Bachelor Thesis

Kernelization for Degree-Constraint Editing on Directed Graphs

Fakultät IV
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Algorithmik und Komplexitätstheorie

by Marcel Koseler, 26. November 2015

Supervisors:
Vincent Froese
André Nichterlein
Rolf Niedermeier

Ich möchte meinen beiden Betreuern Vincent Froese und André Nichterlein für die zahlreichen Stunden danken, welche diese verbracht haben mit der überaus hilfreiche Betreuung während der gesamten Arbeit. Desweiteren möchte ich meinem betreuendem Professor Rolf Niedermeier danken, der während der gesamten Arbeit mehrmals Korrektur las und stets wertvolle Hinweise auf Schwächen und Verbesserungsmöglichkeiten gab.

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

Berlin, den 27.11.2015

Abstract

We study NP-hard graph modification problems on directed graphs where arcs are added to satisfy certain degree constraints. More specifically, we are studying two problems, both of which are asking whether it is possible to insert at most k arcs into an existing graph. After inserting the arcs, the out-degrees and the in-degrees of all vertices must be in a set of allowed degree values where r is the largest allowed degree. We ask whether known preprocessing results for undirected graphs [Fro+14] are also applicable for directed graphs. First, we define two variants of the problem on directed graphs and then show that this question can be answered in the affirmative for both of them. Our methods revolve around dropping the graph structure and solving the problem on the degree sequence. By doing so we can transform the graph problem into a flow network problem whose flow directly translates into the solution for the graph problem. We show that the directed problem admits a problem kernel containing $O(r^4)$ vertices which is a smaller kernel compared to the existing $O(r^5)$-vertex kernel for the undirected case. The kernel for the directed case is computable in $O(n(n+r))$ time. In the end, we also highlight the hurdles when trying to utilize our methods to reduce the kernel size for the undirected case.

Zusammenfassung

In dieser Arbeit beschäftigen wir uns mit Problemen, bei denen einem gerichteten Graphen bis zu k Kanten hinzugefügt werden. Nach dem Hinzufügen der Kanten müssen sowohl der Eingangs- als auch der Ausgangsgrad aller Knoten in einer erlaubten Menge von spezifizierten Werten liegen. Dabei wird der größte Grad durch den Parameter r limitiert. Wir beweisen, dass diese Probleme NP-schwer sind und verwenden daher Datenreduktionsregeln um die Probleme für kleine Parameterwerte effizient zu lösen. Genauer gesagt beschäftigen wir uns in dieser Arbeit damit, ob existierende Resultate für ungerichtete Graphen[Fro+14] sich auch auf gerichtete Graphen übertragen lassen. Dazu definieren wir zunächst zwei Varianten des Problemes für gerichtete Graphen. Im Laufe der Arbeit zeigen wir, dass sich die Ergebnisse für ungerichtete Graphen auf beide Varianten übertragen lassen. Die Methoden, die wir dazu verwenden, basieren darauf, dass wir zunächst die Graphstruktur ignorieren und das Problem lediglich auf der Gradsequenz lösen. Die Lösung auf der Gradsequenz können wir dann dazu verwenden, um den Ausgangsgraph in ein Flussnetzwerk zu überführen. Dieses können wir dann benutzen um eine Lösung für das Ausgangsproblem zu berechnen. Wir beweisen damit, dass für beide Varianten des gerichteten Problemes ein Problemkern existiert, welcher $O(r^4)$ Knoten beinhaltet. Das Berechnen dieses Problemkernes benötigt eine Zeit von $O(n(n+r))$. Der Problemkern ist kleiner als das äquivalente Gegenstück im ungerichteten Fall. Für diesen hatten Froese u. a. [Fro+14] bewiesen, dass ein Problemkern existiert, welcher $O(r^5)$ Knoten beinhaltet. Deswegen gehen wir am Ende darauf ein, was für Probleme auftreten, wenn man unsere Strategien für den gerichteten Fall auf den ungerichteten Fall übertragen will.

Contents

1 Introduction

In this work we study NP-hard graph modification problems on directed graphs. We are especially taking a look at the question whether it is possible to insert at most k arcs into a graph to fulfill certain degree constraints. Since the problems are NP-hard they are likely not to have an efficient solution. Accordingly, we will use a parameterized approach to achieve good running times for small input parameters. By applying data reduction rules we upper-bound the input size of the problem instance by the two main input parameters. We then use techniques from flow network theory to refine these results even further, bounding the input size only by one input parameter. The problem is motivated by the fact that many things like hyper links in the Internet (for example the PageRank algorithm views the internet as a directed graph [LM11]), street networks or scheduling dependencies can be modeled as directed graphs. It is a natural question to ask for certain degree constraints of the vertices. For example, in a street network of a city, a bus company might gain new capacities and therefore wants to add more connections. This can be expressed by adding new arcs to the network which is modeled as a directed graph (since there are one-way streets). However, the company might want to achieve that places of higher population density have more connections than places of lower population density. This can be expressed by specifying the degree constraints of the corresponding vertices in the graph. Moreover, some interesting properties of directed graphs are linked to certain degree constraints. For example, there are many different sufficient conditions so that a directed graph contains a hamiltonian cycle ([BG02], pp. 240-246). A hamiltonian cycle is a cycle in a graph that visits each vertex exactly once. Graphs containing a hamiltonian cycle are called hamiltonian graphs. Some of the conditions in which a graph is hamiltonian are only linked to certain degree constraints. These constraints can be expressed in our problem so that after solving it, the corresponding graph must contain a hamiltonian cycle. Hamiltonian graphs are of interest to any kind of company that wants to plan a route visiting each place once, like mail delivery. These companies might want to add more arcs to their network so that the resulting network is always hamiltonian, so that it is possible to plan their routes as a hamiltonian cycle.

1.1 Related Work

Now, that the initial motivation for the problem is clear, we start with formally defining the problems that are discussed in this work as well as summarizing where we started with this work. The first fundamental starting point is the work of Mathieson and Szeider [MS12]. They introduced the undirected version of the problem this work is based on:

Degree-Constraint Editing(S) (DCE(S))

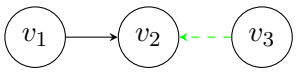
Input: An undirected graph $G = (V, E)$, two integers $k, r > 0$, and a "degree list function" $\tau\colon V \to 2^{\{0,\ldots,r\}}$.

Question: Is it possible to obtain a graph $G' = (V', E')$ from G using at most k editing operations of type(s) as specified by S such that $\deg_{G'}(v) \in \tau(v)$ for all $v \in V'$?

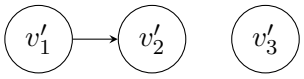
The question is whether it is possible to apply k editing operations as defined in S to satisfy the degree constraints for each vertex as defined in τ. The set of editing operations S is always a subset of the three editing operations v^- (vertex deletion), e^+ (edge addition) and e^- (edge deletion). Actually, Mathieson and Szeider [MS12] introduced the weighted version of this problem. In this work, we will focus on the non-weighted version though. Using a parameterized preprocessing framework they achieved several results, among others, showing fixed-parameter tractability for the combined parameter k and r for each $\emptyset \neq S \subseteq \{v^-, e^+, e^-\}$. However, they left open whether similar results are possible with respect to the single parameter r. This open question was answered by Froese et al. [Fro+14] in 2014. After showing that both DCE(e^-) and DCE(v^-) are not likely to have a polynomial-size kernel with respect to (k, r), they used two reduction rules to obtain a problem kernel for DCE(e^+) containing $O(kr^2)$ vertices. Additionally, they established an upper bound for k. They showed that either $k \leq r(r+1)^2$ or the problem is polynomial-time solvable. Applying their first result leads to a problem kernel for DCE(e^+) containing $O(r^5)$ vertices. Those results, however, all refer to undirected graphs. The question in this work is whether the results for DCE(e^+) from Froese et al. [Fro+14] can be transferred to directed graphs. To our best knowledge most of the works which are thematically close to ours are about undirected graphs. However, there are some other papers that are addressing arc addition problems on directed graphs. Thematically close to our work is the work of Weller et al. [Wel+12]. They ask whether on directed graphs it is possible to add or delete arcs so that the resulting graph is transitive. Similar to that is the work of Dorn et al. [Dor+13]. They are addressing the problem of inserting arcs to make a graph eulerian. Thematically not so close and about mixed graphs (which can however be modeled as directed graphs), there is also the older work of Bang-Jensen et al. [Ban+95] which is about the insertion of edges into an mixed graph to satisfy local edge-connectivity constraints. By mixed graphs we mean graphs that contain both undirected and directed edges. Very recently, Bang-Jensen et al. [Ban+15] published a paper about orienting edges in a partially oriented graph to make it an oriented graph. Orienting undirected edges may be modeled by first deleting all undirected egdes and then adding directed arcs which is what we are discussing in this paper.

1.2 Problem Definition

Now that we talked about related work, we want to start with the novel part of our work. First, one has to define the problem for a directed graph $D = (V, A)$. We consider two ways of defining a directed version of DCE, both being natural modifications of the undirected problem. The first one is the following:



	v_1	v_2	v_3
$\tau^+(v)$	{1}	{0}	{1}
$\tau^-(v)$	{0}	{1,2}	{0}



	v'_1	v'_2	v'_3
$\tau^+(v)$	{1}	{0}	{1}
$\tau^-(v)$	{0}	{2}	{1,2}

Figure 1: Two example instances of DDCE(e^+) with $k = 1$. The left one is a yes-instance and is solvable by adding the dashed arc. The right one is a no-instance.

Directed Degree-Constraint Editing(S) (DDCE(S))

Input: A directed graph $D = (V, A)$, three integers $k, r^+, r^- > 0$, and two "degree list functions" $\tau^+ \colon V \to 2^{\{0,\dots,r^+\}}$, $\tau^- \colon V \to 2^{\{0,\dots,r^-\}}$.

Question: Is it possible to obtain a graph $D' = (V', A')$ from D using at most k editing operations of type(s) as specified by S such that $\deg^+_{D'}(v) \in \tau^+(v)$ and $\deg^-_{D'}(v) \in \tau^-(v)$ holds for all $v \in V'$?

Figure 1 illustrates two example instances of DDCE(e^+). The first one is solvable by adding the arc (v_3, v_2) to the graph. The second one is a no-instance since it is not possible to satisfy the constraint $\tau^-(v'_3) = \{1\}$.
The second way of defining a directed version differs in the degree list function. Now, there is only one degree list function declaring all allowed in- and out-degree combinations rather than separate functions for both the in- and the out-degree.

Directed Tuple Degree-Constraint Editing(S) (DTDCE(S))

Input: A directed graph $D = (V, A)$, three integers $k, r^+, r^- > 0$, and a "degree list function" $\tau \colon V \to 2^{\{0,\dots,r^+\}\times\{0,\dots,r^-\}}$.

Question: Is it possible to obtain a graph $D' = (V', A')$ from D using at most k editing operations of type(s) as specified by S such that $(\deg^+_{D'}(v), \deg^-_{D'}(v)) \in \tau(v)$ for all $v \in V'$?

Although DTDCE(e^+) is a generalization of DDCE(e^+), the pleasant answer to the question of transferability mentioned above is that both DDCE(e^+) and DTDCE(e^+) are amenable to the kernelization strategies that worked for the undirected problem. Converting the problem into a maximum flow problem then makes it possible to upper-bound k which gives a problem kernel containing only $O(r^4)$ vertices (with $r := \max(r^+, r^-)$). First, we need to show NP-hardness for DDCE(e^+) and DTDCE(e^+). We then start the main part of our work by defining some basic notation and then go on to DDCE(e^+). Based on the results of [Fro+14] we will show how to get a problem kernel for DDCE(e^+) whose size is bound by the parameters k and r. We will then refine these result even further. By showing how to bound the parameter k by the parameter r we can achieve a kernel whose size is only bound by the parameter r. Afterwards we will show

that the same strategies that were used for DDCE(e^+) will also work for DTDCE(e^+). In the end we will try to give a short highlight of why our methods are probably not usable for the undirected case. We will not study any other variants than edge addition in this work, since these were already proven to not have polynomial kernels regarding r in the undirected case.

1.3 Preliminaries

In this part we will give a brief introduction into all the concepts we are using in this work as well as defining the notation we are using.

Graphs In this work, when we talk about a graph $D = (V, A)$ with $n := |V|$ and $m := |A|$, if not stated otherwise, we are always referring to a directed graph. In a graph D, $\deg_D^+(v)$ denotes the out-degree, meaning the number of outgoing arcs for a vertex $v \in V$. Analogously, $\deg_D^-(v)$ denotes the in-degree. For a set of arcs $A' \subseteq V^2$, $D + A'$ denotes the graph resulting from adding all arcs in A' to D, while $D[A']$ denotes the graph only containing the arcs in A' and the vertices incident to arcs in A'. Analogously, for a set of vertices $C \subseteq V$, $D[C]$ denotes the subgraph induced by vertices from C. Moreover, for a set of arcs $A' \subseteq V^2$ the set $V^+(A') := \{v \in V : (v, w) \in A' \text{ for a } w \in V\} \subseteq V$ refers to all vertices being a starting point of an arc in A' while $V^-(A') := \{v \in V : (w, v) \in A' \text{ for a } w \in V\} \subseteq V$ refers to all vertices being an endpoint of an arc in A'. The combined set $V(A) = V^+(A) \cup V^-(A)$ just denotes all vertices incident to an arc in A. The set $N_D^+(v) := \{w \in V : (v, w) \in A\}$ denotes all vertices $w \in V$ such that there is an arc $(v, w) \in A$. Analogously, $N_D^-(v) := \{w \in V : (w, v) \in A\}$ denotes all vertices $w \in V$, such that there is an arc $(w, v) \in A$. Furthermore, $\Delta_G^- := \max_{v \in V} \deg^-(v)$ denotes the largest in-degree and $\Delta_G^+ := \max_{v \in V} \deg^+(v)$ denotes the largest out-degree of all vertices in D.

Problem specific notation Both DDCE(e^+) and DTDCE(e^+) ask for at most k arc additions to the graph such that for each vertex v the out-degree $\deg^+(v)$ and the in-degree $\deg^-(v)$ satisfy the constraints given by τ^+ and τ^- (only τ in case of DTDCE(e^+)). The largest allowed out-degree is denoted with r^+ and the largest allowed in-degree is denoted with r^-. In that context, we define a new (meaning it is not part of an instance of DDCE(e^+) or DTDCE(e^+)) parameter $r := \max(r^+, r^-)$ as the maximum of r^+ and r^-. Additionally, for DDCE(e^+) let U^+ be the set of all vertices v with $\deg^+(v) \notin \tau^+(v)$ and analogously let U^- be the set of all vertices v with $\deg^-(v) \notin \tau^-(v)$. These vertices are called *unsatisfied* regarding the out- and in-degree, respectively. If a vertex is neither unsatisfied regarding the out-degree nor unsatisfied regarding the in-degree, then it is called *satisfied*. Finally, T_i^+ (out-type i) denotes all vertices $v \in V$ such that $(\deg^+(v) + i) \in \tau^+(v)$ and T_i^- (in-type i) denotes all vertices $v \in V$ such that $(\deg^-(v) + i) \in \tau^-(v)$.

Parameterized Complexity The concept was first described by Downey and Fellows [DF13]. Further important standard literature regarding this topic is the work of Flum and Grohe [FG06] and Niedermeier [Nie06]. Very recently, Cygan et al. [Cyg+15] published a new book about Parameterized Complexity. Parameterized Complexity is a two-dimensional framework to design algorithms for NP-hard problems. The first dimension of a parameterized instance (x, l) is the size $|(x, l)|$ of the input. The second dimension is the parameter l which can also be a combined parameter $(l_1, l_2, \dots)$. A problem (x, l) is called fixed-parameter tractable if it is solvable in $f(l) \cdot |(x, l)|^{O(1)}$ time

for some computable function f. The idea behind this is that for relatively small l, the problem becomes efficiently solvable due to the fact that $f(l)$ is also small for small l. One way of showing fixed-parameter tractability is the concept of *problem kernels*. Here, the goal is to convert an instance (x, l) into an equivalent instance (x', l') such that $l' \leq l$ and x' is bounded by $g(l)$ where g is some computable function only depending on l. The resulting instance (x', l') is called a *(problem) kernel* for the original problem instance (x, l).

Flow Networks Another concept that is used in this work is the theory of *flow networks*. A flow network is a directed graph $D = (V, A)$ where each arc can carry some amount of traffic. Formally, the following properties must apply[1]:

- Each arc a has a non-negative capacity c_a.
- There is a single source vertex $s \in V$ such that no arc enters s.
- There is a single sink vertex $t \in V$ such that no arc leaves t.

For a given flow network, one can define how much value each arc is carrying–this is called a flow. A flow is a function $f \colon A \to \mathbb{R}^+$ such that

- for each $a \in A$, we have $0 \leq f(a) \leq c_a$ and
- for each vertex v other than s and t, we have $\sum_{a \text{ into } v} f(a) = \sum_{a \text{ out of } v} f(a)$.

Each flow f has a value $v(f) \coloneqq \sum_{a \text{ out of } s} f(a)$. The basic problem concerning flow networks is the problem of finding a flow with maximum value. This problem is solvable in polynomial time, for example in $O(|V| \cdot |A|)$ [Orl13]. Also, if we have a flow f for a given network G, then we can define the *residual graph* G_f as follows:

- The node set of G_f is the same as that of G.
- For each arc $e = (u, v)$ of G on which $f(e) < c_e$, we include e in G_f with a capacity of $c_e - f(e)$. These arcs are called *forward arcs*.
- For each arc $e = (u, v)$ of G on which $f(e) > 0$ we include the arc $e' = (v, u)$ in G_f with a capacity of $f(e)$. These arcs are called *backward arcs*.

Any path from s to t in a residual graph G_f is called an augmenting path. It is a well-known fact that a flow f has maximal value if and only if there is no augmenting path in G_f ([KT06], p.348).

[1] See Kleinberg and Tardos [KT06] for a more comprehensive discussion. This definition is taken from there.

2 Computational Complexity

We will start by showing NP-hardness for both DDCE(e^+) and DTDCE(e^+). Our approach differs from the approach of Froese et al. [Fro+14]. They showed NP-hardness of DCE(e^+) for planar graphs with maximum degree three by providing a polynomial-time many-one reduction from Independent Set to DCE(e^+). We show NP-hardness of DDCE(e^+) for a directed graph D by performing a polynomial-time many-one reduction from the well known Vertex Cover problem to Directed Degree-Constraint Editing(e^+). The Vertex Cover problem is defined as follows:

Vertex Cover
Input: An undirected graph $G = (V, E)$ and an integer $k > 0$.
Question: Is it possible to choose at most k vertices $P \subseteq V$ from G such that for each edge $e = \{v, w\} \in E$ it holds that $v \in P$ or $w \in P$ (or both)?

Vertex Cover is known to be NP-hard [GJ79]. As said before, we now use Vertex Cover to show the NP-hardness of DDCE(e^+), which leads us to the following theorem:

Theorem 2.1. Directed Degree-Constraint Editing*(e^+) is NP-hard.*

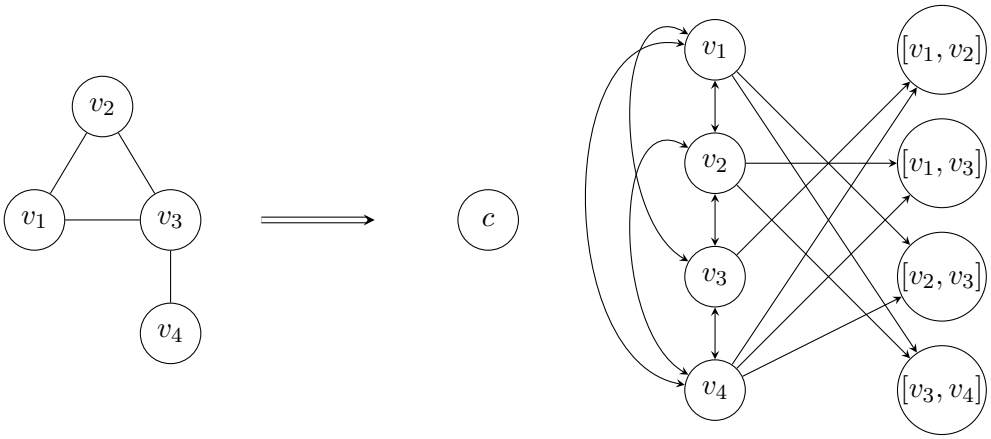
Proof. Let $G := (V, E)$ be an undirected graph and let $I = (G, k)$ be an instance of Vertex Cover with $n := |V|$ and $m := |E|$. In the following, we need to create an instance $I' = (D, k', r^+, r^-, \tau^+, \tau^-)$ of DDCE(e^+) where $D := (V', A)$ is a directed graph such that I is a yes-instance if and only if I' is a yes-instance. In Vertex Cover, we need to add at least one end point of each edge. Hence, our model has to guarantee that covering edges in I is equivalent to adding arcs in I' (since this is the only operation we have in DDCE(e^+)). The idea is to add a vertex $[v_i, v_j]$ for each edge $\{v_i, v_j\} \in E$. Each vertex $[v_i, v_j]$ has an incoming arc from all vertices but v_i and v_j. An arc $v_i \to [v_i, v_j]$ in the solution for I' models that we have v_i in the solution for I (with v_i covering the edge $\{v_i, v_j\}$). We therefore set the degree list function to $\tau^-([v_i, v_j]) = \{n-1, n\}$. Since we can only add the arc $v_i \to [v_i, v_j]$ or the arc $v_j \to [v_i, v_j]$, this assures that we add at least one end point of each edge. Furthermore, we add a count vertex c which is connected to none of the vertices with $\tau^-(c) = \{1, 2, \ldots, k\}$. This guarantees that we only add at most k vertices to our solution. See Figure 2 for an example with a schematic instance of DDCE(e^+) obtained by performing the construction described above. To be more specific, we are performing the following steps:

i) Initialize $V' := \emptyset$ and $A := \emptyset$.

ii) Add all vertices $v \in V$ to V' and set $A := (V')^2$, meaning each vertex has an outgoing arc to each other vertex.

iii) For each edge $\{v, w\} \in E$ add an edge vertex $[v, w]$ to V'. Since there are only $O(n^2)$ edges in E, this adds $O(n^2)$ vertices. Additionally, add an arc $(p, [v, w])$ to A for each vertex $p \notin \{v, w\}$.

iv) Add a count vertex c to V'.



	c	v_1	v_2	v_3	v_4	$[v_i, v_j]$
$\tau^+(v)$	{0}	{5,8}	{5,8}	{4,8}	{6,8}	{0}
$\tau^-(v)$	{1,2}	{3}	{3}	{3}	{3}	$\{3, 4\}$

Figure 2: Schematic reduction from Vertex Cover to DDCE(e^+). The left side shows an instance of Vertex Cover with $k = 2$. The right side shows the resulting instance of DDCE(e^+) after performing the steps described in the proof of Theorem 2.1. The constraints are listed in the table below. The count vertex c makes sure that we do not add more than k vertices. Also, we add an edge vertex $[v_i, v_j]$ for each edge $\{v_i, v_j\}$. Adding an arc $v_i \to [v_i, v_j]$ models that we cover the edge $\{v_i, v_j\}$. The constraint $\tau^-([v_i, v_j])$ is chosen so that we need to add at least one incident arc.

v) Set $k' \coloneqq k+2 \cdot m$ such that k' is large enough to add all possible arcs. Furthermore, set $r^+ \coloneqq n + m$ and $r^- \coloneqq n$.

vi) Set $\tau^+(v) \coloneqq \{\deg_D^+(v), n + m\}$ and $\tau^-(v) \coloneqq \{n - 1\}$ for each $v \in V \cap V'$.

vii) Set $\tau^+([v, w]) \coloneqq \{0\}$ and $\tau^-([v, w]) \coloneqq \{n - 1, n\}$ for each edge vertex $[v, w] \in V' \setminus (V \cup \{c\})$.

viii) Set $\tau^+(c) \coloneqq \{0\}$ and $\tau^-(c) \coloneqq \{1, 2, \ldots, k\}$.

Since V' contains $O(n^2)$ vertices it is possible to perform all steps in polynomial time. It remains to show that I is a yes-instance if and only if I' is a yes-instance.
Assume that I is a yes-instance of Vertex Cover. Hence, there is a set $P \subseteq V$ such that $|P| \leq k$ and for each edge $\{v, w\} \in E$ it holds that $v \in P$ or $w \in P$ (or both). Let $\widetilde{A} \coloneqq \emptyset$ be the set of arcs we are adding to the directed graph D. We will first add an arc (v, c) to $\widetilde{A}$ for each $v \in P$. Because P contains at most k vertices it holds that after these edge additions the constraints for c as specified by $\tau^+(c)$ and $\tau^-(c)$ are satisfied. Notice that in the beginning the constraints for each vertex $v \in V \cap V'$ are satisfied and

we have $\deg_D^+(v) = n - 1 + m - \deg_G(v)$ and $\deg_D^-(v) = n - 1$. Consequently, out of the vertices in $V \cap V'$ we only have to satisfy the constraints for the vertices in P. Since for each vertex $v \in P$, we add an arc to c, it follows from the construction of $\tau^+(v)$ that I' is a yes-instance only if $\deg^+_{D+\widetilde{A}}(v) = n + m$. We therefore add all arcs $(v, [v, w])$ to $\widetilde{A}$ where $v \in P$. Together with the one arc to c, we add exactly $\deg_G(v) + 1$ arcs for each $v \in P$. Accordingly, after these arc additions we have $\deg^+_{D+\widetilde{A}}(v) = (n+m) \in \tau^+(v)$ for each $v \in P$. After performing all arc additions we just described, we also have at least one arc to each vertex $[v, w] \in V' \setminus (V \cup \{c\})$: Since P is a vertex cover and $\{v, w\} \in E$, it holds that $v \in P$ or $w \in P$. We therefore added an arc from v to $[v, w]$ or from w to $[v, w]$ (or both). Thus, it holds that $\deg^-_{D+\widetilde{A}}([v, w]) \in \tau^-([v, w])$ because $\deg^-_D([v, w]) = n - 2$ by construction. Hence, I' is a yes-instance.
Now, assume that I' is a yes-instance. Hence, there exists a set of arcs $\widetilde{A}$ so that adding all arcs in $\widetilde{A}$ solves I'. Define $P \subseteq V$ to be the set of vertices such that for each $v \in P$ there is an arc (v, c) in $\widetilde{A}$. By construction of $\tau^-(c)$ it follows that $|P| \leq k$. For each arc $(v, [v, w])$ in $\widetilde{A}$ it holds that $v \in P$ because a vertex that has no arc to c cannot get an out-degree of $n + m$ and can therefore not be incident to any arc added to the graph. Finally, it holds that the vertices in P cover all edges in E because I' is a yes-instance and as a consequence each vertex $[v, w] \in V' \setminus (V \cup \{c\})$ is incident to at least one added arc in $\widetilde{A}$. Thus, P is a vertex cover and, accordingly, I is a yes-instance. □

This result directly gives us the NP-hardness of DTDCE(e^+) as well, since it is a generalization of DDCE(e^+):

Theorem 2.2. DTDCE(e^+) *is NP-hard.*

Proof. Let $I = ((V, A), k, r^+, r^-, \tau^+, \tau^-)$ be an instance of DDCE(e^+). We define $I' \coloneqq ((V, A), k, r^+, r^-, \tau)$ with $\tau(v) \coloneqq \tau^+(v) \times \tau^-(v)$ for each $v \in V$. Assume that I is a yes-instance. Thus, we can add at most k arcs to A such that after those edge additions it is true that $\deg^+(v) \in \tau^+(v)$ and $\deg^-(v) \in \tau^-(v)$ for each $v \in V$. Due to the fact that we defined τ as the Cartesian product of τ^+ and τ^-, it is also true that adding the same arcs in I' leads to $(\deg^+(v), \deg^-(v)) \in \tau(v)$ for each $v \in V$. Thus, I' is a yes-instance. Analogously, assume that I' is a yes-instance. Hence, we can again add at most k arcs such that $(\deg^+(v), \deg^-(v)) \in \tau(v)$ for each $v \in V$. By definition of τ it follows that adding the same arcs in I results in both $\deg^+(v) \in \tau^+(v)$ and $\deg^-(v) \in \tau^-(v)$ for each $v \in V$. Hence, I is a yes-instance. We provided a polynomial-time many-one reduction from DDCE(e^+) to DTDCE(e^+) and since DDCE(e^+) is NP-hard (Theorem 2.1) it follows that DTDCE(e^+) is NP-hard as well. □

3 Problem kernels for Directed Degree-Constraint Editing(e^+)

Every instance $I = (D, k, r^+, r^-, \tau^+, \tau^-)$ of DDCE(e^+) contains three natural parameters which are usable for kernelization: the maximal amount of arc additions k, the largest allowed out-degree r^+, and the largest allowed in-degree r^-. For purposes of comparing our results to the undirected case, we will–as mentioned in the part about graphs of Section 1.3–define $r := \max(r^+, r^-)$. Hence, we will use the parameters k and r to construct kernels for DDCE(e^+). The final goal of this section will then be to find a problem kernel for DDCE(e^+) with respect to the single parameter r. To achieve this, we first take an intermediate step and construct a problem kernel with respect to the combined parameter (k, r).

3.1 A problem kernel with respect to k and r

This first part is based on the ideas of Froese et al. [Fro+14] which we adapt to fit into the context of directed graphs. They observed that, if one wants to add edges to a graph, it does not exactly matter which vertices become incident to these edges. It is only important that the vertices behave similarly in the way that adding the same number i of edges leads to the same result of the vertices being satisfied. This means that for vertices v_1, v_2 for which both $(\deg(v_1) + i) \in \tau(v_1)$ and $(\deg(v_2) + i) \in \tau(v_2)$, it does not matter which vertex is used for adding these i edges. This concept was introduced as the type of a vertex. Froese et al. [Fro+14] defined a vertex v to have *type* i if $(\deg(v) + i) \in \tau(v)$, meaning that adding i edges incident to v results in v being satisfied. They have proven that only a certain amount of vertices from each type is needed to solve DCE(e^+). The same concept, with slight adjustments, still works for the directed case. Now, we have to define two different kinds of types: in-types and out-types, as defined earlier in the paragraph about graphs from Section 1.3. Basically, a vertex v is of *in-type* i if it allows for i arc additions $(w_1, v), \ldots, (w_i, v)$ to A with the result of v being satisfied regarding the in-degree. Analogously, a vertex v is of *out-type* i if it allows for i arc additions $(v, w_1), \ldots, (v, w_i)$ to A with the result of v being satisfied regarding the out-degree. Consequently, all that is left is to figure out how many vertices of each in- and out-type are needed for DDCE(e^+). The answer is quite similar to the undirected case, which brings us to the following construction:

Construction 3.1. *Define $\alpha^- := k \cdot (\Delta_D^+ + 1)$ and $\alpha^+ := k \cdot (\Delta_D^- + 1)$ where Δ_D^+ is the maximum out-degree and Δ_D^- is the maximum in-degree. Then, build a subset $C \subseteq V$ of vertices:*

- *Add all unsatisfied vertices U^+ and U^- to C.*
- *Add $\min(\alpha^-, |T_i^- \setminus U^-|)$ satisfied vertices regarding the in-degree from each in-type $i \in \{1, \ldots, r^-\}$ to C where T_i^- are all vertices having in-type i.*
- *Add $\min(\alpha^+, |T_j^+ \setminus U^+|)$ satisfied vertices regarding the out-degree from each out-type $j \in \{1, \ldots, r^+\}$ to C where T_j^+ are all vertices having out-type j.*

Let $I = (D, k, r^+, r^-, \tau^+, \tau^-)$ be an instance of DDCE(e^+). Removing from V all vertices $v \in V \setminus C$ that are not in C results in a new instance $I' = (D[C], k, r^+, r^-, \tau_C^+, \tau_C^-)$. The idea behind this construction is that in order to solve DDCE(e^+), we need at least all unsatisfied vertices. Furthermore, we need enough vertices to satisfy the constraints of the unsatisfied vertices. It turns out that enough vertices are exactly α^- vertices of each in-type and α^+ vertices of each out-type. Later in the proof, the idea will then be to swap each vertex not in C with two other vertices. Figure 3 illustrates this situation and shows why α^- and α^+ are chosen the way they are in Construction 3.1. After removing the vertices not in C, in order to maintain formal correctness, we have to adjust the degree list functions. To this end, we define for each $v \in C$:

$$\tau_C^+(v) \coloneqq \{k \in \mathbb{N} : k + |N_D^+(v) \cap V \setminus C| \in \tau^+(v)\} \tag{1}$$
$$\tau_C^-(v) \coloneqq \{k \in \mathbb{N} : k + |N_D^-(v) \cap V \setminus C| \in \tau^-(v)\} \tag{2}$$

Basically, the removed vertices force us to adjust $\tau^+(v)$ and $\tau^-(v)$ for each $v \in V$. Let $v \in V$ be a vertex from V. By removing adjacent vertices from D, the in-degree of v changes. Hence, the gap between the in-degree $\deg^-(v)$ and the degree constraint $\tau^-(v)$ gets bigger. Thus, each entry in $\tau^-(v)$ gets reduced by the amount of adjacent vertices deleted from the graph. The amount equals $|N_D^-(v) \cap V \setminus C|$, because this describes all vertices selected for removal which also have an arc to v. The adjustment for $\tau^+(v)$ works analogously. By doing all this, we get an equivalent instance of DDCE(e^+):

Reduction Rule 3.1. *Let C be a set of vertices as described in Construction 3.1. Then, remove all vertices in $V \setminus C$.*

This reduction rule is correct and can be applied in $O(n(n+r))$ time:

Lemma 3.2. *Reduction Rule 3.1 is correct and is executable in $O(n(n+r))$ time.*

Proof. Let $I = (D, k, r^+, r^-, \tau^+, \tau^-)$ be an instance of DDCE(e^+) and let $I' = (D[C], k, r^+, r^-, \tau_C^+, \tau_C^-)$ be the instance obtained by applying Reduction Rule 3.1. We need to prove that I is a yes-instance if and only if I' is a yes-instance. Since $V \setminus C$ only contains vertices being satisfied regarding both the in-degree and the out-degree, one can easily see that each solution for I' also is a solution for I.
It remains to show that the reverse direction is true as well. Let $E' \subseteq V^2 \setminus E$ be a solution for I. Note that if $V(E') \subseteq C$, then E' already is a solution for I'. Accordingly, assume that $V(E') \setminus C \neq \emptyset$ and let $v^\star \in V(E') \setminus C$. Furthermore, let i^- be the amount of arcs in E' that end in $v^\star$ and let i^+ be the amount of arcs that start in $v^\star$. Because $v^\star \notin C$ it holds that $|C \cap T_{i^-}^-| = \alpha^-$ and $|C \cap T_{i^+}^+| = \alpha^+$. Next, we show that $v^\star$ can be exchanged by two vertices $w^-, w^+ \in C$. We want w^- to have the same in-type and w^+ to have the same out-type as v (meaning $w^- \in C \cap T_{i^-}^-$ and $w^+ \in C \cap T_{i^+}^+$). This results in a new solution E'' such that $|V(E'') \setminus C| = |V(E') \setminus C| - 1$. Thus, it remains to prove the existence of such vertices. Observe that, in order to exchange v, two things must be fulfilled: First, w^- and w^+ cannot already be incident to arcs added in E' $(w^- \notin V^-(E'), w^+ \notin V^+(E'))$. Second, these vertices cannot be adjacent to any

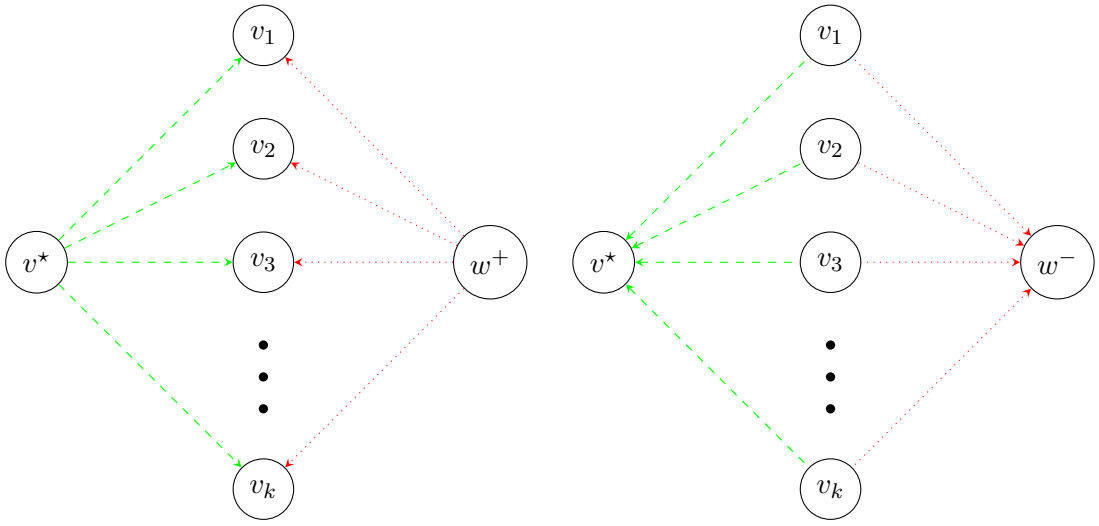


Figure 3: An Illustration of the situation when a vertex $v^\star \notin C$ (as defined in Construction 3.1) exists which shall get swapped for vertices $w^+ \in C$ (left side) and $w^- \in C$ (right side). The k dashed arcs are getting deleted while the dotted arcs are added (and are therefore not allowed to be in the graph). Each vertex v_i has at most Δ_G^- incoming and Δ_G^+ outgoing arcs. Also, w^+ and w^- are not allowed to already be incident to any of the k added arcs. To always find these vertices w^+ and w^- we choose α^+ and α^- large enough, that is, $\alpha^+ = k \cdot (\Delta_G^- + 1)$ and $\alpha^- = k \cdot (\Delta_G^+ + 1)$.

vertex adjacent to $v^\star$ (see Figure 3 for an illustration). More precisely, the following must apply: $N^-_{D[E']}(v^\star) \cap N^-_D(w^-) = \emptyset$ and $N^+_{D[E']}(v^\star) \cap N^+_D(w^+) = \emptyset$. However, because

$$|V^-(E') \cup (N^-_{D[E']}(v^\star) \cap N^-_D(w^-))| \leq k + k \cdot \Delta_G^+ \tag{3}$$

and

$$|V^+(E') \cup (N^+_{D[E']}(v^\star) \cap N^+_D(w^+))| \leq k + k \cdot \Delta_G^- \tag{4}$$

such vertices w^- and w^+ can always be found. By iteratively applying this procedure we obtain a solution only containing vertices from C, and therefore a solution for I'. This procedure obviously terminates since in each step $|V(E') \setminus C|$ gets reduced by one. To compute the vertices for our solution C one does the following. First, one computes the in- and out-degree of each vertex which takes $O(n^2)$ time. Computing the types of each vertex then takes $O(n \cdot r)$ time by simply checking all r in- and out-types for each $v \in V$, with type 0 representing already satisfied vertices. Then, one initializes a counter $i_j := 0$ for each in-type $j \in \{1, \ldots, r^-\}$ as well as a counter $o_j := 0$ for each out-type $j \in \{1, \ldots, r^+\}$. Moreover, one sets the target set $C := \emptyset$. Finally, one parses V and checks for each vertex $v \in V$ with the set of its in-degrees $I_v \subseteq \{0, \ldots, r^-\}$ and the set of its out-degrees $O_v \subseteq \{0, \ldots, r^+\}$ the following:

i) Does v have in-type 0? If not, then add v to C since it is unsatisfied regarding the in-degree, and go on to the next vertex.

ii) Does v have out-type 0? If not, then add v to C since it is unsatisfied regarding the out-degree, and go on to the next vertex..

iii) Does v have an in-type $j \in I_v \setminus \{0\}$ such that $i_j < \alpha^-$? If so, then add v to C and increment i_m by one for each $m \in I_v \setminus \{0\}$ and o_p by one for each $p \in O_v \setminus \{0\}$. Then, go on to the next vertex.

iv) Does v have an out-type $j \in O_v \setminus \{0\}$ such that $o_j < \alpha^+$? If so, then add v to C and increment i_m by one for each $m \in I_v \setminus \{0\}$ and o_p by one for each $p \in O_v \setminus \{0\}$. Then, go on to the next vertex.

This procedure takes $O(n \cdot r)$ time since we have to check at most $r^- \leq r$ in-types and at most $r^+ \leq r$ out-types for all n vertices. Doing all this yields an overall running time of $O(n(n+r))$. □

After applying Reduction Rule 3.1 there are at most α^- satisfied vertices of each in-type and α^+ satisfied vertices of each out-type in C. Since there are only r^- in-types and r^+ out-types, we can use this fact to estimate the size of C by using Construction 3.1:

$$|C| \leq |U^-|+|U^+|+r^- \cdot \alpha^- + r^+ \cdot \alpha^+ \leq |U^-|+|U^+|+r^- \cdot k \cdot (\Delta_D^+ + 1) + r^+ \cdot k \cdot (\Delta_D^- + 1) \quad (5)$$

Observe that if there are more than k unsatisfied vertices regarding the in-degree or more than k unsatisfied vertices regarding the out-degree, then we can just return a trivial no-instance. Furthermore, we can also upper-bound the maximum in-degree to be at most the largest allowed in-degree r^-. Analogously, we can upper-bound the maximum out-degree to be at most the largest allowed out-degree r^+. If there exists a vertex with a degree higher than this, then we can again just return a trivial no-instance. This leads to:

Reduction Rule 3.2. *If an instance of* DDCE(e^+) *contains more than k unsatisfied vertices regarding the in-degree or more than k unsatisfied vertices regarding the out-degree, then return a trivial no-instance. Additionally, if there exists a vertex v with $deg^-(v) > r^-$ or with $deg^+(v) > r^+$, then also return a trivial no-instance.*

This reduction rule is applicable in $O(n)$ time, since all we have to do is to parse all n vertices and check for each vertex in constant time the three properties mentioned above. After applying Reduction Rule 3.2 we can upper-bound the size of both U^+ and U^- by k as well as upper-bound the maximum out-degree Δ_D^+ by r^+ and upper-bound the maximum in-degree Δ_D^- by r^-. Together with $r := \max(r^+, r^-)$ we can extend Inequality (5) to

$$|C| \leq 2k + r \cdot k \cdot (r+1) + r \cdot k \cdot (r+1) \quad (6)$$

leading to a problem kernel consisting of $O(k \cdot r^2)$ vertices. Applying both Reduction Rule 3.1 and Reduction Rule 3.2 leads to an overall running time of $O(n(n+r))$.

Theorem 3.3. DDCE(e^+) *admits a problem kernel containing $O(k \cdot r^2)$ vertices. It is computable in $O(n(n+r))$ time.*

3.2 A problem kernel with respect to r

In the previous section we saw that, by using our reduction rules, we could get a problem kernel utilizing both parameters: the maximal amount of arc additions k and $r := \max(r^+, r^-)$, where r^+ is the largest allowed out-degree and r^- is the largest allowed in-degree. This problem kernel contains $O(kr^2)$ vertices. The goal in this section is to get a problem kernel whose size only depends on the single parameter r. We will show that for large k, the problem becomes polynomial-time solvable by reducing it to a maximum flow problem. The strategy used will be somewhat similar to the strategy used by Froese et al. [Fro+14]. We start with reducing the complexity of the problem by dropping the graph structure and solving the easier Number Constraint Editing problem. This will give us in-demands and out-demands for each vertex, that is, the number of incoming and the number of outgoing arcs in a solution of DDCE(e^+), respectively. That part is analogous to undirected graphs. The next part is where the strategies begin to differ. Froese et al. [Fro+14] used a problem called f-Factor which gave them the realizability of their calculated demands if $k > r(r+1)^2$. That problem however, does not exist for directed graphs. For this reason, we need to come up with something else. Accordingly, we use the demands calculated with Number Constraint Editing to build a flow network. We then show that either $k \leq 2r^2 + r$ (which is an improvement compared to $k \leq r(r+1)^2$ from the undirected case) or there is a flow fulfilling the demands of each vertex. Let us start by defining Number Constraint Editing:

Number Constraint Editing (NCE)

Input: A function $\tau\colon \{1,\ldots,n\} \to 2^{\{0,\ldots,r\}}$ and non-negative integers $d_1 \ldots, d_n, k, r$.

Question: Are there n integers $d'_1, \ldots, d'_n$ such that $\sum_{i=1}^{n}(d'_i - d_i) = k$ and for all $i = 1, \ldots, n$ it holds that $d'_i \geq d_i$ and $d'_i \in \tau(i)$?

One can think about each input number d_i as the in-degree (out-degree) of one vertex, whereas each output-number d'_i represents the in-degree (out-degree) of one vertex after k arc additions. Since each arc addition increases the in-degree (out-degree) of exactly one vertex by one and we are adding k arcs, it is obvious why the overall difference $\sum_{i=1}^{n}(d'_i - d_i)$ has to be exactly k. By using a dynamic programming algorithm due to Froese et al. [Fro+14], this problem can be solved in $O(n \cdot k \cdot r)$ time. Solving this problem for both the list of in-degrees and the list of out-degrees from D will result in two numbers for each vertex, representing the number of incoming, respectively, outgoing arcs in a solution of DDCE(e^+):

$$(\deg^-(v_1), \ldots, \deg^-(v_n)) \xrightarrow{\text{NCE}(\tau^-,k,r^-)} (d_1^-, \ldots, d_n^-)$$

$$(\deg^+(v_1), \ldots, \deg^+(v_n)) \xrightarrow{\text{NCE}(\tau^+,k,r^+)} (d_1^+, \ldots, d_n^+)$$

Note that for each vertex v_i, the difference $d_i^- - \deg^-(v_i)$ gives us an in-demand for v_i, meaning the amount of arcs (w, v_i) ending in v_i we need to add. Analogously, $d_i^+ - \deg^+(v_i)$ gives us an out-demand for v_i. Hence, the remaining problem is to decide

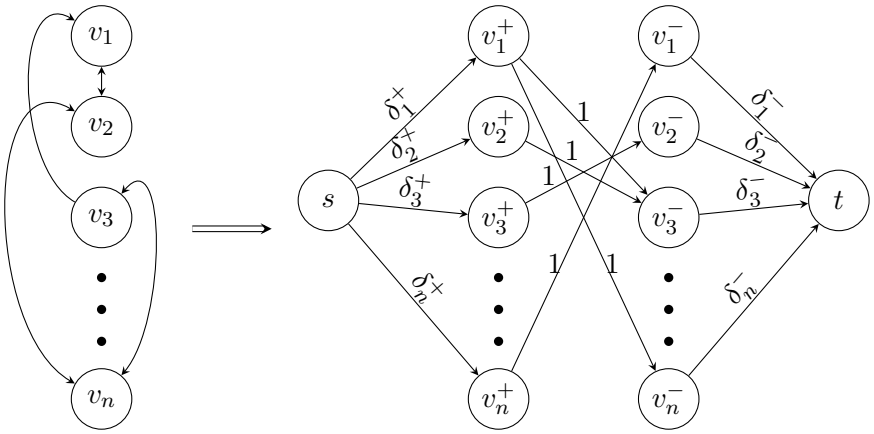


Figure 4: A schematic flow network as described in Construction 3.4. For each vertex v_i in the graph D there are two vertices added to the flow network: v_i^+ and v_i^-. We connect a vertex v_i^+ to a vertex v_j^- if the arc (v_i, v_j) is not in D. Adding an arc (v_i, v_j) is represented by setting the flow on the arc (v_i^+, v_j^-) to one. The capacities on the arcs (s, v_i^+) and (v_j^-, t) come from solving Number Constraint Editing.

whether it is possible to realize the demands inside the given graph. The idea will be to use those in- and out-demands to construct a flow network as follows:

Construction 3.4. *Let $D = (\{v_1, \dots, v_n\}, A)$ be the directed graph we are working on.*

- *For each vertex $v_i \in \{v_1, \dots, v_n\}$, add two vertices to the network: v_i^+ and v_i^-.*
- *Add a source-vertex s and a sink-vertex t to the network.*
- *Add an arc (s, v_i^+) with capacity $\delta^+ := d_i^+ - \deg^+(v_i)$ for each $i \in \{1, \dots, n\}$.*
- *Add an arc (v_i^-, t) with capacity $\delta^- := d_i^- - \deg^-(v_i)$ for each $i \in \{1, \dots, n\}$.*
- *Add an arc (v_i^+, v_j^-) with capacity one if $(v_i, v_j) \notin E$ for each $i, j \in \{1, \dots, n\}, i \neq j$.*

See Figure 4 for a schematic flow network obtained by Construction 3.4. Adding an arc (v, w) to the graph D corresponds to sending flow from v^+ to w^-. Since by definition each vertex v_i^+ will only receive at most δ^+ flow from s and each vertex v_i^- will send at most δ^- flow to t, we can not add too many arcs. Thus, the goal will be to prove that the maximum flow in the network is indeed k.
If this is the case, then we are able to transfer the demands we obtained from solving Number Constraint Editing to the graph problem. Hence, we only need to examine in which situation the flow in the network equals k:

Lemma 3.5. *If $k > 2r^2 + r$, then there always is a maximum flow of value k in the network $N = (V_N, A_N)$ obtained by Construction 3.4.*

Proof. Let $L \coloneqq \{v_i^+ \in V_N : i \in \{1,\dots,n\}\}$ be the vertices on the left side of N and let $R \coloneqq \{v_i^+ \in V_N : i \in \{1,\dots,n\}\}$ be the vertices on the right side of N. In the following, a vertex $v_i^+ \in L$ ($v_j^- \in R$) is called satisfied regarding a flow f, if $f(s, v_i^+) = \delta_i^+$ ($f(v_j^-, t) = \delta_j^-$). Let $k \in \mathbb{N}, k > 2r^2 + r$. Suppose there is a maximum flow $f : A_N \to \mathbb{R}^+$ with value $v(f) < k$. Then, it is fact that there are unsatisfied vertices $v_i^+ \in L$ and $v_j^- \in R$. Let $X \subset V_N$ be the vertices to which v_i^+ has a forward arc and let $Y \subset V_N$ be the vertices which have a forward arc to v_j^- in the residual graph G_f. Observe that the amount of outgoing arcs for the vertex v_i^+ equals $n-1-\deg^+(v_i)$ (by construction we only add arcs $(v_i^+, v_j^-) \in V_N^2$ to the network N if the arc $(v_i, v_j) \in V^2$ is not in the graph D). Analogously, the amount of incoming arcs for the vertex v_j^- equals $n - 1 - \deg^-(v_j)$. Consequently, there are at least

$$n - 1 - \deg^+(v_i) - \delta_i^+ = n - 1 - \deg^+(v_i) - (d_i^+ - \deg^+(v_i)) = n - 1 - d_i^+ \geq n - 1 - r$$

forward arcs from v_i^+ to vertices in R in the residual graph G_f. Therefore, $|X| \geq n-r-1$. Since we know that v_i^+ is unsatisfied, we can even say that $|X| > n-r-1$. By the same reasoning it follows that $|Y| > n - r - 1$. The residual graph is illustrated in Figure 5. Remember that f is a flow of maximum value. Hence, we know that each vertex in X and each vertex in Y is satisfied. Otherwise, there would be an augmenting path in the residual graph, contradicting our assumption of f being maximal. Now, if a vertex in X would receive flow from a vertex in Y, this would give rise to a backward arc in the residual graph resulting in an augmenting path $s \to v_i^+ \to X \to Y \to v_j^- \to t$, again contradicting our maximum assumption for f. As a consequence, we can conclude that all flow that goes into X has to come from the at most r remaining vertices in L (not counting v_i^+). But since $d_p^+ - \deg^+(v_p) \leq r$ for each $p \in \{1,\dots,n\}$ those at most r vertices can cover at most flow of value r^2. Remember that the demands come from solving NCE, implying that $k = \sum_{i=1}^{n}(d_i^- - \deg^-(v_i))$. Thus, we have:

$$\begin{aligned} k &= \sum_{i=1}^{n}(d_i^- - \deg^-(v_i)) \\ &= \sum_{w_i \in R}(d_i^- - \deg^-(v_i)) \; + \sum_{w_i \in R \setminus Y}(d_i^- - \deg^-(v_i)) \\ &\leq r^2 + (r+1) \cdot r = 2r^2 + r \end{aligned} \tag{7}$$

This contradicts our assumption $k > 2r^2 + r$. Hence, the initial assumption must be false, proving the claim. □

With Lemma 3.5 we have everything we need to prove that for $k > r^2 + r$ it is sufficient to compute a solution for the polynomial-time solvable Number Constraint Editing. This results in the following:

Lemma 3.6. *Let* $I = (\{v_1,\dots,v_n\}, A), k, r^+, r^-, \tau^+, \tau^-)$ *be an instance of* DDCE(e^+) *with* $k > r^2 + r$. *If there exists a* $k' \in \{r^2 + r + 1,\dots,k\}$ *such that it holds that* $((\deg^+(v_1),\dots,\deg^+(v_n)), k', r^+, \tau^+)$ *as well as* $((\deg^-(v_1),\dots,\deg^-(v_n)), k', r^-, \tau^-)$ *are yes-instances of* NCE, *then* I *also is a yes-instance of* DDCE(e^+).

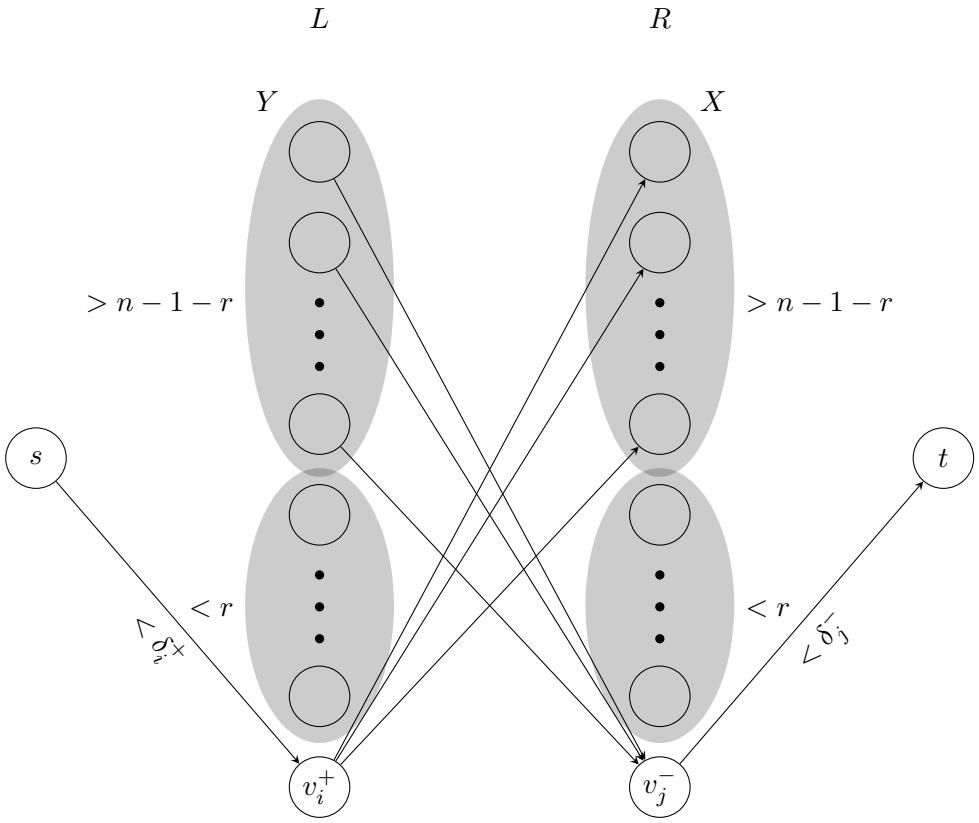


Figure 5: A snippet of a residual graph associated with a flow network as defined in Construction 3.4. The picture illustrates the situation that occurs when unsatisfied vertices v_i^+ and v_j^- exist. The set X denotes all vertices to which v_i^+ has a forward arc while the set Y denotes all vertices that have a forward arc to v_j^-. The indices i and j are not necessarily equal, they were just drawn at the same height for purposes of better visibility.

Proof. Assume that

$$((\deg^-(v_1), \ldots, \deg^-(v_n)), k', r^-, \tau^-)$$

and

$$((\deg^+(v_1), \ldots, \deg^+(v_n)), k', r^+, \tau^+)$$

are yes-instances of NCE with $k' \in \{r^2+r+1, \ldots, k\}$. Hence, there exist $d_1^-, \ldots, d_n^-$ and $d_1^+, \ldots, d_n^+$ such that for each $i \in \{1, ..., n\}$ it holds that $d_i^- \geq \deg^-(v_i)$, $d_i^+ \geq \deg^+(v_i)$ and

$$\sum_{j=1}^{n} d_j^+ - \deg^+(v_j) = k' = \sum_{j=1}^{n} d_j^- - \deg^-(v_j).$$

With these demand-values for each vertex in hand, we can build a flow network N as described in Construction 3.4. Due to Lemma 3.5 and $k' > r^2 + r$, it follows that there exists a flow f in $N = (V_N, A_N)$ with value k'. By adding all arcs $(v_i, v_j) \in V^2$ to A such that $f(v_i^+, v_j^-) = 1$, we perform exactly $k' \leq k$ arc additions, resulting in a new graph D'. Furthermore, each vertex $v_i^+ \in V_N$ receives exactly $d_i^+ - \deg^+(v_i)$ flow from s and each vertex $v_j^- \in V_N$ sends exactly $d_j^- - \deg^-(v_j)$ flow to t. This is why in D' it holds that $\deg^+_{D'}(v_i) = d_i^+$ and $\deg^-_{D'}(v_i) = d_i^-$ for all $i \in \{1, \ldots, n\}$. Since both $(d_1^+, \ldots, d_n^+)$ and $(d_1^-, \ldots, d_n^-)$ arise from solving an instance of NCE, it holds that $\deg^+_{D'}(v_i) \in \tau^+(v_i)$ and $\deg^-_{D'}(v_i) \in \tau^-(v_i)$ for all $i \in \{1, \ldots, n\}$. Accordingly, I is a yes-instance. □

The final kernel containing $O(r^4)$ vertices directly follows from Lemma 3.6.

Theorem 3.7. DDCE(e^+) *admits a problem kernel containing* $O(r^4)$ *vertices computable in* $O(n(n + r))$ *time.*

Proof. Let $I := (D, k, r^+, r^-, \tau^+, \tau^-)$ be an instance of DDCE(e^+). Suppose that there is a $k' \in \{r^2 + r + 1, \ldots, k\}$ such that both $((\deg^+(v_1),\ldots,\deg^+(v_n)),k',r^+,\tau^+)$ and $((\deg^-(v_1),\ldots,\deg^-(v_n)),k',r^-,\tau^-)$ are yes-instances of NCE. As a result of Lemma 3.6 it follows that I is a yes-instance. This yes-instance is computable in $O(n(m + k \cdot r))$ time since, as mentioned earlier, NCE is solvable in $O(n \cdot k \cdot r)$ time and it is fact that the maximum flow problem is solvable in $O(n \cdot m)$ time [Orl13]. However, if for each $k' \in \{r^2 + r, \ldots, k\}$ either $((\deg^+(v_1),\ldots,\deg^+(v_n)),k',r^+,\tau^+)$ or $((\deg^-(v_1),\ldots,\deg^-(v_n)),k',r^-,\tau^-)$ is a no-instance of NCE, then it is safe to solve the problem with $k := r^2 + r$ since every solution that would make use of more than $r^2 + r$ arc additions would also result in a solution for NCE. Applying the $O(kr^2)$-vertex kernel from Theorem 3.3 yields the final kernel with $O(r^4)$ vertices. The running time comes from calculating the $O(kr^2)$-vertex kernel and therefore takes $O(n(n + r))$ time. □

4 Problem kernel for Directed Tuple Degree-Constraint Editing(e^+)

In the previous section we discussed how to achieve a kernel for DDCE(e^+) containing $O(r^4)$ vertices. In this section we are studying the second variant of the problem we defined in the beginning: DTDCE(e^+). Let $I = (D, k, r^+, r^-, \tau)$ be an instance of DTDCE(e^+). The goal will again be to get a kernel with respect to the single parameter $r := \max(r^+, r^-)$. Since we are using tuple constraints now instead of separate constraints for both the in-degree and the out-degree, there are fewer allowed in-degree/out-degree combinations. For example, if we have $\tau^+(v) = \{1, 3, 5\}$ and $\tau^-(v) = \{2, 4, 6\}$ with three entries in each list, this would allow for nine in-degree/out-degree-combinations. Whereas by having $\tau(v) = \{(1, 2), (1, 4), (3, 4), (3, 6), (5, 4), (5, 6)\}$ with the "combined" size of six entries, only those six in-degree/out-degree-combinations are allowed. More specifically, it is clear that DTDCE(e^+) generalizes DDCE(e^+) since it is possible to model every constraint that is defined by two separate lists with just one list. One just has to define the single tuple list as the Cartesian product of both lists. Nevertheless, the key to our solution for DDCE(e^+) was the fact that the problem without the graph structure is polynomial-time solvable. We were able to calculate demands for each vertex to build a flow network which gave us the needed bound for the parameter k. As a consequence, if we could get in- and out-demands for each vertex as before, the solution to the problem would be the same as for DDCE(e^+). Consequently, being able to use the same strategy that worked earlier comes down to finding demands for the tuple based problem. Thus, we need a new definition of Number Constraint Editing. Luckily, by just slightly modifying it we are able to not only define such problem but also use a very similar algorithm as before to solve it. The key to successfully using flow networks was that there were exactly k modifications done to the input-degree list and exactly k modifications done to the output-degree list. For DTDCE(e^+) this means we need to allow for exactly k modifications of the form $(i_1, i_2) \to (i_1 + 1, i_2)$ and also for exactly k modifications of the form $(i_1, i_2) \to (i_1, i_2 + 1)$. While doing this, we restrict ourselves to modifications $(i_1, i_2) \to (i'_1, i'_2)$ such that $(i'_1, i'_2) \in \tau(i)$. This brings us to our new definition:

Tuple Constraint Editing (TCE)

Input: A function $\tau\colon \{1, \ldots, n\} \to 2^{\{0,\ldots,r_1\}\times\{0,..,r_2\}}$ and n tuples of non-negative integers $(c_1, d_1) \ldots, (c_n, d_n)$ as well as three non negative integers k, r_1, r_2.

Question: Are there n integer tuples $(c'_1, d'_1), \ldots, (c'_n, d'_n)$ such that

$$\sum_{i=1}^{n}(c'_i - c_i) = \sum_{i=1}^{n}(d'_i - d_i) = k$$

and for all $i = 1, \ldots, n$ it holds that $c'_i \geq c_i, d'_i \geq d_i$ and $(c'_i, d'_i) \in \tau(i)$?

If one thinks about this problem not as increasing numbers, but instead as adding arcs, the definition becomes very clear. Each input tuple (c_i, d_i) represents the (in-degree, out-degree) tuple of one vertex, whereas each output tuple (c'_i, d'_i) represents the (in-degree,

out-degree) tuple of one vertex after k edge additions. Since each added arc increases exactly one in-degree and exactly one out-degree by one, the overall in-degree difference $\sum_{i=1}^{n}(c'_i - c_i)$ has to be the same as the overall out-degree difference $\sum_{i=1}^{n}(d'_i - d_i)$. And since we are adding k arcs, they both have to equal k. This problem is polynomial-time solvable:

Lemma 4.1. TCE *is solvable in* $O(n \cdot k^2 \cdot r_1 \cdot r_2)$ *time.*

Proof. Let $I := (\tau, (c_1, d_1) \ldots, (c_n, d_n), k, r_1, r_2)$ be an instance of TCE. We solve the problem by using a modified version of the dynamic programming algorithm for NCE due to Froese et al. [Fro+14]. To this end, we use the following recurrence with M being defined as a 3-dimensional boolean array of size $n \times k \times k$:

$$M[i,j,l] = \text{true} :\Leftrightarrow \exists (c'_i, d'_i) \in \tau(i) : c'_i \geq c_i \wedge d'_i \geq d_i \wedge M[i-1, j-(c'_i - c_i), l-(d'_i - d_i)] = \text{true}$$

The parameter i is used for limiting the last tuple pair we are considering. The parameter j limits the amount of additions $(i_1, i_2) \to (i_1 + 1, i_2)$ we have left, whereas the parameter l limits the amount of additions $(i_1, i_2) \to (i_1, i_2 + 1)$ we have left. The recurrence terminates with

$$M[1,j,l] = \begin{cases} \text{true}, & \text{if } (c_1 + j, d_1 + l) \in \tau(1) \\ \text{false}, & \text{else.} \end{cases}$$

The algorithm is correct because in the worst case at position i the algorithm tests all possibilities for $(c'_i, d'_i) \in \tau(i)$. Since there are $n \cdot k^2$ entries in the array, and at each entry the algorithm considers $r_1 \cdot r_2$ possibilities, the running time of the algorithm is in $O(n \cdot k^2 \cdot r_1 \cdot r_2)$. □

We will use TCE on the combined in- and out-degree list of the graph D to obtain in- and out-demands as before:

$$((\deg^-(v_1), \deg^+(v_1)), \ldots, (\deg^-(v_n), \deg^+(v_n))) \xrightarrow{\text{TCE}(\tau, k, r^+, r^-)} ((d_1^-, d_1^+), \ldots, (d_n^-, d_n^+)).$$

The only property we obtained from using Number Constraint Editing in Section 3 on both the in-degree- and the out-degree sequence was the fact that

$$\sum_{i=1}^{n} d_i^+ - \deg^+(v_i) = k = \sum_{i=1}^{n} d_i^- - \deg^-(v_i).$$

This property however is by definition still valid after using TCE on the combined in-degree/out-degree sequence. Hence, at this point the differences between DDCE(e^+) and DTDCE(e^+) are negligible. We can create the very same flow network as in Construction 3.4 and only work with the non-tuple demands. We can then use Lemma 3.5 to obtain a problem kernel of size $O(r^4)$ with exactly the same reasoning as before. Reduction Rule 3.1 and Reduction Rule 3.2 also hold for DTDCE(e^+), with the slight modification of using tuple types (i, j) instead of in-type i and out-type j. A vertex v has *tuple*

type (i, j) if $(\deg^-(v) + i, \deg^+(v) + j) \in \tau(v)$. With the choice of $\alpha := k \cdot (\Delta_D^+ + \Delta_D^- + 1)$ we can use Construction 3.1 with the adjustment of only adding α vertices of each tuple type. We can then use the same proof as in Lemma 3.2. We just switch each vertex $v^\star \notin C$ with another vertex $w \in C$ which has a tuple type such that it can take all added arcs incident to $v^\star$. Such a vertex can always be found because of the choice of α. The running time is a combination of calculating the in- and out-degree for each vertex($O(n^2)$ time), computing the tuple type for each vertex($O(nr^2)$ time) and checking for all tuple types of each vertex whether we still need vertices of that type($O(nr^2)$ time). We refer to Section 3 for more details.

Theorem 4.2. DTDCE(e^+) *admits a problem kernel of size* $O(r^4)$ *which is computable in* $O(n(n + r^2))$ *time.*

5 On Transferring the results to undirected graphs

In this section we are taking a look at DCE(e^+) which is the undirected problem that was the starting point of this work. The existence of problem kernels with respect to the maximum allowed degree r was proven for both the undirected problem DCE(e^+) and the directed problems DDCE(e^+) and DTDCE(e^+). In the previous sections we discussed how to achieve this in the directed case and the result was a problem kernel containing only $O(r^4)$ vertices. This result provides an smaller kernel compared to the $O(r^5)$-vertex kernel from Froese et al. [Fro+14]. Hence, one might ask whether the strategy for directed graphs is also applicable for undirected graphs. Maybe it is possible to improve the $O(r^5)$-vertex-kernel to match our bound $O(r^4)$. Since the first part (achieving a kernel containing $O(k \cdot r^2)$ vertices) was using similar strategies, the main focus lies on constructing a flow network analogous to that one defined in Construction 3.4. The advantage of using directed graphs arises from the fact that it is easier to translate them into flow networks. Because flow networks are directed graphs themselves, it is clear how to translate the arcs of a directed graph into the arcs of a flow network. Furthermore, directed graphs give us a natural partition of the resulting flow network in vertices with outgoing arcs and vertices with incoming arcs so that the flow network has a simple structure. For undirected graphs, this is more difficult. To be able to use the strategies of Section 3 we therefore need a way to translate our undirected graph into a directed flow network. The first approach crossing one's mind is to just replace each undirected edge $\{v, w\}$ with two directed arcs (v, w) and (w, v). Then, we could build the flow network from Construction 3.4 and proceed with the algorithm as before. Regarding the demands, each vertex v_i has only one demand d_i, that arises from the solution of Number Constraint Editing with the degree-sequence from G as the input. Also, we need to double k for solving Number Constraint Editing since each added arc increases the degree of two vertices:

$$(\deg(v_1), \ldots, \deg(v_n)) \xrightarrow{\text{NCE}(\tau, 2k, r)} (d_1, \ldots, d_n)$$

We will set $d_i^+ = d_i^- = d_i$ in Construction 3.4. This leads to:

Construction 5.1. *Let $G = (\{v_1, \ldots, v_n\}, E)$ be the graph we are working on.*

- *For each vertex $v_i \in \{v_1, \ldots, v_n\}$, add two vertices to the network: v_i^+ and v_i^-.*
- *Add a source-vertex s and a sink-vertex t to the network.*
- *Add an arc (s, v_i^+) with capacity $d_i - \deg(v_i)$ for each $i = 1, \ldots, n$.*
- *Add an arc (v_i^-, t) with capacity $d_i - \deg(v_i)$ for each $i = 1, \ldots, n$.*
- *Add an arc (v_i^+, v_j^-) with capacity 1 if $\{v_i, v_j\} \notin E$.*

See Figure 6 for a schematic flow network obtained by performing Construction 5.1. The problem with this approach is the translation from the directed arcs back to the undirected edges. Earlier, we added an arc $(v_i, w_j) \in V^2$ to the graph $D = (V, A)$ if the

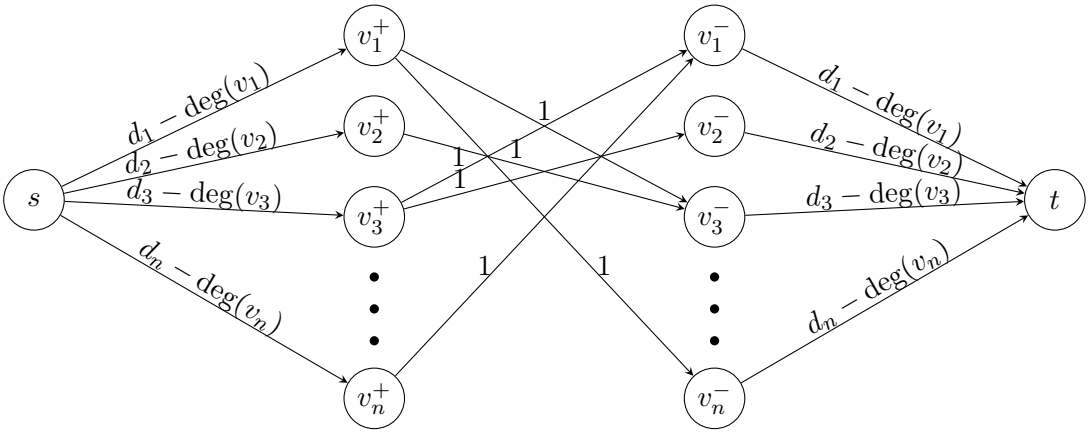


Figure 6: A schematic flow network obtained by Construction 5.1. The integers d_i come from solving an instance of Number Constraint Editing. For each vertex v_i in the graph D there are two vertices added to the flow network: v_i^+ and v_i^-. We connect a vertex v_i^+ to a vertex v_j^- if the edge $\{v_i, v_j\}$ is not in D.

flow on the arc $(v_i^+, v_j^-) \in V_N^2$ in the flow network $N = (V_N, A_N)$ equaled one. But that alone does not ensure the output-graph to be undirected. For that to be the case we would need to ensure that for the resulting flow f, both $f(v_i^+, v_j^-) = 1$ and $f(v_j^+, v_i^-) = 1$. Only if this is the case we could add both arcs (v_i, v_j) and (v_j, v_i) to the graph so that we are able to exchange them by the undirected edge $\{v, w\}$. We obtain the following:

Observation 5.2. *If we build a flow network as described in Construction 5.1, then we are able to translate a maximum flow f back to undirected edge additions if and only if for each $i, j = 1, \ldots, n$ it holds that $f(v_i^+, v_j^-) = 1 \Leftrightarrow f(v_j^+, v_i^-) = 1$.*

However, there are flow networks where maximum flows exist which do not satisfy this property. The complete bipartite graph $K_{3,3}$ containing six vertices where we set $\tau(v) = \{4\}$ for each vertex $v \in \{v_1, \ldots, v_6\}$ and we choose $k = 3$ is an example for such a graph. Obviously, $((\deg(v_1), \ldots, \deg(v_6)), 2k, 4, \tau)$ is a yes-instance of Number Constraint Editing. It is solvable by just incrementing the degree of each vertex by one. But using Construction 5.1 results in a flow network where no flow exists such that Observation 5.2 holds. Figure 7 illustrates the graph as well as the resulting flow network. By setting the flow on each dashed arc to one we can obtain a maximum flow. However, one can verify that there is no flow f such that $f(v_i^+, v_j^-) = 1 \Leftrightarrow f(v_j^+, v_i^-) = 1$. That may not be a problem since $K_{3,3}$ with the chosen constraints is a no-instance of DCE(e^+). Nevertheless, this example shows that the first simple approach alone is not enough. We would additionally need a new flow algorithm that always chooses maximum flows where the property described in Observation 5.2 is valid. This may give us a relationship of the kind that there is a flow f such that this property is valid if and only if the original graph is a yes instance of DCE(e^+). There are, however, examples for flow networks where exponentially many maximum flows exist. We therefore leave the question whether

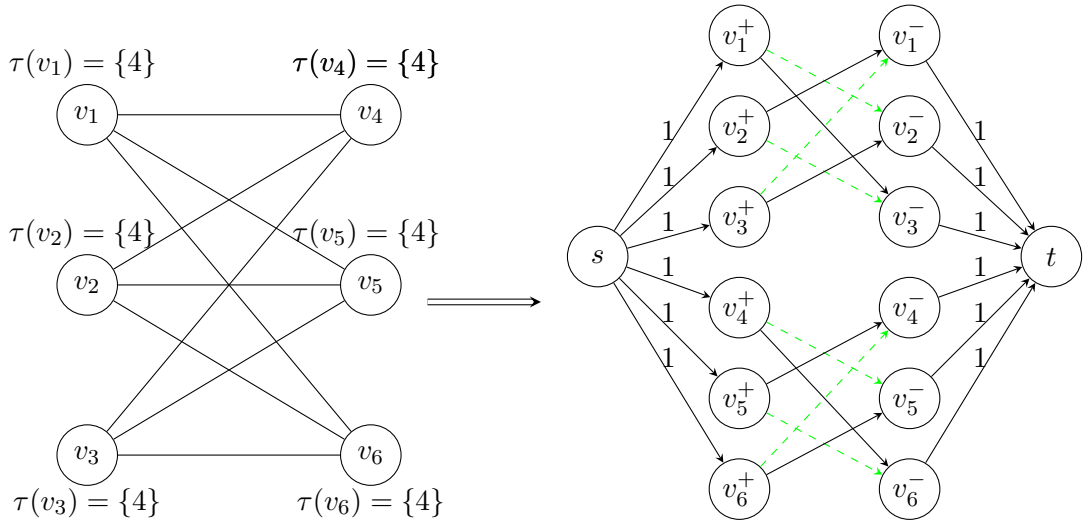


Figure 7: The capacity of each arc (v_i^+, v_j^-) is one. It was dropped from the picture to improve readability. The left side shows the complete biparte graph $K_{3,3}$ with given degree constraints. The right side shows the flow network that is obtained by Construction 5.1. Choosing the dashed arcs results in a flow of value six. There is however no maximum flow such that the property mentioned in Observation 5.2 is valid.

there is a polynomial-time algorithm that always finds maximum flows with the property mentioned in Observation 5.2 open for future research. Overall, the problem with the first approach revolves around translating a maximum flow back to edge additions. So maybe it is possible to design a flow network where that is no problem.

This brings us to our second approach of sorting the vertices. If the vertices are in a certain order $<$, then this would give us a 1-to-1 relation where an undirected edge $\{v, w\}$ is associated with the directed edge (v, w) if and only of $v < w$. We therefore modify Construction 3.4 so that we add an arc $v_i^+ \to v_j^-$ only if $v_i < v_j$. This will result in a flow network which is similar to the flow network illustrated in Figure 8.

The problem with this construction is the question of how to choose the demands δ_i^+ and δ_i^-. Let us say that we solve Number Constraint Editing as usual and for the vertex v_3 in the picture we get the solution $d_3 = 2$. Then, do we choose $\delta_3^+ = 1 = \delta_3^-$ or do we choose $\delta_3^+ = 0$ and $\delta_3^- = 2$? Choosing the first option will clearly force the algorithm to add the edge $\{v_3, v_n\}$ to the original graph while the second option will force the algorithm to not add this edge. However, since the edge may or may not be in a solution for the given graph, there is no answer to this question. Also, one cannot brute force this since with n vertices and every solution d_i being at most r there are in the worst case r^n possibilities of distributing the demands. To conclude it can be said that it may be possible to use our strategies to improve the kernel for the undirected case. A new flow algorithm which chooses maximum flows with the property mentioned in Observation 5.2 would be one way of using our strategies in the undirected case. Otherwise, it would

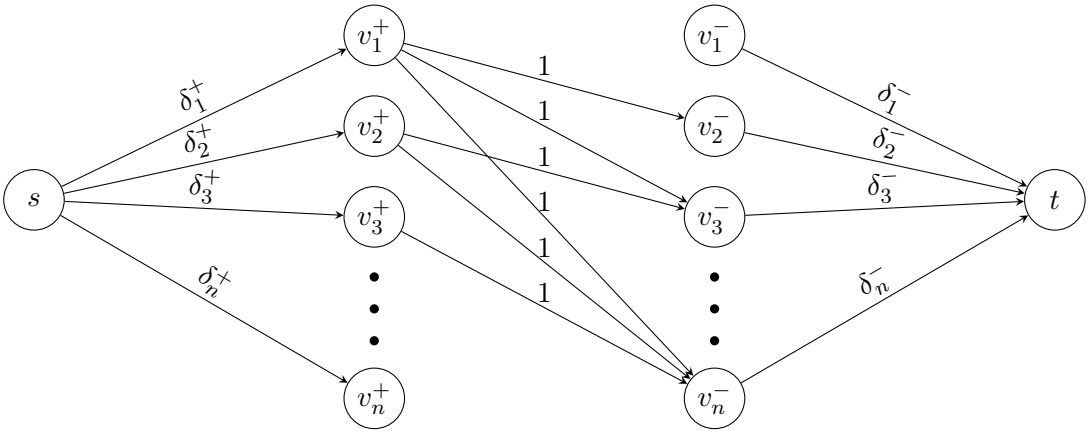


Figure 8: A schematic flow network if we modify Construction 3.4 so that we sort the vertices in a certain order $<$. For each vertex v_i in the graph D there are two vertices added to the flow network: v_i^+ and v_i^-. We connect a vertex v_i^+ to a vertex v_j^- if the arc (v_i, v_j) is not in D and if $v_i < v_j$. The problem with this construction is the question of how to choose δ_i^+ and δ_i^-.

require a better way of translating an undirected graph into a flow network. Directed graphs give a natural partition of the flow network in vertices with outgoing and vertices with incoming arcs which is not the case for the undirected problem. We therefore at this point leave the question of transferability open for future research.

6 Conclusion

The goal of this work was to transfer results from Froese et al. [Fro+14] for undirected graphs to directed graphs. They achieved a kernel with respect to the single parameter r (which limits the maximum allowed degree) by first taking an intermediate step and constructing a kernel with respect to the combined parameter (k, r), where k is the solution size. This final problem kernel contained $O(r^5)$ vertices. We first came up with two possible ways of defining the problem on directed graphs: DDCE(e^+) and DTDCE(e^+). Although DTDCE(e^+) is a generalization of DDCE(e^+), we showed that both problems are amenable to the same strategies. To be able to do that, we took the same intermediate step as Froese et al. [Fro+14] and first constructed a kernel with respect to the combined parameter (k, r). In that part, we used the same strategies. The novel part of this work is the second step: constructing a kernel with respect to r. While Froese et al. [Fro+14] used the concept of *f-factors*, we were able to utilize flow networks. By doing so, we were able to reduce the kernel size, leading to our final results: both DDCE(e^+) and DTDCE(e^+) admit a kernel containing $O(r^4)$ vertices. The difference between using flow networks and f-factors seems to result in a kernel that saves a linear factor for the kernel size compared to the $O(r^5)$-vertex kernel from Froese et al. [Fro+14]. We can therefore say in conclusion that our initial goal was not only met, but also surpassed. In the end, we also took a look at the problem on undirected graphs. We outlined two basic approaches that tried to make use of the strategies that worked for directed graphs to improve the kernel for undirected graphs. We outlined the problems that occur with both approaches and left open whether it is possible to improve the kernel from Froese et al. [Fro+14] on undirected graphs. We also left open whether similar results are achievable for vertex deletion or arc deletion on directed graphs. It was proven by Froese et al. [Fro+14] that these problems probably do not admit a polynomial-size kernel in the undirected case, but the results may not transfer to directed graphs. Other fields of research could be to examine what exactly makes the problem computationally hard and design efficient algorithms for special cases. The problem on the degree sequence is polynomial-time solvable, so maybe it also is efficiently solvable on graphs with low connectivity. In fact, at the moment we are uncertain whether the problem is efficiently solvable on the empty graph. The empty graph provides no initial constraints on what degree values we can raise. However, there are cases where the number problem is a yes-instance and the graph problem is a no-instance.

Literature

[Ban+15] J. Bang-Jensen, J. Huang, and X. Zhu. "Completing orientations of partially oriented graphs". In: *Computing Research Repository* abs/1509.01301 (2015) (cit. on p. 6).

[Ban+95] J. Bang-Jensen, A. Frank, and B. Jackson. "Preserving and Increasing Local Edge-Connectivity in Mixed Graphs". In: *SIAM Journal on Discrete Mathematics* 8.2 (1995), pp. 155–178 (cit. on p. 6).

[BG02] J. Bang-Jensen and G. Gutin. *Digraphs - Theory, Algorithms and Applications*. Springer, 2002 (cit. on p. 5).

[Cyg+15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015 (cit. on p. 9).

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013 (cit. on p. 9).

[Dor+13] F. Dorn, H. Moser, R. Niedermeier, and M. Weller. "Efficient Algorithms for Eulerian Extension and Rural Postman". In: *SIAM Journal on Discrete Mathematics* 27.1 (2013), pp. 75–94 (cit. on p. 6).

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, 2006 (cit. on p. 9).

[Fro+14] V. Froese, A. Nichterlein, and R. Niedermeier. "Win-Win Kernelization for Degree Sequence Completion Problems". In: *Proceedings of the 14th Scandinavian Symposium and Workshops*. Vol. 8503. Lecture Notes in Computer Science. Springer, 2014, pp. 194–205 (cit. on pp. 3, 6, 7, 11, 14, 18, 24, 26, 30).

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979 (cit. on p. 11).

[KT06] J. M. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2006 (cit. on p. 10).

[LM11] A. N. Langville and C. D. Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011 (cit. on p. 5).

[MS12] L. Mathieson and S. Szeider. "Editing graphs to satisfy degree constraints: A parameterized approach". In: *Journal of Computer and System Sciences* 78.1 (2012), pp. 179–191 (cit. on pp. 5, 6).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cit. on p. 9).

[Orl13] J. B. Orlin. "Max flows in O(nm) time, or better". In: *Symposium on Theory of Computing Conference*. 2013, pp. 765–774 (cit. on pp. 10, 22).

[Wel+12] M. Weller, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. “On making directed graphs transitive”. In: *Journal of Computer and System Sciences* 78.2 (2012), pp. 559–574 (cit. on p. 6).