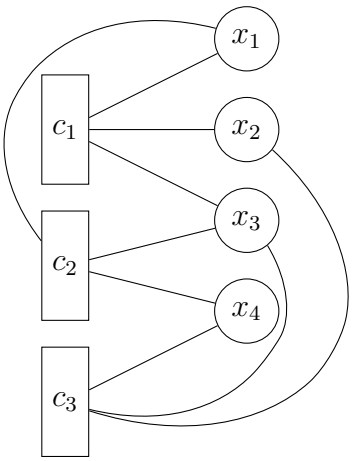


Figure 4: The formula graph $G(\phi)$ for a Planar 3Sat instance with $C \coloneqq \{(x_1, x_2, x_3), (x_1, x_3, x_4), (x_2, x_3, x_4)\}$ and $U \coloneqq \{x_1, x_2, x_3, x_4\}$.

2.3.2 Planar 3Sat

The problem of Planar 3Sat is a variation of 3SAT such that the corresponding formula graph is planar.

Planar 3Sat
Input: A set U of variables, a collection C of clauses over U such that every clause contains three literals and a planar formula graph $G(\phi)$.
Question: Is there a boolean assignment $\delta : U \to \{\top, \bot\}$ for every $x \in U$ such that every clause $c \in C$ evaluates to $\top$.

The *formula graph* $G(\phi)$ for some boolean formula ϕ is defined as following:

$$G(\phi) \coloneqq (U \cup C, \{\{u, c\} \in U \times C \mid u \in c\})$$

The visual representation of a formula graph can be found in Figure 4.

3 Complexity of Finding an Optimistic Update Sequence

With asynchronous update sequences there is a huge variation in possible outcomes. For various reasons, one might be interested in maximizing the number of vertices for a given opinion. In the work of Bredereck and Elkind [BE17] an update sequence maximizing the number of black vertices for a set of opinions of size two is defined as a *optimistic update sequence*. We generalize the declaration of an optimistic update sequence. Instead, we say that an update sequence maximizing the number of vertices for a given opinion q is an optimistic update sequence. Note, that we use the opinion $q = 2$ (black) as preferred opinion. For a given InfNet $(G, \circ)$ and update rule an optimistic update sequence σ has the following properties:

- σ is asynchronous, thus each update step is a singleton,
- $\circ[G, \sigma]$ is stable, thus $\text{maj}(v, \circ[G, \sigma]) = \circ[G, \sigma](v)$ for each vertex v, and
- there is no stable update sequence σ' such that $\circ[G, \sigma']$ contains more vertices with the preferred opinion than $\circ[G, \sigma]$.

We formally define the decision problem Optimistic Update Sequence:

Optimistic Update Sequence
Input: An undirected *InfNet* $(G, \circ)$ with $G = (V, E)$, a set of opinions O, an opinion function $\circ : V \to O$, a preferred opinion $q \in O$ and a number k.
Question: Is there an asynchronous and stable update sequence σ such that $|\{v \in V \mid \circ[G, \sigma](v) = q\}| \geq k$?

Computing an optimistic update sequence σ for an opinion function $\circ : V(G) \to O$ such that $|O| = 2$ can be achieved in $\mathcal{O}(n \cdot m)$ time as shown by Bredereck and Elkind [BE17]. For the preferred opinion black and non-preferred opinion white the algorithm for $|O| = 2$ has two phases:

1. While there is an unstable white vertex v, append it to σ and update $\circ(v)$ to black.
2. While there is an unstable black vertex v, append it to σ and update $\circ(v)$ to white.

For all influence networks $(G, \circ)$ with $|O| = 2$ the property $\circ[G, \sigma] = \circ[G, \sigma^*]$ holds for every other update sequence σ^* maximizing the number of black vertices. That is, every optimistic update sequence produces the same outcome. Unfortunately, this property does not hold for $|O| = 3$. An example for this is shown in Figure 5. Further, we observed that there are instances as shown in Figure 6, where changing the opinion of a vertex in order to stabilize a black vertex may lead to removing other black vertices. Hence, we must decide whether to update a non-preferred vertex which can lead to updating black vertices later on. This indicates that Optimistic Update Sequence is computational complex in case of three opinions. Indeed, we show the containment in NP in Section 3.1 and NP-hardness in Section 3.2.

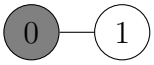


Figure 5: A graph example for which there are two update sequences $\sigma = [0]$ and $\sigma' = [1]$ such that $\circ[G, \sigma] \neq \circ[G, \sigma']$ but $|\{v \in V \mid \circ[G, \sigma](v) = 2\}| = |\{v \in V \mid \circ[G, \sigma'](v) = 2\}|$.

3.1 NP-membership

The problem Optimistic Update Sequence is in NP when for an update sequence σ we can determine in *polynomial time* whether the amount of black vertices is equal to or exceeds the value of a given number k. We compute $\circ[G, \sigma]$, which can be done in $\mathcal{O}(|\sigma| \cdot m)$ time, because in each step of σ we compute the majority opinion of the vertex updating in this step. In order to compute the majority opinion, we need to visit at most m edges. Afterwards, we verify that

$$|\{v \in V \mid \circ[G, \sigma](v) = q\}| \geq k.$$

We calculate the number of black vertices in $\mathcal{O}(n)$ time by iterating through all vertices and incrementing a counter for each black vertex for $\circ[G, \sigma]$.
When the number of black vertices is equal to k or exceeds k, then we have a YES-instance. Otherwise, we have a NO-instance. Each step of the verification runs in polynomial time, thus the overall running time is polynomial. Hence, Optimistic Update Sequence is in NP.

3.2 NP-hardness

In order to prove NP-hardness we show that

$$\text{3Sat} \preceq_p \text{Optimistic Update Sequence}.$$

For our polynomial-time reduction from 3Sat to Optimistic Update Sequence we make use of two kinds of gadgets: *clause gadgets* and *variable gadgets*. The *clause gadgets* represent the clauses of a 3Sat instance while the *variable gadgets* represent the variables of a 3Sat instance. In the following we will define both gadgets, explain the connection between the two gadget structures and describe the constructed Optimistic Update Sequence instance.

3.2.1 Variable Gadget

The idea of the variable gadget is to construct a gadget with a black vertex b_{x_i} for each variable x_i such that this black vertex remains black when either all x_i or all $\bar{x}_i$ are assigned to true.

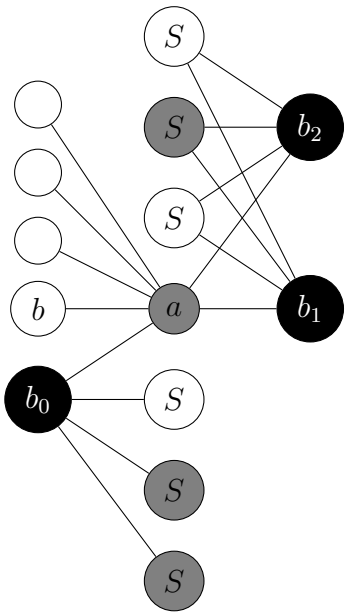


Figure 6: An example for Optimistic Update Sequence where we need to decide whether to update one black vertex to gray or two black vertices to white. When we update the vertex a first, then the vertex b_0 is stable but the vertices b_1 and b_2 are unstable and will turn white. However, when b gets updated first, then the vertices b_1 and b_2 are stable but the vertex b_0 is unstable and will turn gray.

Gadget Structure

For each variable $x_i \in U$, we create two white vertices called $x_i^\top$ and $x_i^\bot$. The vertex $x_i^\top$ represents the collection of all positive literals x_i. The vertex $x_i^\bot$ represents the collection of all negative literals $\bar{x}_i$. We connect the vertices $x_i^\top$ and $x_i^\bot$ to a common black vertex b_{x_i} as well as a common, permanently stable gray vertex. An example for a variable gadget is shown in Figure 7.

Connection

For each positive literal x_i we connect the vertex $x_i^\top$ to one gray vertex from each clause gadget corresponding to a clause that contains x_i. Similarly, for each negative literal $\bar{x}_i$ we connect the vertex $x_i^\bot$ to one gray vertex from each clause gadget corresponding to a clause that contains $\bar{x}_i$. The vertices $x_i^\top$ and $x_i^\bot$ of this gadget need to have an odd number of neighbors. Initially, the majority of the neighborhood of $x_i^\top$ and $x_i^\bot$ needs to be gray. However, when one gray neighbor updates to white, then the majority of the neighborhood needs to be white. Thus, we add white vertices to the influence network and connect them to $x_i^\top$. The number of these added white vertices is equal to the number of gray vertices from the clause gadgets having an edge to $x_i^\top$ subtracted by one. As a result, the initial neighborhood of $x_i^\top$ is a gray majority neighborhood.

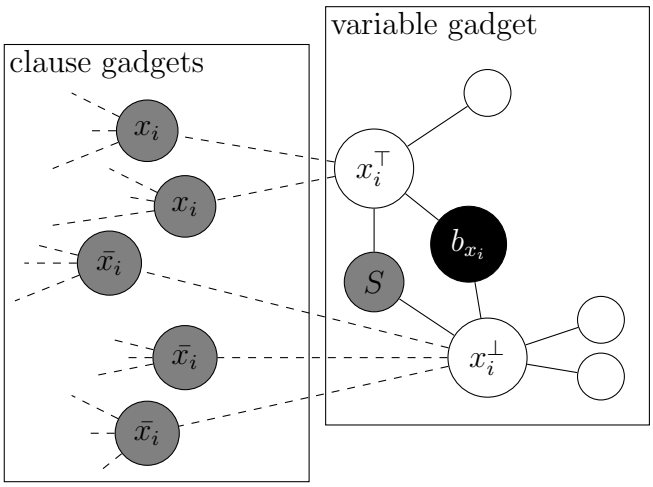


Figure 7: A variable gadget for x_i where each positive literal is connected to $x_i^\top$ and each negative literal to $x_i^\perp$.

Accordingly, we add white vertices to $x_i^\perp$ in relation to the number of negative literals $\bar{x}_i$.

Properties

The vertices $x_i^\top$ and $x_i^\perp$ are the sole neighbors of the black vertex b_{x_i}. In order for b_{x_i} to remain black for any stable update sequence at least one of both neighbors needs to change its opinion. Hence, one of both neighbors must change opinion to gray while the other remains white.

3.2.2 Clause Gadget

The idea of the clause gadget is to construct a gadget such that all vertices of the gadget update to white when at least one literal of the clause is assigned to true. We refer to the clause gadget for the clause $c \in C$ as CG(c).

Gadget Structure

For every clause $c \in C$, we create a gray vertex for each literal; i.e., for the literals x_i, x_j and x_k and the clause c we label the corresponding gray vertices with x_i^c, x_j^c and x_k^c. We connect these vertices to each other with edges. Additionally, for each size-two combination chosen from x_i^c, x_j^c and x_k^c we create a permanently stable white vertex and connect the newly created vertex with the two chosen gray vertices.

Connection

After we have created all clause gadgets, we create the black vertex b_0. For each clause gadget we choose one gray vertex and connect it to b_0. For each of these gray vertices, we create another permanently stable white vertex and connect this vertex to the gray

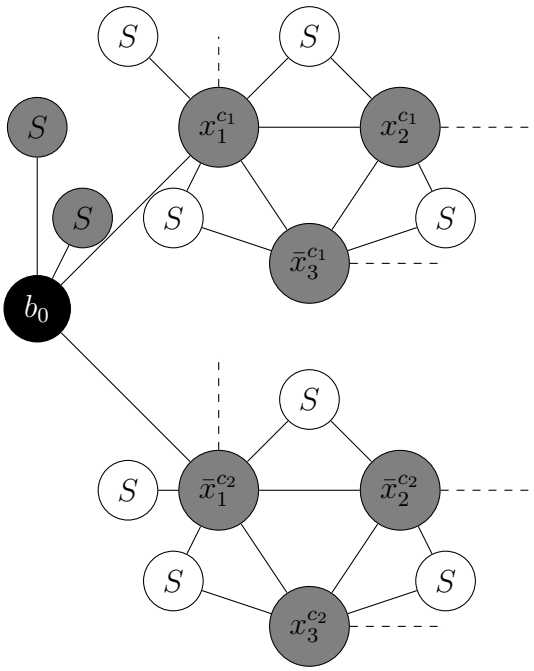


Figure 8: An example for the structure of clause gadgets and the connection to the black vertex b_0 for the example $C \coloneqq \{(x_1, x_2, \bar{x_3}), (\bar{x_1}, \bar{x_2}, x_3)\}$ and $U \coloneqq \{x_1, x_2, x_3\}$.

vertex. Further, for each vertex connected to the vertex b_0 we add another permanently stable gray vertex and connect it to b_0. For each positive literal x_i from the clause c, we connect the vertex x_i^c to $x_i^\top$ from the variable gadget. For each negative literal $\bar{x}_i$ from the clause c, we connect the vertex $\bar{x}_i^c$ to $x_i^\bot$ from the variable gadget. The construction of the clause gadgets is shown in Figure 8.

Properties

The black vertex b_0 has a neighborhood of size $2|C|$ only containing gray vertices. In order to remain black, we need to balance the neighborhood such that

$$|\{v \in N(b_0) \mid \circ(v) = 1\}| = |C| = |\{v \in N(b_0) \mid \circ(v) = 0\}|.$$

The vertex b_0 is connected to $|C|$ permanently stable gray vertices. Therefore, in order for b_0 to remain black and become stable we need to update the remaining vertices to white. These vertices are the gray vertices of the clause gadgets. The vertex b_0 remains black when all these vertices update to white and remain white for the final outcome of the update sequence.

For each literal x_i all gray vertices x_i^c for $c \in C$ are unstable with a white majority neighborhood. However, the vertex $x_i^\top$ is unstable with a gray majority neighborhood. When we update any x_i^c, then $x_i^\top$ becomes stable. Instead, when we update $x_i^\top$ first, then all x_i^c become stable. Further, when a particular x_i^c updates before $x_i^\top$, then the

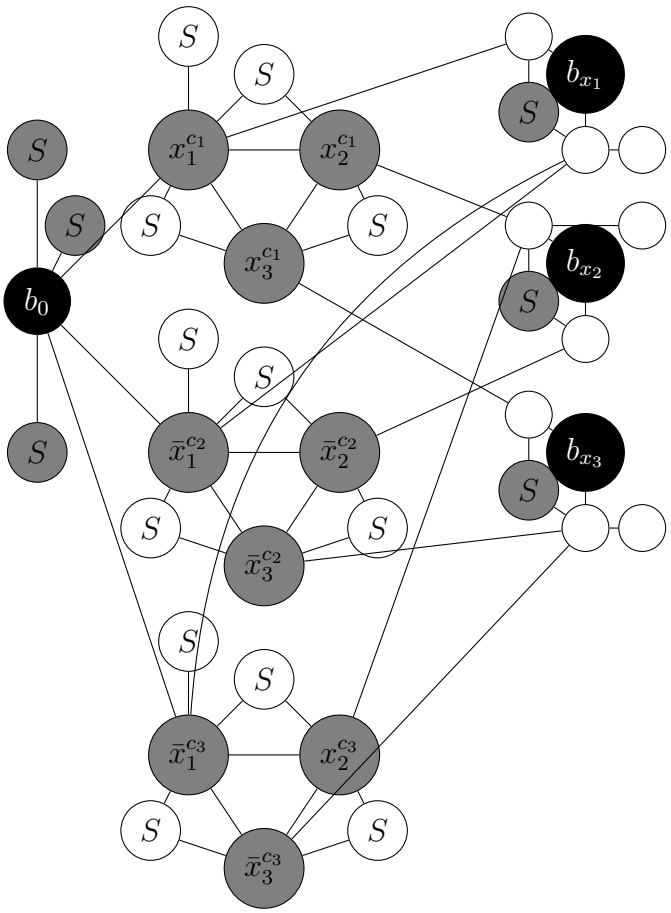


Figure 9: An example for a full construction of variable gadgets and clause gadgets for $U \coloneqq \{x_1, x_2, x_3\}$ and $C \coloneqq \{(x_1, x_2, x_3), (\bar{x_1}, \bar{x_2}, \bar{x_3}), (\bar{x_1}, x_2, \bar{x_3})\}$.

majority of the neighborhood of $x_i^\top$ consists of permanently stable white vertices. As a result, the vertex $x_i^\top$ cannot update to gray afterwards.

3.2.3 Correctness of the Reduction

In Figure 9 we have an example of an influence network constructed by using the reduction from 3Sat to Optimistic Update Sequence. We show that a Optimistic Update Sequence instance is a YES-instance if and only if the corresponding 3Sat instance is a YES-instance. If this is the case, then

$$\text{3Sat} \preceq_p \text{Optimistic Update Sequence}.$$

$\Rightarrow$:
Let (U, C) be a YES-instance of 3Sat with $x_1, ..., x_n \in U$ and $c_1, ..., c_m \in C$ such that $|U| = n$ and $|C| = m$. Let $\delta : U \to \{\top, \bot\}$ be a boolean assignment such that every $c \in C$ contains at least one literal assigned to true. Let $(G, \circ)$ be the Optimistic

Update Sequence instance constructed from (U, C) as described above. We show that there is a sequence σ such that $\circ[G, \sigma]$ satisfies $|\{v \in V \mid \circ(v) = 2\}| \geq n + 1$.
We define $U_1, U_2 \subseteq U$ with $U_1 \cup U_2 = U$, $U_1 := \{x_i \in U \mid \delta(x_i) = \top\}$ and $U_2 := \{x_i \in U \mid \delta(x_i) = \bot\}$. Since σ assigns every variable with either true or false, it holds that $U_1 \cap U_2 = \emptyset$.
We construct the update sequence σ with the following steps:

1. for each $x_i \in U_1$ append $\{x_i^{\bot}\}$ to σ,
2. for each $x_i \in U_1$ append $\{x_i^c\}$ for every $c \in C$ that contains x_i to σ,
3. for each $x_i \in U_1$ append singletons of all white vertices connected to $x_i^{\bot}$ to σ,
4. for each $x_i \in U_2$ append $\{x_i^{\top}\}$ to σ,
5. for each $x_i \in U_2$ append $\{\bar{x}_i{}^c\}$ for every $c \in C$ that contains $\bar{x}_i$ to σ,
6. for each $x_i \in U_2$ append singletons of all white vertices connected to $x_i^{\top}$ to σ.

For each clause $c \in C$ there is at least one literal which is assigned to true. Hence, for each clause gadget CG(c) there is at least one gray vertex such that the corresponding variable is an element of U_1. For that reason, each clause gadget contains at least one vertex which is part of σ after these six steps. Also, each vertex $\{x_i^{\bot}\}$ for $x_i \in U_1$ did change opinion to gray. Further, we update the white vertices in the neighborhood of such a vertex $\{x_i^{\bot}\}$. Thus, the vertex cannot update to white afterwards when the remaining neighbors update to white. The same procedure is done for all $x_i \in U_2$ in the third and fourth step. At this point each clause contains at least one white literal and the black vertices of all variable gadgets are stable. Afterwards we continue the update sequence with the following step:

- We append singletons of all unstable gray vertices from all clause gadgets to σ.

For $\circ[G, \sigma]$ we make the following observations:

- every clause gadget CG(i) for $i \in [0, m]$ consists of stable white vertices,
- the black vertex b_0 is connected to m white vertices and m permanently stable gray vertices, thus is stable,
- for each $x_i \in U$ the vertices $x_i^{\top}$ and $x_i^{\bot}$ have different non-preferred opinions and are stable,
- for all $x_i \in U$ the black vertex b_{x_i} is connected to a white vertex and a gray vertex, thus is stable, and
- all vertices are stable.

The update sequence σ is an update sequence on $(\mathrm{G}, \circ)$ resulting in a stable outcome such that b_0 as well as all b_{x_i} for $x_i \in U$ are black. This concludes that there is an update sequence σ satisfying

$$|\{v \in V \mid \circ(v) = 2\}| \geq n + 1.$$

In the end, whenever the 3Sat instance is a YES-instance, then the Optimistic Update Sequence instance is a YES-instance.

$\Leftarrow$:
Let (U, C) be a 3Sat instance with $x_1, ..., x_n \in U$ and $c_1, ..., c_m \in C$ such that $|U| = n$ and $|C| = m$. Let $(G, \circ)$ be the Optimistic Update Sequence instance constructed from (U, C) as described above. We show that there is a truth assignment $\delta : U \to \{\top, \bot\}$ such that every clause $c \in C$ contains at least one literal assigned with true when there is a stable update sequence σ with $[G, \sigma]$ satisfying $|\{v \in V \mid \circ(v) = 2\}| \geq n + 1$.

Assume that there is a σ such that $[G, \sigma]$ satisfies $|\{v \in V \mid \circ(v) = 2\}| \geq n + 1$ but the 3Sat instance is no YES-instance. For every fulfilling Optimistic Update Sequence instance we observe the following properties for $(\mathrm{G}, \circ)$:

1. There is no sequence σ such that $[G, \sigma]$ contains more than $n + 1$ black vertices,
2. There is no sequence σ such that $[G, \sigma]$ contains black vertices besides the initial black vertices,
3. For a stable sequence σ such that b_0 remains black all connected vertices from clause gadgets are white,
4. All vertices in all clause gadgets are white, and
5. For a stable sequence σ and a variable $x_i \in U$ when b_{x_i} is black then either $x_i^\top$ or $x_i^\bot$ is gray, while the other vertex is white.

For Property 1 and Property 2:
We assume that there are more than $n + 1$ black vertices. Therefore, there must be a vertex which did update to black first. This vertex needs a black majority neighborhood, thus only a vertex in the neighborhood of the initial black vertices can update to black. The neighbors of the black vertex b_0 are either permanently stable or have a neighborhood containing more than one non-black vertex while only having one black vertex in their neighborhood. Hence, none of these vertices updates to black first. The neighborhood of any black vertex b_{x_i} consists of $x_i^\top$ and $x_i^\bot$. Both have at least three neighbors with only one of them being a black vertex, thus none of these update to black first either. As a result, there is no update sequence σ such that $[G, \sigma]$ contains more than $n + 1$ black vertices. This implies that there is no vertex besides the initial black vertices which can update to black. Consequently, Property 2 holds.
For Property 3:

Because of Property 1 and Property 2 the vertex b_0 is black for any sequence σ such that $[G, \sigma]$ satisfies $|\{v \in V \mid \circ(v) = 2\}| \geq n + 1$. Therefore, $\text{maj}(b_0, \circ[G, \sigma]) = 2$. The m permanently stable vertices in the neighborhood of b_0 do not update to another opinion and no vertex can update to black. Hence, in order to remain black the remaining neighbors need to update to white. This implies that Property 3 holds.

For Property 4:
Because Property 3 holds, each clause gadget contains at least one vertex which updates to white. The clause gadget is constructed in a way such that whenever at least one vertex updates to white, then for a stable update sequence the remaining gray vertices updates to white as well. Additionally, none of these vertices can update to gray afterwards, because the majority of the neighborhood is permanently stable and white. As a result, Property 4 holds.

For Property 5:
Due to Property 1 and Property 2 each vertex b_{x_i} for $x_i \in U$ is black for any sequence σ such that $[G, \sigma]$ satisfies $|\{v \in V \mid \circ(v) = 2\}| \geq n + 1$. Hence, the neighborhood of b_{x_i} needs to contain as many gray vertices as white vertices. Also, the neighborhood consists of only $x_i^\top$ and $x_i^\bot$. Either $x_i^\top$ or $x_i^\bot$ remains white while the other updates to gray. As a result, Property 5 holds.

To summarize, all initially black vertices remain black, all vertices in the clause gadgets update to white, and for any $x_i \in U$ the vertices $x_i^\top$ and $x_i^\bot$ have different non-preferred opinions for $\circ[G, \sigma]$. Therefore, we assign each variable $x_i \in U$ in the corresponding 3Sat instance (C, U) with truth values as follows:

- When $\circ(x_i^\top) = 0$, then we assign the variable x_i to **true**.
- When $\circ(x_i^\bot) = 0$, then we assign the variable x_i to **false**.

The 3Sat instance is a YES-instance, when all clauses contain a positive literal assigned to true or a negative literal assigned to false. This is the case, when each clause gadget in the corresponding Optimistic Update Sequence instance is connected to a white vertex from a variable gadget.

Let us assume, that there is a clause gadget which is not connected to a white vertex from any variable gadget but all vertices in the clause gadget are white. For the Optimistic Update Sequence instance we found an optimistic update sequence such that all $n + 1$ black vertices remain black for a stable outcome. As a result, Property 4 holds. Therefore, for each clause gadget at least one gray vertex needed to update to white. This indicates, that for each clause gadget one gray vertex is connected to a white vertex from a variable gadget. Further, the gray vertex of the clause gadget cannot be connected to a $x_i^\top$ or $x_i^\bot$ that is finally gray. The only chance of such a vertex to become gray is that no gray neighbor from the clause gadgets has yet updated to white. However, then the gray vertices of the clause gadget did not update to white. This is a contradiction to our assumption, that finally all clause gadgets contain only white vertices. As a result, each clause contains a literal assigned with true. Thus, the

3Sat instance is a YES-instance. If and only if the Optimistic Update Sequence instance $(G, \circ)$ constructed from a 3Sat instance (U, C) is a YES-instance, then (U, C) is a YES-instance as well.

With "$\Rightarrow$" and "$\Leftarrow$", we have shown that 3Sat $\preceq_p$ Optimistic Update Sequence. As a result, Optimistic Update Sequence is NP-hard. In Section 3.1, we have shown that Optimistic Update Sequence is in NP. Altogether, this proves that Optimistic Update Sequence is NP-complete.

We note that the optimistic update sequences for the instances resulting from the reduction from 3Sat to Optimistic Update Sequence update each vertex at most once. Thus, the update sequence is a balanced asynchronous update sequence. As a result, Optimistic Update Sequence is even NP-complete for balanced asynchronous update sequences.

4 Complexity on Restricted Graph Classes

In Section 3.2 we have shown that Optimistic Update Sequence is NP-complete. In this section we investigate Optimistic Update Sequence in case of restricted influence networks. In particular, we consider planar influence networks as well as influence networks with constant degree. Also, we investigate influence networks without cycles. We focus on whether Optimistic Update Sequence is still hard for these restrictions.

4.1 Constant Degree

An influence network $(G, \circ)$ with constant degree is a network such that the maximum degree $\triangle(G)$ is limited to c for some constant $c \in \mathbb{N}$. There are $NP - complete$ graph problems such that the problem is easy for limiting the maximum degree of graphs to a constant c. For the example of Clique, there is an algorithm solving the problem in polynomial time for for small values of c. (see Garey and Johnson [GJ99, p. 84]). In the following sections, we investigate the time complexity of Optimistic Update Sequence for small values of $\triangle(G)$.

4.1.1 Polynomial Time Complexity

We show, that Optimistic Update Sequence for an influence network $(G, \circ)$ with $\triangle(G) < 3$ is easy. First, we restrict $\triangle(G)$ to 1.

In such an influence network G the connected components in G contain either one single vertex or two vertices sharing an edge. We can compute the maximum number of black vertices for two connected components separately. All components with only one vertex are stable. Consequently, we cannot update vertices in these components. All components with two vertices not containing a black vertex can be updated arbitrary. When a connected component contains a black vertex, then the other vertex is either black or can update to black. In order to maximize the number of black vertices, we update all vertices sharing an edge with a black vertex. Afterwards, all vertices are stable. We can compute an optimistic update sequence for a connected component in $\mathcal{O}(1)$ time. Further, the number of connected components in G is upper bounded by the number of vertices n. As a result, computing an optimistic update sequence σ for $\triangle(G)$ can be done in $\mathcal{O}(n)$ time.

Now, we show that Optimistic Update Sequence is still easy for in case of $\triangle(G) = 2$. For a connected component $G' \subseteq G$ with $\triangle(G') = 2$ we know that G' is either a *cycle* or a *path*. An illustration for updating vertices in a cycle can be found in Figure 10.

Let us assume, that G' is a cycle. Each vertex $v \in G'$ satisfies $|N(v)| = 2$. At first, we update all unstable vertices with black majority neighborhood to black. By updating these vertices, we notice that an unstable vertex with black majority neighborhood has two black neighbors. Further, any unstable vertex v has two neighbors with the same opinion differing from $\circ(v)$. After updating all unstable vertices with black majority neighborhood we cannot update additional vertices to black. From there on, we want

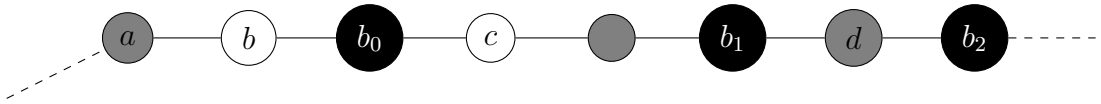


Figure 10: An example of a cyclic graph where we can update d to make b_1 and b_2 stable. On the other side, we cannot manipulate b or c to stabilize b_0. The vertex b is stable for all non-preferred opinions of the neighbor a.

to minimize the number of black vertices updating to any non-preferred opinion. For that reason, we investigate whether we can manipulate the graph such that an unstable black vertex does not update. However, an unstable black vertex b has a neighborhood of two vertices sharing the same non-preferred opinion. We know, that these vertices cannot update to black. Let us assume, that for b the neighbor x updates to the other non-preferred opinion. However, the vertex x has exactly one black neighbor. Therefore, x is stable, which is a contradiction to the assumption. Accordingly, this holds for the other neighbor of b. Consequently, there is no update sequence such that an unstable black vertex remains black after we did update all unstable vertices with black majority neighborhood. Summarizing, after updating all unstable vertices with black majority neighborhood we can safely update all other unstable vertices. We visit every vertex at most twice and need polynomial time to do so. Thus, computing an optimistic update sequence σ can be done in polynomial time, more specific in $\mathcal{O}(n)$ time.

We have shown, that Optimistic Update Sequence is easy for cycles. A path contains exactly two vertices with one neighbor each. The remaining vertices have a neighborhood of size two each. In the following, we show that we can reduce each graph which is a paths to a graph which is a cycle. Therefore, in the end of the reduction each vertex shall have exactly two neighbors. We add a P_6 to a given path and create edges between the initial path and the P_6 such that the resulting graph is a cycle. Therefore, we create edges between the vertices of the path with just one neighbor and the vertices of the P_6 with just one neighbor. We show that the number of black vertices for an optimistic update sequence on the initial set of vertices remains the same.

The vertices for a given path with only one neighbor are labeled as s and t. We add the P_6 to the graph such that for graphs the vertex s has the same opinion at the end of a stable optimistic update sequence. The same goes for t. For cycles we have shown that a vertex v is unstable when both neighbors have the same opinion and this opinion is different from the opinion of v. We use this property to make the vertices of the P_6 permanently stable and change the size of the neighborhood of s and t such that the majority opinion of the neighborhood does not change. In Figure 11 we have shown an example for the reduction.

We begin the reduction by adding six vertices to the graph.

- We add the vertex s_1, which has the same opinion as the neighbor of s.
- We add the vertex s_2, which has a opinion differing from the opinion of s and s_1.
- We add the vertex s_3, which has the same opinion as s_2.

- We add the vertex t_1, which has the same opinion as the neighbor of t.
- We add the vertex t_2, which has a opinion differing from the opinion of t and t_1.
- We add the vertex t_3, which has the same opinion as t_2.

These vertices have no neighbors yet. Therefore, we create the following edges.

- We add an edge between s and s_1, s_1 and s_2 as well as an edge between s_2 and s_3.
- Similarly, we add an edge between t and t_1, t_1 and t_2 as well as an edge between t_2 and t_3.
- Further, we add an edge between s_3 and t_3.

Each vertex in this graph has exactly two neighbors. Clearly, the vertices s_2, s_3, t_2 and t_3 are stable. Further, the vertex s_1 is stable and remains to be stable even if the vertex s updates. This also holds for the relation between t_1 and t. As a result, the update sequence does not contain singletons of the newly added vertices. The graph is a cycle, thus we can compute the optimistic update sequence by just appending singletons of all non-black vertices with black majority neighborhood. Afterwards, we append singletons of the remaining unstable vertices. This can be done in polynomial time. We show that the vertices s and t have the same opinion for the path and the cycle constructed from the reduction from this path.

In the path, the vertex s updates when the sole neighbor of s has a different opinion. We label the neighbor with a. However, when a is unstable, then a may update first and s remains stable. As a result, we know that s updates, when a does not update first.

In the cycle, the vertex s updates if both neighbors have the same opinion and this opinion is different from the opinion of s. By definition the vertex s_1 has the same opinion as a. Further s_1 is stable by definition. However, the vertex a may be unstable. If a is unstable and updates before s does, then s becomes stable without updating. Otherwise, s updates to the opinion of s_1. As a result, we know that s updates when a does not update first.

In the end, in both graphs the vertex s has the same opinion at the end of an stable optimistic update sequence. This also holds for t as well. The remaining vertices of the path have the same neighborhood as the corresponding vertices in the reduction to a cycle. Consequently, the number of black vertices in the path is equal to the number of black vertices in the cycle subtracted by the number of black vertices in the added P_6. As a result, we have shown, that we can reduce every path to a cycle for Optimistic Update Sequence. In terms of time complexity, we visit every vertex constantly often to compute an optimistic update sequence and need polynomial time to do so. Thus, computing an optimistic update sequence can be done in polynomial time for cycles and paths; more specific in $\mathcal{O}(n)$ time.

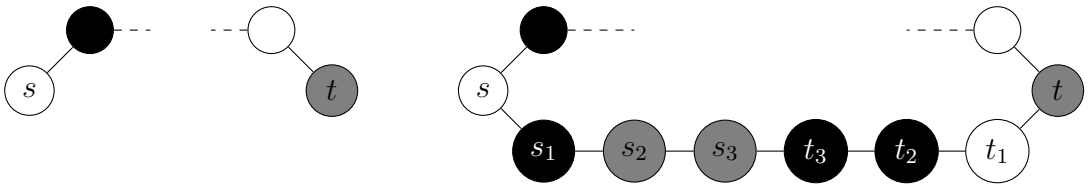


Figure 11: An example for a reduction from a path to a cycle.

4.1.2 NP-hardness

We show, that Optimistic Update Sequence is NP-hard for $\triangle(G) = 7$. We show this by modifying the reduction from Section 3.2. We want the modified instance to have properties as follows:

- Each vertex has at most seven neighbors.
- There is an optimistic update sequence σ such that the number of black vertices is at least k, if and only if there is a truth assignment satisfying the formula of the corresponding 3Sat instance.

All clause gadgets are connected to a common neighbor—the black vertex b_0. As a result, the degree of b_0 is lower bounded by some function depending on $|C|$. Further, the variable gadgets have vertices whose degree is bounded by the occurrence of a literal in the clauses. In the following, we explain the modification done in these gadgets.

Clause Gadget

We recap the idea of the clause gadget. It ensures that all vertices of the gadget update to white when at least one literal of the corresponding clause is assigned to true.

Gadget Structure

In the previous clause gadget, we used the black vertex b_0 to connect all clause gadget to this vertex. However, the size of the neighborhood of b_0 is twice as big as the number of clauses in C. Instead of creating an edge from each clause gadget to b_0, we create a black vertex for each clause gadget separately. We label the black vertex for the clause gadget CG(i) with b_{c_i}. We connect the vertex b_{c_i} to a permanently stable gray vertex and one gray vertex from the clause gadget CG(i). To this end, each clause gadget has four gray vertices instead of three. The first three vertices still represent the literals. The new gray vertex is used to reduce the maximum degree $\triangle(\text{CG(i)})$. In the initial version of the clause gadget, we randomly selected a gray vertex which is connected to b_{c_i}. However, instead of connecting a random selected gray vertex to b_{c_i}, we connect this fourth gray vertex to b_{c_i}. The four gray vertices are connected in a circular structure to each other. For each pair of gray vertices sharing an edge, we add a permanently stable white vertex to the clause gadget and connect it to both gray vertices. An example for the new clause gadget can be seen in Figure 12.

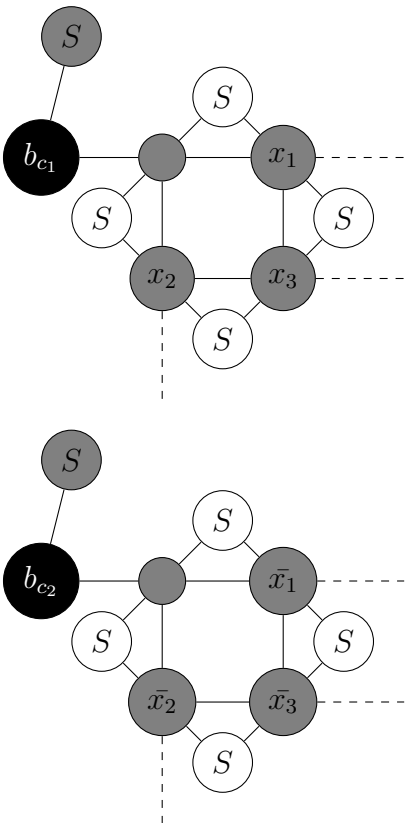


Figure 12: An example of two clause gadgets for the constant degree reduction. Each clause gadget is connected to another black vertex.

Properties

We observe that the maximum degree of a clause gadgets CG(i) is reduced to five.

When at least one of the gray vertices turns white, then the remaining gray vertices are unstable with a white majority neighborhood. Therefore, the clause gadget still works in the same way as in the previous version described in Figure 8.

Variable Gadget

We recap the idea of the variable gadget. A variable gadget for the variable x_i contains a black vertex b_{x_i} such that the black vertex only remains black when $x_i^\top$ and $x_i^\top$ have different opinions. The size of the neighborhood of the vertices $x_i^\top$ and $x_i^\perp$ needs to be reduced for a graph with constant degree. In order to reduce the number of neighbors for these vertices, we introduce three new gadgets which are used instead of the variable gadget: the *connector gadget*, the *merge gadget* and the *switch gadget*.
The connector gadget is connected to a maximum of four vertices from clause gadgets that represent x_i respectively. Further, the connector gadget ensures that these vertices always have the same opinion for every stable optimistic update sequence.
The merge gadget is connected to up to two connector gadgets and will ensure that

the x_i representing vertices from the clause gadgets connected to these two connector gadgets always have the same opinion for every stable optimistic update sequence.
The switch gadget is connected to a merge gadget for x_i and another merge gadget for $\bar{x}_i$ and will ensure that the merge gadget for x_i and the merge gadget for $\bar{x}_i$ correspond to different opinions.

Connector gadget

The connector gadget is connected to up to four x_i representing vertices from clause gadgets. The idea of the connector gadget is that all the connected vertices from the clause gadgets always have the same opinion. Thus, we construct the connector gadget such that the number of black vertices in the connector gadget is maximized when all connected vertices from the clause gadgets have the same opinion.

Note, that we may use multiple connector gadgets and merge gadgets for every literal. We use an index z to label specific vertices in the connector gadgets. We use this labeling to clarify, which gadgets are connected to each other. The first connector gadget uses $z = 0$. Further, we increment z by two for every new connector gadget.

Gadget Structure

At first we create the two white vertices $w(x_i, z)$ and $w(x_i, z + 1)$. We connect both of them to a common permanently stable gray vertex. Further, we create the two black vertices $a(x_i, [z, z + 1])$ and $b(x_i, [z, z + 1])$. The vertex $a(x_i, [z, z + 1])$ is connected to $w(x_i, z)$, $w(x_i, z+1)$ and two permanently stable gray vertices. The vertex $b(x_i, [z, z+1])$ is connected to $w(x_i, z)$, $w(x_i, z + 1)$ and three permanently stable white vertices. An example for a connector gadget can be found in Figure 13.

Connection

The vertex $b(x_i, [z, z + 1])$ has two outgoing edges to two black vertices from a merge gadget. The white vertices $w(x_i, z)$ and $w(x_i, z + 1)$ are connected to two gray vertices from the clause gadgets each. That is, instead of being connect to a variable gadget, the gray vertices of the clause gadgets are connected to the white vertices of connector gadgets. However, one connector gadget can be connected to at most four x_i^c. Therefore, we use multiple connector gadgets, if needed. Additionally, when there are less then four x_i^c left, then we create additional gray vertices and connect them to the connector gadget.

Properties

The maximum degree of a connector gadget is seven, because $b(x_i, [z, z + 1])$ has the highest degree with $\deg(b(x_i, [z, z + 1])) = 7$.

For the connection between an vertex x_i^c from the clause gadget CG(c) and the vertex $w(x_i, z)$ from the connector gadget, we observe that x_i^c is unstable with a white majority

neighborhood. Also, $w(x_i, z)$ is unstable with a gray majority neighborhood. However, if x_i^c updates to white before $w(x_i, z)$ updates to gray, then both vertices are stable. The vertex x_i^c cannot update to gray after updating to white due to the structure of the clause gadget. Also, the vertex $w(x_i, z)$ cannot update to gray after x_i^c updated to white, because $w(x_i, z)$ is connected to a permanently stable gray vertex, a permanently stable white vertex and two black vertices. In order for $w(x_i, z)$ to update to gray, both black vertices would need to update to gray. However, the vertex $b(x_i, [z, z+1])$ cannot update to gray, because the neighborhood of $b(x_i, [z, z+1])$ consists of four permanently stable white vertices in a neighborhood of size seven. As a result, the vertices x_i^c and $w(x_i, z)$ are stable after x_i^c updates to white. Similarly, when $w(x_i, z)$ updates to gray before x_i^c updates, then $w(x_i, z)$ cannot turn white afterwards, because the vertex is connected to two gray vertices and two black vertices. However, the vertex x_i^c can still update to white if another gray vertex in the clause gadget updates to white. In order for $w(x_i, z)$ to update to white after updating to gray however, both black vertices would need to update to white. The vertex $a(x_i, [z, z+1])$ cannot update to white, because it has two permanently stable gray vertices with a neighborhood of size four.

We constructed the connector gadget such that only one of the two black vertices of the connector gadgets can remain black. In the following we describe all possible combinations of opinions for $w(x_i, z)$ and $w(x_i, z+1)$ and the consequences for the black vertices. When $w(x_i, z)$ and $w(x_i, z+1)$ update to gray, then the black vertex $a(x_i, [z, z+1])$ updates to gray and $b(x_i, [z, z+1])$ remains black. When $w(x_i, z)$ and $w(x_i, z+1)$ remain white, then $a(x_i, [z, z+1])$ remains black and $b(x_i, [z, z+1])$ updates to white. However, when just one of the two white vertices update to gray and the other remains white, then $a(x_i, [z, z+1])$ updates to gray and $b(x_i, [z, z+1])$ updates to white. Summarizing, one black vertex remains black in a connector gadget if all four connected clause gadget vertices have the same opinion. No black vertex remains black in a connector gadget if at least one of the four connected clause gadget vertices has a different opinion. As a result, for all optimistic update sequences in each connector gadget at least one black vertex remains black.

Merge Gadget

The idea of the merge gadget is to combine two connector gadgets such that the both connector gadgets have the same opinion. Further, the merge gadget is constructed such that we can also combine two merge gadgets to ensure that the two combined merge gadgets have the same opinion. In order to do so, a merge gadget consists of two separate parts. We label the connected vertices from the two connector gadgets with $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ with $l_1, l_2, l_3 \in \mathbb{N}$ and $l_1 < l_2 < l_3$.

Graph Structure

The first part is labeled with $A(x_i, [l_1, l_3])$ and the second part is labeled with $B(x_i, [l_1, l_3])$. Both parts are connected to the vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ from the connector gadgets. For $A(x_i, [l_1, l_3])$, we create the gray vertex $g(x_i, [l_1, l_3])$. This vertex

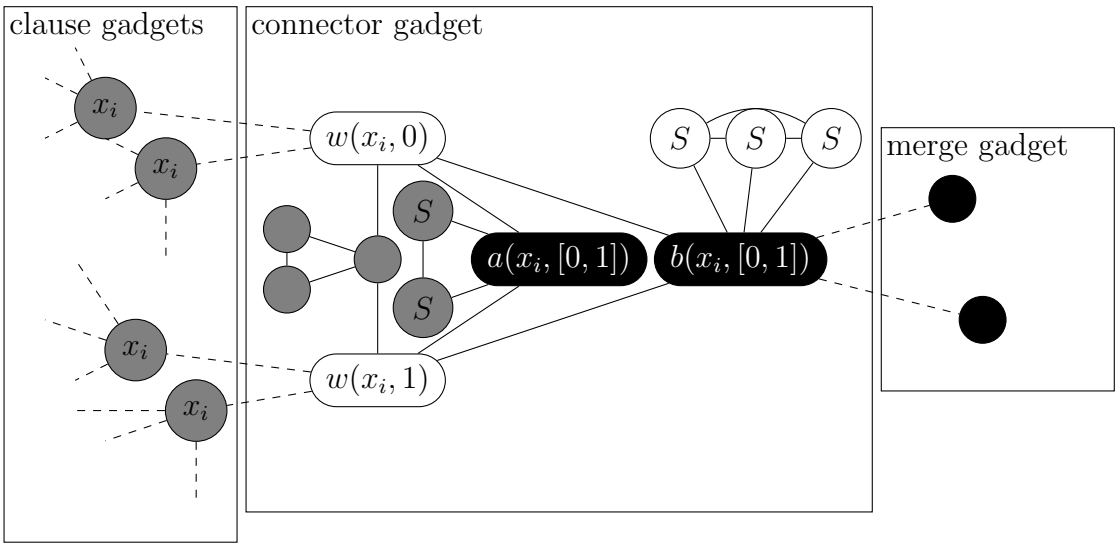


Figure 13: A connector gadget connected to four clause gadgets on x_i representing vertices. The connector gadget is connected to a merge gadget via two edges from the vertex $b(x_i, [0, 1])$.

is connected to $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ from the connector gadgets. Further, $g(x_i, [l_1, l_3])$ is connected to a permanently stable gray vertex and two permanently stable white vertices. Additionally, $g(x_i, [l_1, l_3])$ is connected to a gray vertex, which is connected to another gray vertex which is connected to the black vertex $a(x_i, [l_1, l_3])$. The vertex $a(x_i, [l_1, l_3])$ is also connected to $g(x_i, [l_1, l_3])$.

For $B(x_i, [l_1, l_3])$, we create the black vertex $b(x_i, [l_1, l_3])$, which is connected to two permanently stable white vertices as well as to the vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$. An example for a merge gadget is shown in Figure 14.

Connection

The merge gadget is constructed such that we can either combine two connector gadgets or merge gadgets. As a result, we construct the part of $B(x_i, [l_1, l_3])$ for two scenarios. In the first scenario, the vertex $b(x_i, [l_1, l_3])$ is connected to a gray vertex from the switch gadget. We label the gray vertex with $x_i^\top$, because it has a similar functionality as the vertex in the variable gadget with the same label. Note, that when the merge gadget represents the negative literal $\bar{x}_i$ instead, the black vertex is labeled $b(\bar{x}_i, [l_1, l_3])$ and is connected to $x_i^\bot$ instead. In the second scenario however, the vertex $b(x_i, [l_1, l_3])$ is connected to another merge gadget with two outgoing edges. In this case, we add three permanently stable white vertex instead of two permanently stable white vertices to $B(x_i, [l_1, l_3])$ and connect them to $b(x_i, [l_1, l_3])$. An example for the first scenario is shown in Figure 14. An example for the second scenario is shown in Figure 15.

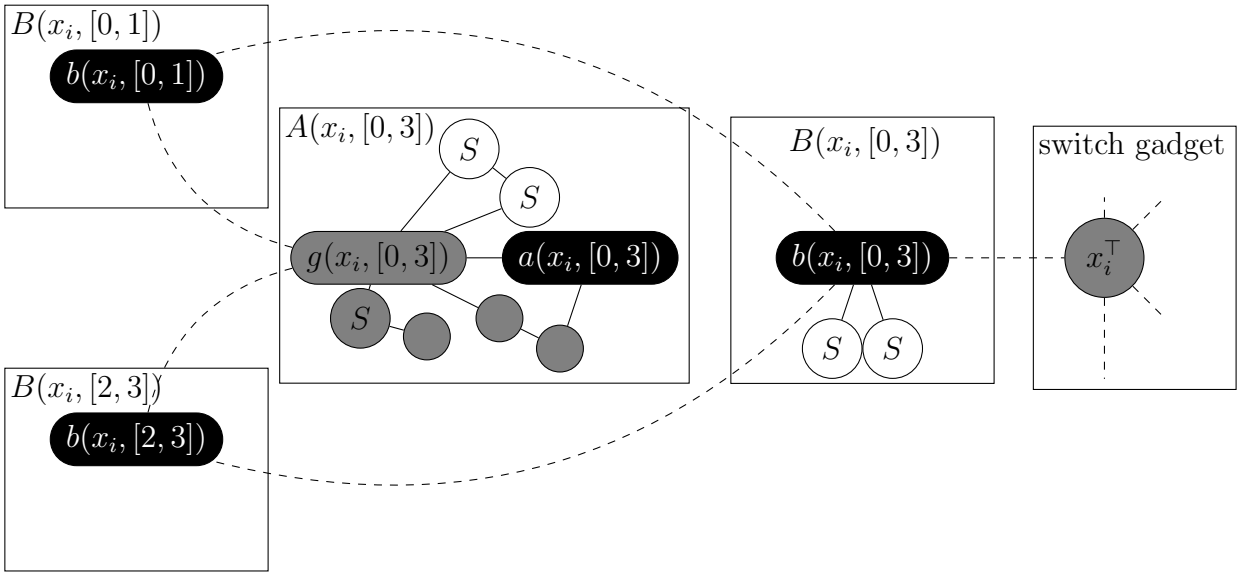


Figure 14: An example of an merge gadget which combines the two connector gadget on the vertices $b(x_i, [0, 1])$ and $b(x_i, [2, 3])$. The first part of the merge gadget is $A(x_i, [0, 3])$. The second part is $B(x_i, [0, 3])$, which is also connected to the gray vertex $x_i^\top$ from a switch gadget

Properties

The maximum degree of a merge gadget is seven, because amongst others the vertex $g(x_i, [l_1, l_3])$ has the highest degree with $\deg(g(x_i, [l_1, l_3])) = 7$.

For the connector gadgets, we observe that the vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ can either remain black or update to white. We observe the behaviour of the merge gadget for all combinations of opinions for the incoming connector gadget vertices. There are three possible combinations for the vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ from the connector gadgets. First, the vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ remain black. Second, the vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ update to white. Third, only one of these two vertices updates to white. There are further cases, because these two vertices can either remain black or update to white.

When $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ remain black, then the vertex $b(x_i, [l_1, l_3])$ remains black, but $a(x_i, [l_1, l_3])$ updates to gray. This behaviour for $A(x_i, [l_1, l_3])$ is not obvious, thus we go through the update steps. The vertex $g(x_i, [l_1, l_3])$ is connected to two gray vertices, two white vertices and three black vertices. As a result, the vertex $g(x_i, [l_1, l_3])$ is stable and remains gray. Then the vertex $a(x_i, [l_1, l_3])$ updates to gray, because the neighborhood consists of two gray vertices.

When $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ update to white, then the vertex $b(x_i, [l_1, l_3])$ updates to white, but $a(x_i, [l_1, l_3])$ remains black. Again, we observe the gray vertex $g(x_i, [l_1, l_3])$. Now, the vertex has four white neighbors, two gray neighbors and one black neighbors. As a result, the vertex $g(x_i, [l_1, l_3])$ updates to white. The vertex $a(x_i, [l_1, l_3])$

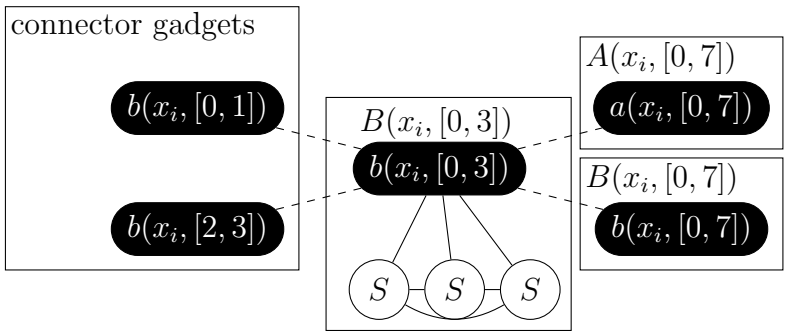


Figure 15: The second part of a merge gadget, labeled with $B(x_i, [0, 3])$, is connected to black vertices of two incoming connector gadgets;$b(x_i, [0, 1])$ and $b(x_i, [2, 3])$. The vertex $b(x_i, [0, 3])$ is connected to three permanently stable white vertices instead. Additionally, $B(x_i, [0, 3])$ is connected to another merge gadget on the outgoing edges.

has one white neighbor and one gray neighbor now, thus is stable and remains black.

When just one of the two vertices $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ updates to white and the other remains black, then $a(x_i, [l_1, l_3])$ updates to gray and $b(x_i, [l_1, l_3])$ updates to gray. This implies, that for this merge gadget, we maximize the number of black vertices, when $b(x_i, [l_1, l_2])$ and $b(x_i, [l_2 + 1, l_3])$ have the same opinion.

Switch Gadget

The switch gadget essentially keeps the functionality of the variable gadget with some perks to the structure. The switch gadget for the variable x_i is connected to one merge gadget for x_i and one merge gadget for $\bar{x}_i$. We want the vertices of the two merge gadgets to have differing opinions. Therefore, the vertex b_{x_i} shall remain black if and only if the vertices of the connected merge gadgets have different opinions. Otherwise, b_{x_i} updates to a non-preferred opinion. The vertex b_{x_i} updates to white if both neighbors are white. If both neighbors are gray, then b_{x_i} updates to gray.

Graph Structure

For the switch gadget structure we use a similar structure as for the variable gadget. However, the vertex $x_i^\top$ is connected to two permanently stable white vertices instead of a varying number of white vertices. The same holds for $x_i^\bot$. Also, $x_i^\top$ is connected to just one vertex from a merge gadget, instead of all x_i^c from clause gadget CG(c) for $c \in C$. Again, this also holds for $x_i^\bot$. Further, the initial opinion of $x_i^\top$ and $x_i^\bot$ is gray. An example for a switch gadget is shown in Figure 16.

Properties

The maximum degree of a switch gadget is five, because $x_i^\top$ has the highest degree with $\deg(x_i^\top) = 5$.

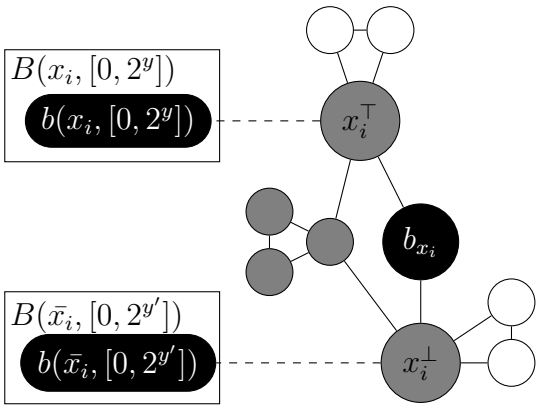


Figure 16: An example of a switch gadget connected to a merge gadget for x_i and a merge gadget for $\bar{x}_i$. The vertices $b(x_i, [0, 2^y])$ and $b(x_i, [0, 2^{y'}])$ can be either white or black. If $b(x_i, [0, 2^y])$ is black, then the vertex $x_i^\top$ updates to white. The idea is that just one of both merge gadget vertices remains black and the other updates to white.

We label the vertices from the merge gadgets with $b(x_i, [0, z])$ and $b(\bar{x}_i, [0, z'])$. The value of z is the number of connector gadgets for the literal x_i in the graph and the value of z' is the number of connector gadgets for the literal $\bar{x}_i$. For the merge gadgets, we observe that the vertices $b(x_i, [0, z])$ and $b(\bar{x}_i, [0, z'])$ can either remain black or update to white. Therefore, we check the behaviour of the switch gadget for all combinations of opinions for the incoming merge gadget vertices. When $b(x_i, [0, z])$ and $b(\bar{x}_i, [0, z'])$ remain black, then $x_i^\top$ and $x_i^\bot$ remain gray. As a result, b_{x_i} updates to gray. When $b(x_i, [0, z])$ and $b(\bar{x}_i, [0, z'])$ update to white, then $x_i^\top$ and $x_i^\bot$ update to white. In this case, b_{x_i} updates to white. However, when just one of the two vertices $b(x_i, [0, z])$ and $b(\bar{x}_i, [0, z'])$ updates to white and the other remains black, then either $x_i^\top$ or $x_i^\bot$ update to white while the other one remains gray. Hence, b_{x_i} remains black.

Combination

We have introduced the base functionalities of the connector gadget, merge gadget and switch gadget. Now, we explain how they interact with each other in full detail and how they connect to the clause gadgets.

For a variable x_i, we connect the gray vertices x_i^c for all $c \in C$ to connector gadgets such that each connector gadget is connected to four gray vertices x_i^c. Note, that there may be connector gadgets connected to less then four gray vertices. In this case, we just add gray additional gray vertices to these connector gadgets. Further, the number of connector gadgets for any variable $x_i \in U$ is supposed to be 2^y for a $y \in \mathbb{N}$. Otherwise, we add more connector gadgets until the number of connector gadgets can be described by 2^y for some $y \in \mathbb{N}$. This number of connector gadgets is necessary in order to combine all connector gadgets to each other via merge gadgets. As mentioned before, the two black vertices of a connector gadget are labeled with $a(x_i, [z, z+1])$ and $b(x_i, [z, z+1])$ for some $z \in \mathbb{N}$. The

use of the index z is described in Section 4.1.2. For every second connector gadget we add a merge gadget to the graph, thus we have 2^{y-1} merge gadgets in the graph. Sorted by the index z, we always connect two connector gadgets to one merge gadget. The two parts of the merge gadget combining the connector gadgets with the black vertices $b(x_i, [0, 1])$ and $b(x_i, [2, 3])$ are labeled $A(x_i, [0, 3])$ and $B(x_i, [0, 3])$. Further, we add 2^{y-2} additional merge gadgets to the graph in order to combine the previously created merge gadgets. From there on, we always combine the new created merge gadgets in pairs of two to another merge gadget until there is only one merge gadget left without outgoing edges. So far, the structure of the combination of merge gadgets can be observed in Figure 17. In the end, the remaining merge gadget is connected to the switch gadget for the variable x_i. An example for the connection between the clause gadget, connector gadget, merge gadget and switch gadget for the variable x_i is shown in Figure 18.

We observe the structure and the relation to the number of stable black vertices for a stable outcome. We know, that each connector gadget and merge gadget has two black vertices. Previously, we have shown that for any update sequence σ at most one black vertex can remain black for each connector gadget and merge gadget. When all connector gadget and merge gadgets have a black vertex for a stable graph, then all the white vertices $w(x_i, j)$ for $j \in 2^{y+1}$ must finally have the same opinion. When just the vertex $w(x_i, 1)$ has a different opinion, then at least y additional black vertices update to a non-preferred opinion. Both black vertices update to a non-preferred opinion for the connector gadget containing the vertex $w(x_i, 1)$. Further, all black vertices from merge gadgets such that the label indicates a connection to $w(x_i, 1)$ update to a non-preferred opinion as well;i.e., the vertex $b(x_i, [0, 3])$ updates to white because the range of $[0, 3]$ contains 1. In the end, both black vertices update for the connector gadget containing $w(x_i, 1)$ and both black vertices update for $y - 1$ merge gadgets. When updating all these vertices, then the vertex b_{x_i} from the switch gadget may remain black for some instances. However, for this vertex to remain black, we updated y vertices with $y > 1$. As a result, an optimistic update sequence σ updates exactly one vertex for each merge gadget and connector gadget.

When the corresponding 3Sat instance is a YES-instance, then there is a black vertex in the final outcome $\circ[G, \sigma]$ for each merge gadget, connector gadget, clause gadget and switch gadget. For the 3Sat reduction in Section 3.2 we use the parameter $k = n + 1$ with n being the number of variables in the 3Sat instance. We remember, in order for the Optimistic Update Sequence instance to be a YES-instance the number of black vertices in the final outcome must be at least k . However, for the constant degree reduction from 3Sat, we update the value to

$$k = |V_{2,\circ}| - \#mergeGadgets - \#connectorGadgets,$$

where $|V_{2,\circ}|$ is the number of black vertices in the Optimistic Update Sequence instance. We observe, that there are n switch gadgets and m clause gadgets, thus

$$\frac{|V_{2,\circ}| - n - m}{2} = \#mergeGadgets - \#connectorGadgets,$$

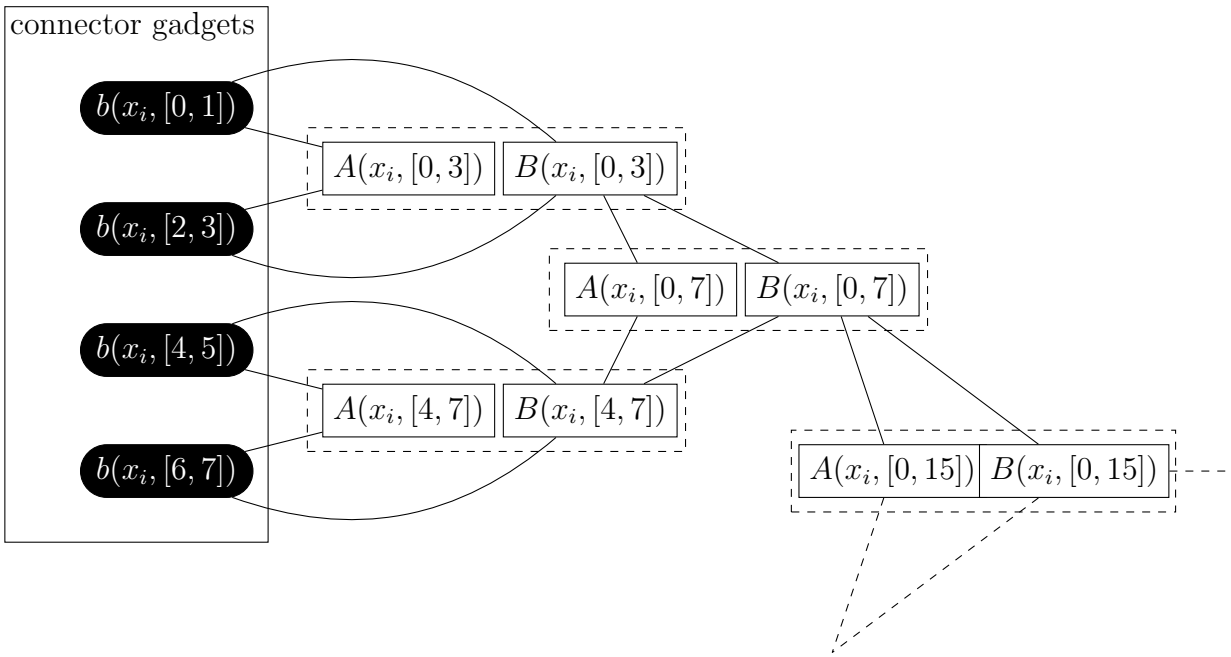


Figure 17: An example of connector gadgets combined by merge gadgets. Further, we combine the merge gadgets in a tree-like structure.

where n is the number of variables and m is the number of clauses for the 3Sat instance used to construct the Optimistic Update Sequence instance. We transform this equation to

$$k = \frac{|V_{2,\circ}| + n + m}{2}.$$

When the 3Sat instance is a YES-instance, then the Optimistic Update Sequence instance satisfies

$$|\{v \in V \mid \circ[G, \sigma](v) = q\}| \geq \frac{|V_{2,\circ}| + n + m}{2}$$

for the preferred opinion $q = 2$.

We have reduced 3Sat to Optimistic Update Sequence such that each vertex in the influence networks $(G, \circ)$ has at most seven neighbors. In the end, Optimistic Update Sequence is trivially still NP-hard for $\triangle(G) = 7$. Also, Optimistic Update Sequence is still in NP for constant degree. Altogether, Optimistic Update Sequence is NP-complete for a constant degree of seven and above. However, for the range of $\triangle(G) = 3$ up to $\triangle(G) = 6$, the complexity still remains open.

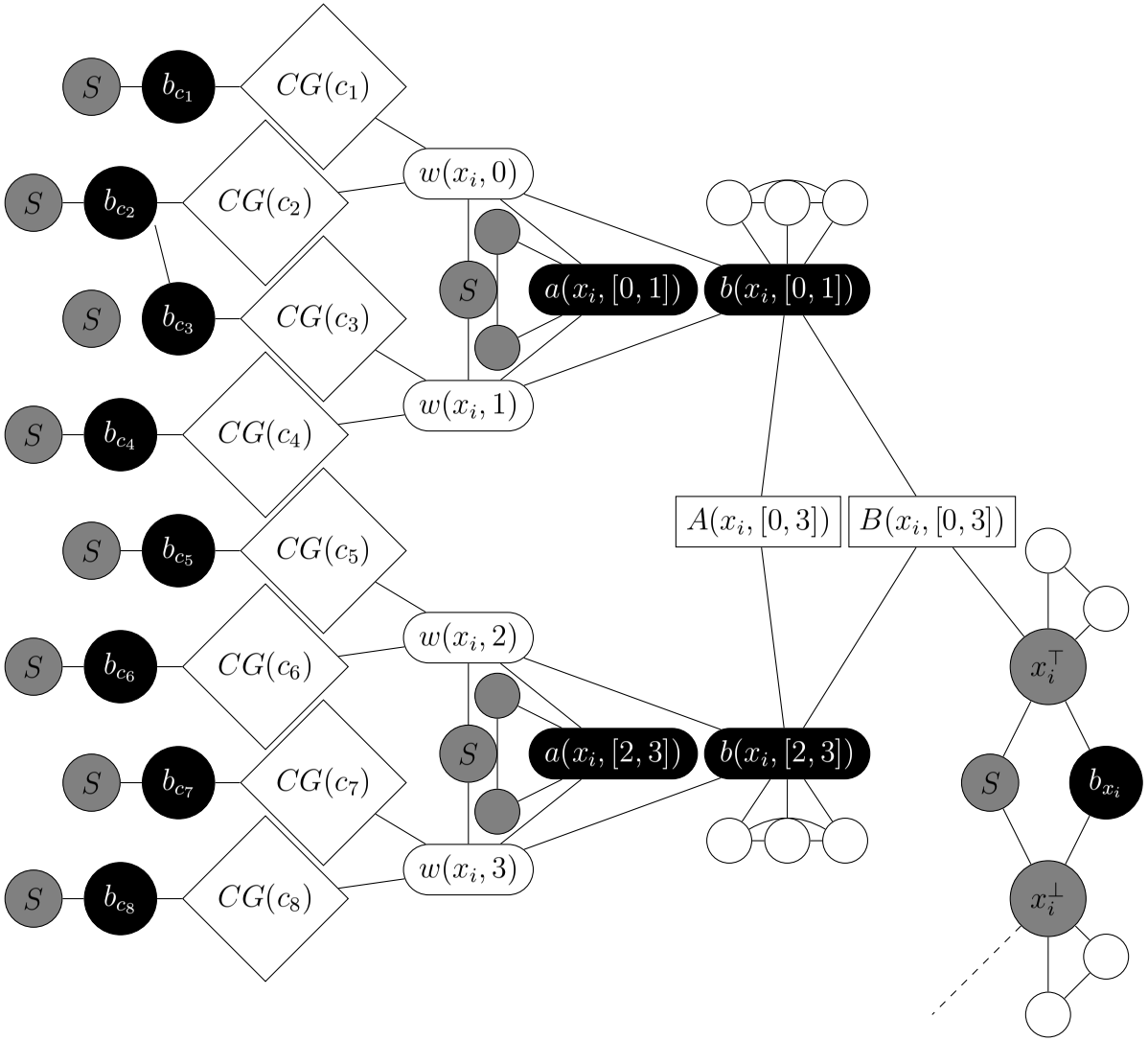


Figure 18: An example of the connection between clause gadgets, connector gadget, merge gadgets and switch gadgets for the variable x_i.

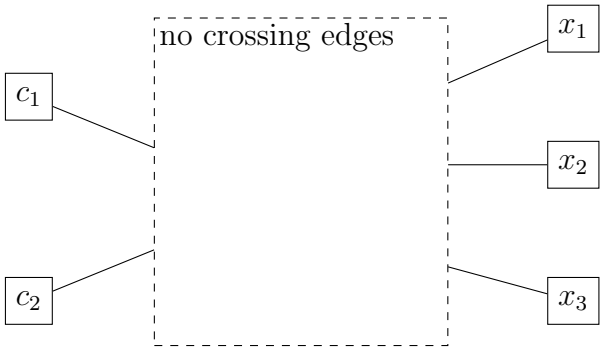
4.2 Complexity on Planar Graphs

In the previous section, we have shown that Optimistic Update Sequence is NP-hard for constant degree of at least seven. In this section, we show that Optimistic Update Sequence is still hard for the class of planar graphs. Therefore, we reduce Planar 3Sat to Optimistic Update Sequence. Particularly, we show for the reduction used in the case of constant degree, that the Optimistic Update Sequence instance has a planar embedding for all instances of Planar 3Sat.

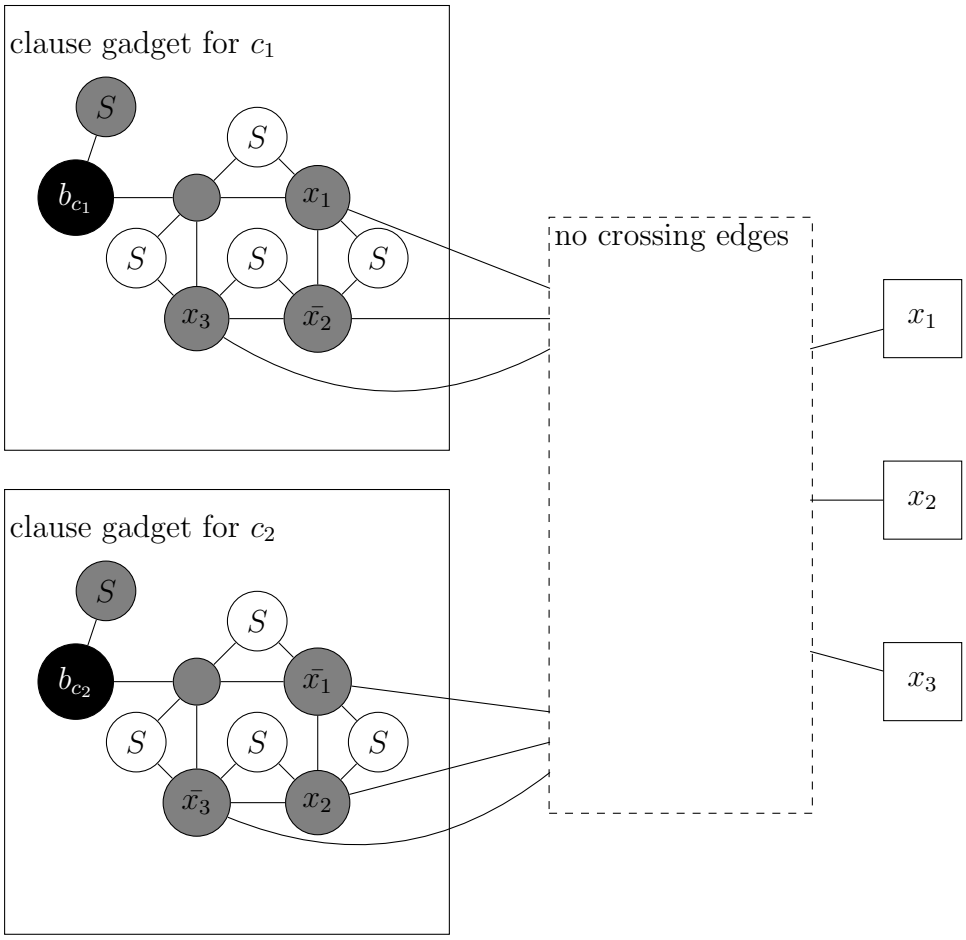
For every Planar 3Sat instance, there is a corresponding formula graph $G(\phi)$ such that $G(\phi)$ has a planar embedding. The definition of formula graphs is denoted in Section 2.3.2. The edges of the formula graph are only between clause vertices and variable vertices. Thus, the graph is bipartite. In order to show that there is a planar graph for the corresponding Optimistic Update Sequence instance, we replace the vertices of the formula graph with gadgets used in Section 4.1.2.

First, we replace the clause vertices with clause gadgets. We know, that the clause gadget has a planar embedding, thus the graph is still planar. An example for the replacement of clause vertices is shown in Figure 19. Second, we replace the variable vertices with connector gadgets. We use multiple connector gadgets to replace a single variable vertex, because each connector gadget is connected to at most four vertices from the clause gadgets. The structure of connector gadgets is described in Section 4.1.2. We note, that the connector gadget has a planar embedding. An example for a formula graph after replacing the variable vertices is shown in Figure 20. Afterwards, we use multiple merge gadgets and one switch gadget for each variable as described in Section 4.1.2. We observe, that the merge gadget has a planar embedding as well as the switch gadget. The merge gadget is described in Section 4.1.2 and the switch gadget is described in Section 4.1.2. Due to the treelike structure of merge gadgets and connector gadgets to each other, there are no edges crossing between these gadgets. Also, for each variable there is an embedding such that the edges between the switch gadget and both merge gadgets do not cross. An example for the planarity of the connection between connector gadgets and merge gadgets can be observed in Figure 17. Another example is Figure 18.

Let us assume, that there is no embedding such that the graph is planar. Therefore, there is a crossing between two edges. We have shown, that all involved gadgets have a planar embedding. Further, there is a planar embedding for the connection between connector gadgets, merge gadgets and the switch gadget for each variable. As a result, the needs to be a crossing between two of the remaining edges. However, only the edges between connector gadgets and clause gadgets are left. We did not rearrange these edges when replacing the vertices from the initial formula graph, thus these edges are planar by definition. This is a contradiction to the assumption, that there is no planar embedding for the graph. As a result, the influence network constructed by replacing the vertices of the formula graph does not violate planarity. Therefore, we have shown Planar 3Sat $\preceq_p$ Optimistic Update Sequence. In the end, Optimistic Update Sequence is still NP-complete for planar graphs.



(a) An example for a formula graph $G(\phi)$ for a PLANAR 3SAT instance with four clauses and three variables.



(b) An example for the formula graph $G(\phi)$ after replacing clause representing vertices with clause gadgets.

Figure 19: A formula graph still has a planar embedding after replacing clause representing vertices with clause gadgets.

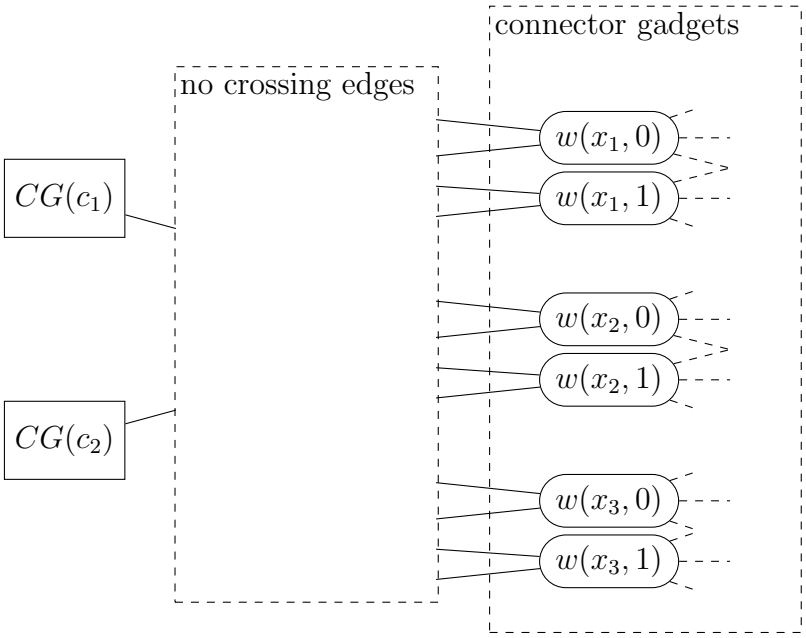


Figure 20: The formula graph $G(\phi)$ after replacing the variables with connector gadgets. Afterwards, we combine the connector gadgets with merge gadgets and finally a switch gadget for each variable. In Figure 18 we show the connection between connector gadgets, merge gadgets and switch gadgets.

4.3 Complexity on Trees

Motivated by the fact that many graph problems become computational tractable when the graph does not contain cycles (see Cygan et al. [Cyg+15]), we analyze the complexity of Optimistic Update Sequence for trees. A tree is a connected component such that this connected component contains no cycle. Briefly, we introduce the following terms: *root*, *parent*, *child* and *leaf*. We assign some vertex of the tree as *root*. In an orientated tree, this vertices is the only vertex without incoming edges. However, we just observe undirected graphs, thus the purpose of the root is for us to orientate ourself in the tree. For an arbitrary vertex v a vertex x is considered *child* of v, if both vertices are connected by an edge and the distance of v to the root is smaller then the distance of x to the root. Also, the vertex v is considered *parent* of x. A *leaf* is a vertex such that the vertex has no children.

The work of Bredereck and Elkind [BE17] shows, that for an optimistic update sequence each vertex updates at most twice for a binary set of opinions. When this property still holds for three opinions on trees then the problem may become easy. First, we need to introduce the definition of *patterns*.

Pattern

We denote the ordered list of all update steps of a specific vertex v for the update sequence σ as *pattern* of v. Further, we split the list in two parts.

- An ordered list of all opinions of v before the last update in σ, and
- the opinion of v for $[G, \sigma]$.

The pattern of v is written as $P(v) = (q_1 q_2 \mid q_3)$ with $q_1, q_2, q_3 \in O$, when the vertex has the initial opinion q_1, updates to q_2 and then updates to q_3;i.e., a black vertex v which updates to gray and then updates to white has the pattern $P(v) = (bg \mid w)$.

Counterexample

There is an example of a tree such that there is vertex with a pattern of size four. The example is shown in Figure 21. In the following, we describe the updates of the instance to show, that there are instances with at least three updates for a single vertex. We note that the program ODAT (see Section 5) contains the example as instance so that we can load the sequence as separat file for visual assistance. The vertex b_0 remains black, when the vertex v is white at the end of the update sequence. The vertex b_1 remains black, when the vertex x is gray at the end of the update sequence. The vertex b_2 remains black, when the vertex x_3 updates to white. In order to do so, the vertex x needs to update to white.

The vertex x has three white neighbors, three gray neighbors and one black neighbors. First, we update v to white due to the white majority neighborhood of v. This leads to x having a white majority neighborhood as well. Now, we update x to white. The vertex x_3

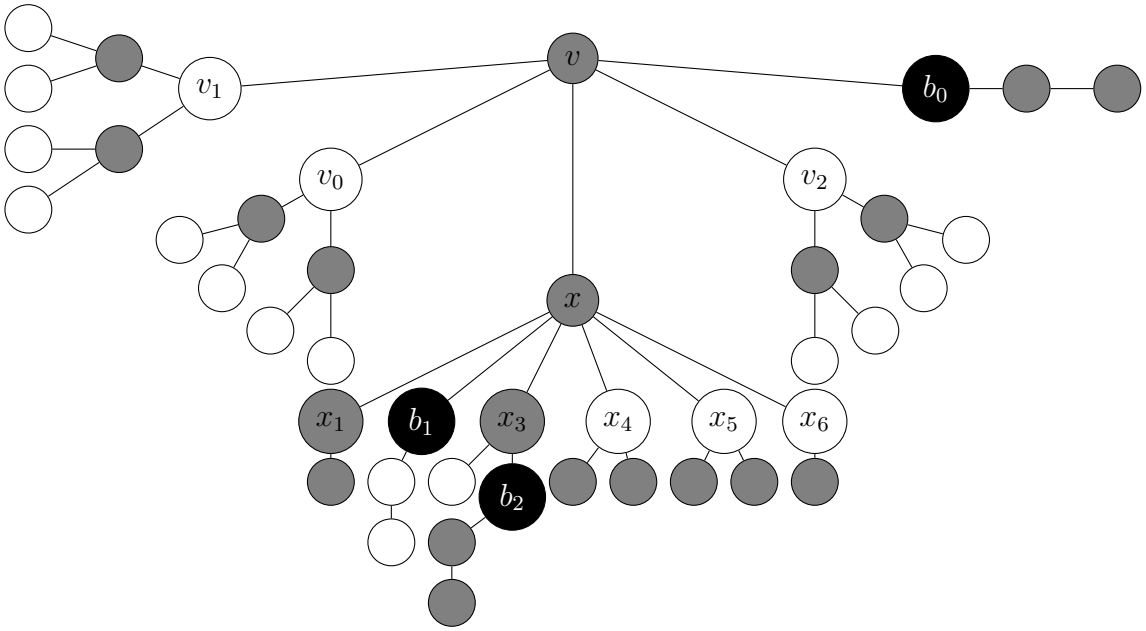


Figure 21: A counterexample to prove, that there is a tree instance such that the pattern $(gwg \mid w)$ is necessary to maximize the number of black vertices. The pattern of v is $(gwg \mid w)$ and the pattern of x is $(gw \mid g)$.

has a white majority neighborhood now, thus we update the vertex x_3. The vertex v has five white neighbors, one gray neighbor and one black neighbor now. However, we need to update x to gray for b_1 to become stable. We update the vertices x_4 and x_5 to gray. Also, we update the vertices v_0, v_1 and v_2 to gray. As a result, the vertex v has a gray majority neighborhood and we update v to gray again. We note, that the current pattern of vertex v is $(gw \mid g)$. With this, the vertex x has four gray neighbors, thus x updates to gray. As a result, the vertex b_1 is stable. The vertex x_6 updates to gray and becomes stable. In order for b_0 to remain black, the vertex v needs to update to white again. We update the children of v_0, v_1 and v_2 to the opinion white. Afterwards, we also update v_0, v_1 and v_2 to the opinion white. With this, the vertex v has a white majority neighborhood and updates to white, thus b_0 is stable. All vertices are stable and the graph contains the maximum of three black vertices.

For $|O| = 2$, each vertex updates at most twice. However, in this example the vertex v has the pattern $(gwg \mid w)$, thus has changed opinion three times. Therefore, we cannot adapt this property to Optimistic Update Sequence for a set of three opinions.

5 Opinion Diffusion Assistance Tool

The *Opinion Diffusion Assistance Tool* (ODAT) is designed for constructing influence networks and simulating update sequences. In this section, we briefly describe the major functionalities of ODAT. Further, the software also contains an operation manual for more detailed explanation and a list of all commands and interactions possible.

First, the program is started by using the following command:

java -jar ODAT.jar [-console]

When starting the program with the optional flag *-console* enabled, then a console version is started instead. However, the use of the software is mainly focused on the visual representation of update sequences and manipulation of sequences by the user. Therefore, the console version has fewer functionalities. The program comes with a few test instances. The folder *examples* contains files for influence networks. The folder *scripts* contains a few example for scripts and the folder *sequence* contains a few examples for sequences. With a script, we can define a fixed set of commands to interact with the influence network. For example, we can create a fixed set of vertices, add edges and change their opinions. The program enables us to save the sequence of the currently observed instance. Later, we can reload the sequence.

5.1 Update Sequences

The program contains algorithms to compute synchronous updating sequences, asynchronous updating sequences, balanced asynchronous updating sequences and optimistic update sequences.

Synchronous

Following the definition of synchronous update sequences in Section 2.1.2, when we use the synchronous update mode of the software, then we update all vertices of the current influence network simultaneously. The synchronous update sequence does not necessary terminate. Thus, when we observe an influence network such that a synchronous update sequence converges to a stable outcome with period two, then the program detects this. When the user uses the *fast-forward* button to skip the update sequence to a stable outcome, then the sequence stops after finding the periodic behavior. We can still use the *forward* button to alternate between the two sets of opinions.

Asynchronous

Following the definition of the asynchronous update sequences in Section 2.1.2, when we use the asynchronous update mode of the software, then we update one vertex of the current influence network at each step. In the program the update sequences only include update steps, when the corresponding vertex is unstable at the given moment.

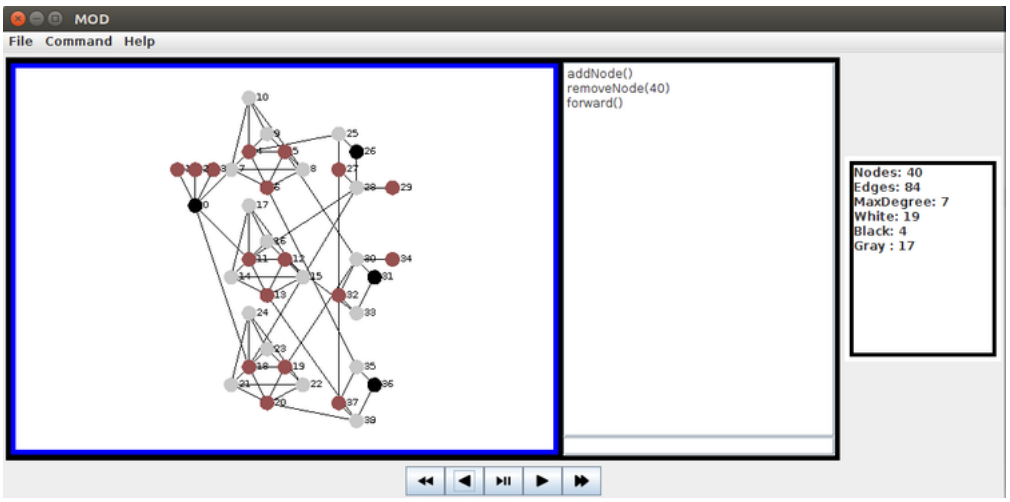


Figure 22: Screenshot of the interface of the program Opinion Diffusion Assistance Tool (ODAT).

Balanced Asynchronous

In the balanced asynchronous update mode of the program, we use the characteristic of the balanced asynchronous update sequence defined in Section 2.1.2. The program creates a list containing the ID's of all vertices and randomly shuffles the list. Afterwards, singletons of the vertices are appended to the update sequence. Doing this, the program follows the order of the shuffled list. The process terminates, when a full iteration through the list does not lead to an update step such that the opinion of any vertex changes.

Optimistic Update

When the optimistic update mode is selected, the program computes an optimistic update sequence for a preferred opinion. However, the correctness of the algorithm is limited to networks with $|O| = 2$ since we have not found an usable algorithm to compute an optimistic update sequence for three or more opinions. We can change the preferred opinion to $q \in O$ for the optimistic update mode with the command preferred_color(q).

5.2 Program Layout

In Figure 22 we can observe the four major parts of the GUI: the graph canvas, the command line, the statistic panel and the control panel.

Graph Canvas

The graph canvas shows a visual representation of the currently loaded influence network and represents the opinions with different colors.

Command Line

With the command line we can manipulate the influence network, update sequences, manually change opinions, update the position of vertices, and change the update rule and update mode. The full list of commands and results are documented in the operation manual, which can be accessed by clicking on the *Help* tab.

Statistic Panel

The small panel on the right side of the user interface provides us some information about the currently loaded influence network. We can access the number of vertices, the number of edges, and the distribution of opinions in the network.

Control Panel

With the small control panel at the bottom of the user interface we interact with the update sequence for the current network. From left to right we have five buttons: *fast-backward*, *backward*, *play/stop*, *forward*, *fast-forward*.
With the forward button we go one step of the sequence for given update rule and mode. For an influence network $(G, \circ)$ after z steps after pressing the forward button the graph canvas shows the opinion function $\circ[G, \sigma, z + 1]$. However, when pressing the backward button instead the graph canvas shows the opinion function $\circ[G, \sigma, z - 1]$. With the fast-forward button we skip to $\circ[G, \sigma, |\sigma|]$. Using the fast-backward button shows the opinion function for $\circ[G, \sigma, 0]$. When using the play/stop button we start a thread which periodically invokes the underlying function called by the forward button. While navigating through the update sequence, we can invoke commands to update the opinion of a specific vertex to observe the results of the manipulation.

6 Conclusion and Outlook

We investigated the problem of finding an update sequence that maximizes the number of vertices with the preferred opinion in an influence network for the discrete majority update rule. While the problem was known to be easy for a binary set of opinions, we proved NP-completeness for three opinions. Further, we showed that the problem is still NP-hard for planar graphs and graphs with a maximum degree of at least seven. However, for a maximum degree of at most two we can compute an optimistic update sequence in polynomial time. The computational complexity for a maximum degree for the range of three to six is still unknown.

We investigated Optimistic Update Sequence on acyclic graphs, however we could not verify whether one can find an optimistic update sequence in polynomial time. We have shown, that there are tree instances such that it is necessary for a vertex to update at least tree times in order to maximize the number of vertices with preferred opinion. In further research of Optimistic Update Sequence on trees, it is important to investigate, whether there are vertices, which need to update four times or even more often. For the example in Figure 21 we have used the subtree of v_0 including the vertex itself to update v three times. We can continue this structure to allow a vertex to update as often as desired. However, we still need to prove whether it is necessary to update the same vertex more often. Further, we need an exponential number of vertices in comparison to the number of updates for a selected vertex.

In this thesis, we mainly focused on the discrete majority update rule. However, the computational complexity of Optimistic Update Sequence for the non-discrete majority update rule is still unknown. For reference, the non-discrete majority update rule is defined in Section 2.1.1. The complexity of finding an optimistic update sequence for this update rule is still unknown. One may modify the construction in Figure 9 to show that Optimistic Update Sequence is still hard for the non-discrete majority update rule. We could change the values of the opinion black to 1 and the value of the opinion of gray to 2. The black vertices would still remain black for a equal number of white and gray vertices in the neighborhood. However, we need to modify the clause gadgets and some vertices of the variable gadgets, because a balance between gray and white vertices in the neighborhood would lead to the vertex updating to black.

We have proven NP-completeness for Optimistic Update Sequence, however we have not talked about the computational complexity of the problem. The brute-force attempt of computing the optimistic update sequence needs $\mathcal{O}(n^{|\sigma|})$ time. For the brute-force algorithm we try all combinations of choosing one vertex in each update step until the influence network is stable for a given maximum sequence size of $|\sigma|$. Whether Optimistic Update Sequence is fixed-parameter tractable is still unknown. Some interesting parameters for further research on fixed-parameter tractability are the length of the update sequence $|\sigma|$, the number of initially unstable vertices, the number of vertices n as well as the number of edges m.

A key element to construct the reduction from 3Sat to Optimistic Update Sequence

is the use of permanently stable vertices. In a social environment an agent with such behavior is one such that the agent influences the opinion of his neighbors, but does not get influenced himself. Another variation of uncommon behavior is an agent that updates to the opposite opinion of the majority of the neighborhood; an anti-agent. For a non-binary set of opinions, we need to define the opposite opinion first. For the non-discrete majority update rule, we compute the mean value of the opinions of the neighborhood. We compute the distance of the mean value to the opinion of the anti-agent. If the opinion of the anti-agent has a higher value than the mean value, then we choose the opinion closest to the value received by adding the mean value to the opinion of the anti-agent. Otherwise, we choose the opinion closest to the value received by subtracting the opinion of the anti-agent by the mean value.

Finally, we introduced asynchronous update sequences as well as synchronous update sequences. An asynchronous update sequence updates singletons. A synchronous update sequence updates all vertices simultaneously. We introduce a variation of update sequences such that we update a selected set of vertices in one update step. In a synchronous update sequence there is the possibility of two neighbors to update to the opinion of each other, effectively resulting in the swap of their opinions. When using this variation of update sequences for Optimistic Update Sequence instead, we could stabilize more black vertices for some instances by swapping the opinion of two adjacent vertices.

Literature

[BE17] R. Bredereck and E. Elkind. "Manipulating opinion diffusion in social networks". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17)*. AAAI Press. 2017, pp. 894–900 (cit. on pp. 9, 18, 46).

[BGP] S. Botan, U. Grandi, and L. Perrussel. "Multi-Issue Opinion Diffusion under Constraints". In: () (cit. on p. 10).

[CKO13] F. Chierichetti, J. Kleinberg, and S. Oren. "On discrete preferences and coordination". In: *Proceedings of the fourteenth ACM conference on Electronic commerce*. ACM. 2013, pp. 233–250 (cit. on p. 9).

[Cyg+15] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Vol. 3. Springer, 2015 (cit. on p. 46).

[Fal+18] P. Faliszewski, R. Gonen, M. Koutecký, and N. Talmon. "Opinion Diffusion and Campaigning on Society Graphs." In: *IJCAI*. 2018, pp. 219–225 (cit. on p. 9).

[GJ99] M. R. Garey and D. S. Johnson. *Computers and Intractability : a Guide to the Theory of NP-Completeness*. 1999 (cit. on pp. 15, 29).

[GO80] E. Goles and J. Olivos. "Periodic behaviour of generalized threshold functions". In: *Discrete mathematics* 30.2 (1980), pp. 187–189 (cit. on p. 9).

[Gra17] U. Grandi. "Social choice and social networks". In: *Trends in Computational Social Choice. AI Access* (2017), pp. 169–184 (cit. on p. 9).

[KZS11] Z. Katona, P. P. Zubcsek, and M. Sarvary. "Network effects and personal influences: The diffusion of an online social network". In: *Journal of marketing research* 48.3 (2011), pp. 425–443 (cit. on p. 9).

[Lic77] D. Lichtenstein. "A Technique for Proving NP-Completeness Results on Planar Graphs". PhD thesis. University of California, Berkeley, 1977 (cit. on p. 15).

[Sil16] A. Silva. "Opinion manipulation in social networks". In: *International Conference on Network Games, Control, and Optimization*. Springer. 2016, pp. 187–198 (cit. on p. 9).

[Yil+13] E. Yildiz, A. Ozdaglar, D. Acemoglu, A. Saberi, and A. Scaglione. "Binary opinion dynamics with stubborn agents". In: *ACM Transactions on Economics and Computation (TEAC)* 1.4 (2013), p. 19 (cit. on p. 9).