Structural Properties of Decomposable Digraphs

by

Gabriella Juhl Jensen

supervised by

Prof. Jørgen Bang-Jensen

UNIVERSITY OF SOUTHERN DENMARK

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Contents

0.1 Introduction

In everyday life there are many different problems that can be described with graphs. For instance, when planning postal routes for post delivery services it might become of interest if there exists a route in which every house is visited once. The problem of finding a path that visits every house exactly once, is called the hamiltonian path problem. If it is further required that the postman's final destination is the postal office, that is, where the route began, then the problem is called the hamilotnian cycle problem. Both problems lie in a class of problems named NP-complete, which are computationally difficult. In fact, many graph related problems lie within the class of NP-complete. Fortunately, for some particular graphs named "decomposable digraphs", a solution might exist that is computationally faster than otherwise.

In this work, algorithms are explored that either find a solution in polynomial time or decide that the problem is not solvable in polynomial time. We will focus on three problems that are the hamiltonian path and cycle problem, the linkage problem and the weak linkage problem. In all of the aforementioned problems, the structure of the decomposable digraph in question is the key to finding a computationally fast solution. The decomposable digraphs that investegated the most, are the quasi-transitive and the locally-semicomplete, and therefore are the focus of this thesis.

Part I

Decomposable Digraphs and the Hamilton Cycle Problem.

Chapter 1

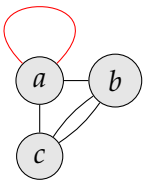
Notation and Graph Classes

This chapter introduces graphs and notation. Notation in this thesis may diverge from the notation of some articles to ensure a uniform notation. section 1.3 introduces names and notations of graph classes that will be explored throughout this thesis.

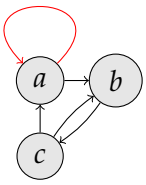
1.1 Graphs and Digraphs

Before delving into structural properties of decomposable digraphs, we first need to establish what a graph is. Let $G(V, E)$ where V and E are two sets contaning the **vertices** (also commonly called nodes) and **edges** of the graph respectivlely (for an example of one, see Figure 1.1a). We define the **size** of the graph to be the number of vertices $|V|$. This is also known as the **cardinality** of V. An **edge** $e \in E$ means $e \equiv (a, b)$ and $\{a, b\} \subseteq V$, say e is an edge in G, e is called **incedent** to a and b. We call $a, b \in V$ **adjacent** if there is an edge (a, b) or (b, a) (two given vertices connected by an edge is said to be adjacent). If an edge goes from and to the same vertex (a, a) then the edge is called a **loop**. The set of edges $e_1, \ldots, e_k$ are denoted by E where each edge is a pair of vertices that are adjacent. The letter V denotes the set of vertices in a graph.
In a graph, we have something called a **walk** which is a alternating sequence
$W = x_1e_1x_2e_2x_3 \ldots x_{k-1}e_{k-1}x_k$ of vertices x_i and edges e_j from the graph G such that the edge e_i is between x_{i-1} and x_i for every $i \in [k]$. We call a walk closed if the first vertex in the walk is the same as the last.
Every vertex $v \in V$ of $G(V, E)$ have a **degree** denoted $d(v)$ which is the number of incident edges to v. A **path** in a graph is a walk where each vertex in the ordering only appears one time. A **cycle** is a closed walk where the only vertex present more then one time is the first vertex (also called a closed path). Let X be a subset of the vertices $X \subseteq V$. Then we say that $V \backslash X$ is the set of vertices without the vertices in X, i.e. $V \backslash X \equiv V - X$. A **subgraph** H of G can contain any of the vertices and the arcs connected to the chosen vertices in H. You can not have an edge conecting no vertices in H but you do not have to choose all the arcs in G between the chosen vertices in H for H to be a subgraph.
As we can look at subsets we sometimes need to look at sub-paths, for a path $P = x_1 \ldots x_k$ a **sub-path** is a path $P' = x_i \ldots x_j$ of P where $1 \leq i < j \leq k$.

Before delving more specific into graphs and digraphs, we must establish some important prerequisites and properties. A graph is called **simple** if there are no loops and no multiple edges. With multiple edges it means multiple edges between the same pair of



(a) Graph $G(V, E)$ is an example of a graph, the red edge is a loop, and all piars of vertices in this graph are adjacent.



(b) This is an orientation of the edges in the graph which makes this a digraph.

vertices like in Figure 1.1a between b and c.

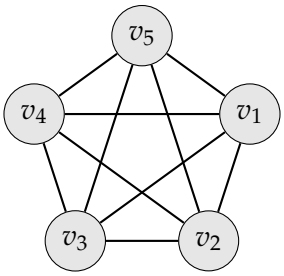


Figure 1.2: Complete graph with 5 vertices.

A graph is **connected** if there exists a path between every pair of vertices in the graph and **disconnected** otherwise. A graph is called **complete** if, for all pair of vertices in the graph is an edge between them as seen in Figure 1.2.
Somtimes, when looking at a specific set of vertices, we are actually interested in something called an **independent set**, which is a set of vertices of G where there is no edge between the vertices in the set. A maximal independent set of G is an independent set where you can not add any new vertex in the set that is not adjencant to any vertex in the set (adding a vertex makes the set no longer independent). The maximum independent set is the maximal independent set with greatest cardinality. Let $I \subset V$ be a maximum independet set. Then, $|I|$ is called the **independence number**.

If, instead of edges, have **arcs** between the vertices we call it a **digraph**. An arc is described just like an edge with two adjacent vertices (a, b). The first element in tuple is the vertex **from** which the arc starts, also called the **tail**. The second vertex is where the arc is pointing **to** also called **head**. The set of arcs is normaly denoted A like the set of edges is denoted E (so the arc (a, b) goes from a to b, if you wanted it the other way around the arc is (b, a)). This graph containing only arcs and no edges is called a digraph $D(V, A)$, which is what we in this project are focusing on. See Figure 1.1b (as G denotes a **G**raph, D denotes a **D**igraph).
For two vertices x and y in $D(V, A)$, if we have an arc from x to y, we say that x **dominates** y. This is denoted $x \to y$. If we consider subgraphs A and B, then A **dominates** B if for all $a \in A$ and $b \in B$, $a \to b$. If there are no arcs from B to A, we denote it $A \mapsto B$, and if both $A \to B$ and $A \mapsto B$, we say that A **completely dominates** B. This is denoted $A \Rightarrow B$.

Sometime, when working with a digraph or solving a problem, we have a subset of vertices $X \subseteq V(D)$ that want to work with as one vertex. Then we **contract** the vertices X into one vertex x where $N^+(X)\backslash X = N^+(x)$ and $N^-(X)\backslash X = N^-(x)$ (so we only keep the

ingoing and outgoing arcs of X and delete all vertices of X and the arcs inside). When we contract X in D, we will use the notation D/X. There is also another kind of contraction where you also delete possible multiplicity of arcs, if this is the case it will be explained in the section.

In a digraph we have something called the **underlying graph** denoted $UG(D)$. An underlying graph of a digraph is where all arcs are replaced by edges (edge is used every time we talk about undirected edges between vertices, when using directions it is called an arc). Let $X \subseteq V$. Then we can make the subdigraph $D\langle X\rangle$ which is the subgraph D **induced by** the set X meaning that the subdigraph contains all the vertices in X and all arcs in $A \in G$ where both head and tail is incident to the vertices in X. We will denote the graph $D\langle V \backslash X\rangle$ for some $X \subseteq V$ as $D - X$.
A digraph is **connected** if the underlying graph is connected, also called weakly connected. A digraph can be **strongly connected** and **semi connected** too. A digraph is called **semi connected** if there for each pair u and v exists a path from either u to v or v to u. It is said to be **strongly connected** if for each pair of vertices u and v there exists a path from both u to v and v to u. A strongly connected digraph is also called a **strong** digraph. A strong digraph has a subset S called a **seperator**. If $D - S$ is not strong, we also say that S **seperates** D. A seperator S is called **minimal seperator** of D if there exists no proper subset $X \subset S$ that seperates D. Now we can introduce a k**-strong** digraph D which is a strong digraph where $|V| > k$ and a minimal seperator S is where $|S| = k$. In the same way we can define k**-arc-strong** digraph, where at least k arcs need to be deleted for the digraph to no longer be strong.

In a digraph $D(V, A)$, we mostly use the **degree** as two different degrees, namely **out-degree**, $d^+(v)$, and **in-degree**, $d^-(v)$, that is the number of arcs going from v and to v, respectively. In a digraph D, we consider the overall **minimum out-degree**, $\delta^+(D) = \min\{d^+(v)|v \in V\}$ and **minimum in-degree**, $\delta^-(D) = \min\{d^-(v)|v \in V\}$. Somtimes, we are going to need the minimum of these two $\delta(D) = \min\{\delta^+(D), \delta^-(D)\}$ called the **minimum degree**. For every vertex v, the vertices that are **adjacent** with v is called **neighbours** of v. We denote $N^+(v)$ and $N^-(v)$ as the set of vertices that is dominated by (**out-neighbours** of) v and dominates (**in-neighbours** of) v, respectivly. This means that $d^+(v) = |N^+(v)|$ and $d^-(v) = |N^-(v)|$.

For simplicity, when mentioning paths and cycles in digraphs, it will be **directed** paths and cycles if not anything else is mentioned. **Directed** means that we go from tail to head on every arc on the path or cycle. When mentioning paths in a digraph, it sometimes makes more sense specifing the head and tail of the path, so a path from s to t is denoted as an $(s,\ t)$**-path**. In some digraphs, there is more than one path between the same two vertices, say Q and P. If the paths are **disjoint**, they have no vertices in commen, $V(Q) \cap V(P) = \emptyset$, and they are **arc-disjoint** if they have no arcs in commen $A(Q) \cap A(P) = \emptyset$ the maximum number of disjoint path between two vertices in a digraph is denoted $\lambda_D(s, t)$.

1.2 Computational complexity

In this section we will go over how time is measured for an algorithm and what it means for a problem to be polynomially solveable or polynomially verifyable. Also what it means for a problem to be NP-hard and NP-complete and how we found out if a problem is either

of them.

1.2.1 Measure time of algorithm (Polynomial, exponential)

The runing time of an algorithm is based on how many steps it is going thourgh which is somtimes based on the input that the algorithm takes we are going to denote an algorithms running time as a function $f(n)$ over the input n. This is how different functions can descirbe the running time of an algorithm, if an algorithm has the same number of steps no matter what the input is it has a constat running time where the constant is the number of steps the algorithm uses.

An algorithm can also take the form of an polynomial function or even exponentiel, if this is the case we uses notation as big-O for the runningtime rounded up. Big-oh is the most used one and is the notation we are going to use in this thises, if the algorithm takes $f(n) = 4n^3 + 2n^2 - n + 2$ time we denote it in big-oh notation as $O(n^3)$ as it is the biggest term of $f(n)$.

The shorter the runningtime is the better the algortihm is. Since the exponentiel runningtime algortihms take forever on large inputs, we would want to improve them, but sometimes you are left with problems where that is not a possibility.

So we are going to classify the problems in gruops of how long time it take to decide or verify the problems solution. a problem that is decided i polynomiel time is in the class called P. Which means for every given time of input in a problem from **P** we can find the solution for the problem in polynomiel time.

1.2.2 NP problems and classifications

As shortly described above there is something called a **polynomial verifier** for a problem. That means given a problem and then given a solution we can in polynomial time verify if it is a solution to the given problem. This is the class we call NP.

Definition 1.2.1. ***NP*** *is the class of languages that have polynomial time verifiers.*

Obiously if you can find a solution in polynomial time you can also verify whether a solution is correct in polynomial time. So $P \subseteq NP$. There is also a class called $NP - Hard$ but before we can explain that we need to explain what it means for a problem to be polynomial reduceable to another problem. For a specific problem A and another problem B then if there exists an algorithm that can take a solution from A and make it a solution for B in polynomial time. When such a algorithm exists it is called a polynomial verifier and we say that A is **polynomial reducable** to B or just that A is **reduced** to B. **NP-Hard** are the class of problems that every NP problem can be polynomiel reduced to. A problems in the class of NP-Hard problems does not nessesarily mean that it is NP it-self. If a problem is both **NP** and **NP-Hard** we call it **NP-Complete**. The problems we are mostly focusing on is in the class of **NP-Complete** problems.

1.3 Classes of Digraphs

A **Tournament** is a digraph in which the underlying graph is complete. So in a complete graph of order 5, any orientation of the edges concludes in a tournament. Strong digraphs are also in themselves a classification of digraphs. Classes of digraphs can overlap each other or fully contained in each other, like tournaments is fully contained in the class called semicomplete digraphs. A **semicomplete** digraph is where the underlying graph is a complete multigraph, there can be some multiple edges in between the same pair of vertices in the underlying graph (parallel edges). Since the class called semicomplete digraphs contains all digraphs where the underlying graph is a complete multigraph, it clearly also contains the graph with only one arc between every pair of vertices (Tournaments). A digraph is **complete** if for every pair of vertices $a, b \in V$, the arc (a, b) and (b, a) are present in the graph.
If you can split the vertices of a graph V into two sets of vertices A and B such that $A \cup B = V$, and there are no arcs inside these sets, then we classify this as an **bipartite** digraph. This means all arcs in the graph are in the form (a, b) or (b, a) for all $a \in A$ and $b \in B$. The sets A and B are called the partites of $D(V, A)$. The underlying graph of a bipartite digraph is also called bipartite since there are no edges inside A or B. A much used type of digraph is an **acyclic** digraph. It is a digraph where there exists an ordering of the vertices $V = v_1, v_2, \ldots, v_n$ where the arcs in the digraph are (v_i, v_j) where $i < j$ for all $(v_i, v_j) \in A$. This ordering is called an **acyclic ordering**, and there can be many of these orderings in the same digraph. This ordering can also be used to order strong components in a non-strong digraph, such that the ordering of the components $C_1, C_2, \ldots C_k$ is an acyclic digraph when contracting the components into k vertices. When classifying digraphs, there are several ways of doing this, like **transitive** digraphs which are digraphs where for all vertices $a, d, c \in V$ where the arc (a, b) and b, c is present in the digraph ($\in A$), the arc (a, c) has to be a part of A too. Using the same kind of classification, there are digraphs which are **Quasi-transitive**, which is for all vertices $a, d, c \in V$ where the arc (a, b) and b, c is present in the digraph ($\in A$, a and c has to be adecent by at *least* one (more arcs in between are also allowed) arc in either direction ((a, c) or (c, a))). These graphs are going to be mentioned a lot in this thises since the graph is also what we call **decomposable**.
A **Decomposable** digraph $D = S[H_1, H_2, \ldots H_s]$ is a digraph D that can be decomposed into $H_1, H_2, \ldots, H_s$ **houses** and a digraph S denoted as the **quotient** digraph where $|V(S)| = s$. Let S be the quotient digraph of D where $V(S) = \{s_1, s_2, \ldots, s_s\}$. If each s_i is replaced by the digraph H_i $i = 1, 2, \ldots, k$, we have the digraph D, where $H_i \to H_j \in D$ if $s_i \to s_j \in S$. This is called a **composition** of S or a **decomposition** of D. This is the class of digraphs we are focusing on in this thesis. If all the houses are independent sets, we call $D = S[H_1, H_2, \ldots, H_k]$ the extension of S. If S is a semicomplete digraph, we call the extension of these **extended semicomplete** digraph. Like we already mentioned, Quasi-transitive digraphs are decomposable but we have several classes that are decomposable, and another class of digraphs that we are giong to cover in this dissertation are **locally semicomplete** digraphs.
First, we introduce **locally in-semicomplete** digraphs where for every in-neighboor of a vertex, $x \in V$ has to be adjacent. This has to be true for all $x \in V$ ($x \cup N^-(x)$ induces a semicomplete digraph $\forall x \in V$). When $x \cup N^+(x)$ for all vertices in D, it classifies as **locally out-semicomplete** digraphs. Respectively, it is called an out-locally semicomplete digraph if $\forall x \in V$, $N^+(x)$, has to be adjacent. If a digraph is both locally in-semicomplete and locally out-semicomplete, it is called a **locally semicomplete** digraph. Why both quasi-transitive digraphs and some locally semicomplete digraphs are decomposeble will be described in section 2.1.
The last class of digraph that are important for this thesis are the round digraphs. A di-

graph is called a **round** digraph if there exists an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that for all v_i, $N^+(v_i) = v_{i+1}, v_{i+2}, \ldots, v_{i+d^+(v_i)}$ and $N^-(v_i) = v_{i-d^-(v_i)}, v_{i-(d^-(v_i)-1)}, \ldots, v_{i-1}$.

Chapter 2

Decomposable Digraphs

Decomposable digraphs are what we in this thesis is focusing on. We have briefly introduced what a decomposable digraph is, but there are subclasses to focus on and a lot of other crucial definitions and theroems to cover about these digraphs before delving into the NP-hard problems. First, we cover some general things about decomposable digraphs. The next section is about quasi-transitive digraphs, and why they are a subclass of decomposable digraphs and ϕ_1-decomposable digraphs. At the end of the section, we prove that these decompositions can be found in polynomial time, which is going to be crucial for solving some NP-hard problems for this class of digraphs. Then we have a very general class of digraphs, locally semicomplete digraphs, this class can be split up to 3 different subclasses, where 2 of those are decomposable. This is covered in section 2.3 and is going to be used in later chapters.

2.1 General about Decomposable Digraphs

Recall that a decomposable digraph $D = S[H_1, H_2, \dots H_k]$ can be decomposed into a main graph S (also sometimes called **quotient** graph), where $|S| = k$ and k houses $H_1, H_2, \dots, H_k$, where each vertex in $S = \{v_1, v_2 \dots, v_k\}$ is replaced by the house (H_i replaces v_i). The arcs between the houses are as follows: $H_i \to H_j$ in D if $v_i \to v_j$ in S. Remember that for a set X to dominate another set Y (meaning every vertex in the dominating set dominates every vertex in the dominated set), we denoted it $X \to Y$. If no arc between v_a and v_b is in S, then there is no arc between the sets H_a and H_b in D. A nice property of decomposable digraphs is that if there is an arc between H_i and H_j, either one of the houses totally dominates the other (ex. $H_i \Rightarrow H_j$) or they dominate each other (ex. $H_i \to H_j$ and $H_j \to H_i$).

Decomposable digraphs can be classified by a set of digraphs ϕ. When $D = S[H_1, H_2, \dots, H_k]$ it is **ϕ-decomposable** if $D \in \phi$ or if $S \in \phi$. The chioces of H_i for $i = 1, 2 \dots, k$ does not determine anything about the digraph being ϕ-decomposable, but the class of **totally ϕ-decomposable** digraphs is where D is ϕ-decomposable and each H_i is totally ϕ-decomposable. We are going to make two such sets of digraphs: ϕ_1, which is the union of semicomplete digraphs and acyclic digraphs both classes described in section 1.3 and ϕ_2, which is the union of semicomplete and round digraphs also described in section 1.3.

$$\phi_1 = \{\text{Semicomplete digraphs}\} \cup \{\text{Acyclic digraphs}\} \tag{2.1}$$

$$\phi_2 = \{\text{Semicomplete digraphs}\} \cup \{\text{Round digraphs}\} \tag{2.2}$$

Take these sets ϕ_1 and ϕ_2. Then for every induced subdigraph of a digraph D where either $D \in \phi_1$ or $D \in \phi_2$, then the induced digraph is in the same set (so if $D \in \phi_1$ the induced subdigraph is in ϕ_1, same goes for ϕ_2). When this is true for a set ϕ, the set is called **hereditary**. So both ϕ_1 and ϕ_2 are hereditary.

Lemma 2.1.1. *Let ϕ be a hereditary set of digraphs. If a given digraph D is totally ϕ-decomposable, then every induced subdigraph D' of D is totally ϕ-decomposable.*

It also turns out that for ϕ_1 and ϕ_2, there exists an algorithm that checks whether a digraph D is totally ϕ_i-decomposable ($i = 1, 2$).

Theorem 2.1.2. *[1] There exists an $O(n^2m + n^3)$-algorithm for chekking if a digraph with n vertices and m arcs is totally ϕ_i-decomposable for $i = 1, 2$.*

$O(n^2m + n^3)$ is clearly a polynomial algortihm.

2.2 Quasi-transitive Digraph

First we need to recall what a quasi-transitive digraph is. For every triplet x, y, z in a quasi-transitive digraph, if $x \to y$ (x dominates y) and $y \to z$ (y dominates z), then there has to be at least one arc in either dirction between x and z. When working with quasi-transitive digraphs, there are many things you can depend on, things that the structure has already decided for us.

Lemma 2.2.1. *[1] Suppose that A and B are distinct strong components of a quasi-transitive digraph D with at least one arc from A to B. Then $A \to B$.*

Recall that this means that every vertex in A has an arc to every vertex in B. Like non-strong quasi-transitive digraphs, we can also say something about strong quasi-transitive digraphs.

Lemma 2.2.2. *[1, 2] Let D be a strong quasi-transitive digraph on at least two vertices. Then the following hold:*

(a) $\overline{UG(D)}$ is disconnected;

(b) If S and S' are two subdigraphs of D such that $\overline{UG(S)}$ and $\overline{UG(S')}$ are distinct connected components of $\overline{UG(D)}$, then either $S \to S'$ or $S' \to S$ or both $S \to S'$ and $S' \to S$ in which case $|V(S)| = |V(S')| = 1$.

These to lemmas play an important part in proving the following theorem, which states that quasi-transitive digraphs can be decomposed no matter if there are strong or nonstrong digraphs.

Theorem 2.2.3. *[2] Let D be a quasi-transitive digraph.*

1. *If D is not strong, then there exists a transitive acyclic digraph T on t vertices and strong quasi-transitive digraphs $H_1, \ldots, H_t$ such that $D = T[H_1, \ldots, H_t]$.*

2. *If D is strong, then there exists a strong semicomplete digraph S on s vertices and quasi-transitive digraphs $Q_1, \ldots, Q_s$ such that each Q_i is either a single vertex or is non-strong and $D = S[Q_1, \ldots, Q_s]$.*

This theorem is also what we are going to use more then once, to prove several of the problem solving theorems throughout this thesis.

Proof. Since we can decompose both strong quasi-transitive digraphs and non-strong quasi-transitive digraph, we are going to prove if D is not strong, and then if D is strong. So suppose D is not strong, then we know we can enumerate the strong components in an acyclic order let these be $H_1, \ldots, H_t$.
Recall that an acyclic ordering of the strong components does not mean that there are no arcs going back in the ordering, but we will prove that now.
Now from Theorem 2.2.1 we know that if there is an arc between two of the strong components, one of them dominates the other. Let without loss of generality these set be H_i and H_j and let $H_i \to H_j$. Then Since D is not-strong $H_j \nrightarrow H_i$. Now, let us say that $H_j \to H_k$. This means since D is quasi-transitive, then either $H_k \to H_i$ or $H_i \to H_k$. But since $H_i \cup H_j \cup H_k$ is not strong, $H_k \nrightarrow H_i$, meaning contracting each H_i for $i = 1 \ldots, t$ results in a transitive digraph T. We have also shown that there are no backwards arcs in the ordering, meaning that T is not only transitive but acyclic. This ends the proof of the non-strong quasi-transitive digraph leaving only the strong ones left.

Now suppose that D is a strong quasi-transitive digraph. We now look at the underlying graph $UG(D)$. After this, we find the complement of it, $\overline{UG(D)}$. Since D is strong, we know from Theorem 2.2.2 that $\overline{UG(D)}$ is disconnected, so we find $Q_1, \ldots, Q_s$ where $\overline{UG(Q_i)}$ is connected in $\overline{UG(D)}$ $\forall i \in [s]$.
Since these subdigraphs $\overline{UG(Q_i)}$ of $\overline{UG(D)}$ are connected, we know that Q_i is non-strong or a single vertex in D. From the same lemma, each Q_i (represent S in Theorem 2.2.2), which means when contracting Q_i $\forall i \in [s]$ into a single vertex q_i the resulting in a digraph S. We have that every pair of vertices $v, u \in S$ have an arc between them in either direction or arcs in both direction making S semicomplete.
This concludes the proof. □

From this theorem, we can see that quasi-transitive digraphs are totally ϕ_1-decomposable. Since the transitive digraph for the nonstrong quasi-transitive digraphs is acyclic $T \in \phi_1$ and each H_i is itself a strong quasi-triansitive digraph, and you can therefore use Theorem 2.2.3 again. For the strong quasi-transitive digraphs D, S is semicomplete so $S \in \phi_1$ and each $Q_i \in \phi_1$ because it is either one vertex which is a digraph that is both acyclic and semicomplete or it is non-strong and must be quasi-transitive and therefore Theorem 2.2.3 can be used agian. So every nonstrong and strong quasi-transitive digraphs is totally ϕ_1-decomposable. Could not find a therom, lemma or anything else so i made my own corollary.

Corollary 2.2.3.1. *quasi-transitive digraphs D are totally ϕ_1-decomposable and you can find the decomposition in polynomial time.*

The polynomial time comes from Theorem 2.1.2 since it is totally ϕ_i-decomposable where $i = 1$.

2.3 Locally Semicomplete Digraph

Every locally semicomplete digraph can be classified into some other groups of digraphs, namely semicomplete digraphs and round decomposable digraphs and the last one which is neither of the two is called evil (a name first introduced in [3] and will be defined more precisely towards the end of this section). Round-decomposable digraph $D = R[D_1, \ldots, D_r]$ is where R is a round digraph of the strong semicomplete digraphs D_i and $|R| = r$. In section 1.3 there was a briefly definition of round digraph for a more formal definition we use the definition from [4].

Definition 2.3.1. *[4] A digraph on n vertices is round if we can label its vertices $v_1, \ldots, v_n$ so that for each i, we have $N^+(v_i) = \{v_{i+1}, \ldots, v_{i+d^+(i)}\}$ and $N^-(v_i) = \{v_{i-d^-(i)}, \ldots, v_{i-1)}\}$. We call the labeling $v_1, \ldots, v_n$ a round ordering.*

It turns out the class of locally semicomplete digraph is split up in these 3 subclasses and these subclasses are going to be important for proving a lot of the problems we are going to cover in this thesis.

Theorem 2.3.1. *[5, 3] Let D be a locally semicomplete digraph. Then exactly one of the following possibilities holds.*

(a) D is round decomposable with a unique round decomposition $R[D_1, \ldots, D_r]$, where R is a round local tournament on $r \geq 2$ vertices and D_i is strong semicomplete digraph for $i = 1, 2, \ldots, r$.

(b) D is evil.

(c) D is a semicomplete digraph that is not round decomposable.

Furthermore, there is a polynomial algorithm that decides which of the properties holds and gives a certificate for this.

If the locally semicomplete digraph is nonstrong, it turns out that it is decomposable. This is called a semicomplete decomposition.

Theorem 2.3.2. *[3, 1, 6] Let D be a nonstrong locally semicomplete digraph and let $D_1, D_2, \ldots, D_p$ be the acyclic order of the strong components of D. Then D can be decomposed into $r \geq 2$ disjoint subdigraphs $D'_1, D'_2, \ldots, D'_r$ as follows:*

$$D'_1 = D_p, \qquad \lambda_1 = p,$$
$$\lambda_{i+1} = min\{j | N^+(D_j) \cap V(D'_i) \neq \emptyset\},$$

and

$$D'_{i+1} = D\left\langle V(D_{\lambda_{i+1}}) \cup V(D_{\lambda_{i+1}+1}) \cup \cdots \cup V(D_{\lambda_i - 1})\right\rangle.$$

The subdigraphs $D'_1, D'_2, \ldots, D'_r$ satisfy the properties below:

(a) D'_i consists of some strong components that are consecutive in the acyclic ordering of the strong components of D and is semicomplete for $i = 1, 2, \ldots, r$;

(b) D'_{i+1} dominates the initial component of D'_i and there exists no arc from D'_i to D'_{i+1} for $i = 1, 2, \ldots, r-1$;

Figure 2.1: (a),(b) and (c)

(c) if $r \geq 3$ then there exists no arc between D'_i and D'_j for i, j satisfying $|j - i| \geq 2$

For simplification of Theorem 2.3.2 the properties are drawn out in Figure 2.1 If D is a locally semicomplete digraph that is not semicomplete, it can still be strong and if this is the case, we can find a minimum seperator S. Since a seperator S make $D - S$ a non-strong digraph, we can make a semicomplete decomposition out of $D - S$.

Theorem 2.3.3. *[1] If a strong locally semicomplete digraph D is not semicomplete, then there exists a minimal seperating set $S \subseteq V$ such that $D - S$ is not semicomplete. Furthermore, if $D_1, D_2, \ldots, D_p$ is the acyclic ordering of the strong components of $D - S$ and $D'_1, D'_2, \ldots, D'_r$ is the semicomplete decomposition of $D - S$, then $r \geq 3$, $D\langle S\rangle$ is semicomplete and we have $D_p \mapsto S \mapsto D_1$.*

Some of them are round-decomposable and we will later see that it is the when $r > 3$ in the semicomplete decomposition. It also turns out that this round-decomposistion is unique.

Corollary 2.3.3.1. *[5] If a locally semicomplete digraph D is round decomposable, then it has a unique round decomposition $D = R[D_1, D_2, \ldots, D_\alpha]$.*

From Theorem 2.3.1 we know that for a round-decomposable digraph the quotient graph is round and the houses are semicomplete making them totally ϕ_2-decomposable then by Theorem 2.1.2 we know that we can find the decomposition in polynomial time.

Propersition 2.3.4. *[5] There exists a polynomial algorithm to decide whether a given locally semicomplete digraph D has a round decomposition and to find this decomposition if it exists.*

Like we briefly meantioned above the locally semicomplete digraph that are not semicomplete and not round have a semicomplete decomposistions on where $r = 3$ also those we call evil. The evil locally semicomplete digraphs are the ones we are focusing on for the rest of this section.

Lemma 2.3.5. *[5] Let D be a strong locally semicomplete digraph which is not semicomplete. Either D is round decomposable, or D has a minimal seperating set S such that the semicomplete decomposition of $D - S$ has exactly three components D'_1, D'_2, D'_3.*

There is a good understanding of the structure of round-decomposable and the semicomplete digraphs, even the semicomplete decomposition which is a part of the evil structure too. We are now going to construct then use this to construct what we call a **good** seperator.

Lemma 2.3.6. *[3] Let S be a minimal seperator of the locally semicomplete digraph D. Then either $D\langle S\rangle$ is semicomplete or $D\langle V - S\rangle$ is semicomplete.*

Then a **good** seperator of a locally semicomplete digraph is minimal and $D\langle S\rangle$ is semicomplete. When finding a good seperator in an evil locally semicomplete digraph, then the part that is left $D - S$ is a non-strong locally semicomplete digraph and we can therefore

use Theorem 2.3.2 to find the semicomplete decomposition of $D\langle V - S\rangle$, it turns out that there is a lot to say about this decomposition. With this decomposistion, we can classify the quotient graph but we can try to describe more deeply how it looks.

Theorem 2.3.7. *[3, 6] Let D be an evil locally semicomplete digraph then D is strong and satisfies the following properties.*

(a) There is a good seperator S such that the semicomplete decomposition of $D - S$ has exactly three components D'_1, D'_2, D'_3 (and $D\langle S\rangle$ is semicomplete by Theorem 2.3.6);

(b) Furthermore, for each such S, there are integers α, β, μ, ν with $\lambda_2 \leq \alpha \leq \beta \leq p-1$ and $p+1 \leq \mu \leq \nu \leq p+q$ such that

$$N^-(D_\alpha) \cap V(D_\mu) \neq \emptyset \text{ and } N^+(D_\alpha) \cap V(D_\nu) \neq \emptyset, \tag{2.3}$$

$$\text{or } N^-(D_\mu) \cap V(D_\alpha) \neq \emptyset \text{ and } N^+(D_\mu) \cap V(D_\beta) \neq \emptyset, \tag{2.4}$$

where $D_1, D_2, \ldots, D_p$ and $D_{p+1}, \ldots, D_{p+q}$ are the strong decomposition of $D - S$ and $D\langle S\rangle$, respectively, and $D_{\lambda_2}$is the initial component of D'_2.

Even though this is a structure we can work with, we can actually go deeper into the structure of this evil locally semicomplete digraph. Namely trying to group the components inside the semicomplete decomposition D'_1, D'_2, D'_3 and the good seperator S. This structure is mentioned in [3] but also in [7]. First we can establish lemma, which is a big part of the structure of evil locally semicomplete digraphs.

Lemma 2.3.8. *[7] Let D be an evil locally semicomplete digraph and let S be a good seperator of D. Then the following holds:*

(i) $D_p \Rightarrow S \Rightarrow D_1$.

(ii) If sv is an arc from S to D'_2 with $s \in V(D_i)$ and $v \in V(D_j)$, then

$$D_i \cup D_{i+1} \cup \ldots D_{p+q} \Rightarrow D_1 \cup \cdots \cup D_{\lambda_2 - 1} \Rightarrow D_{\lambda_2} \cup \cdots \cup D_j.$$

(iii) $D_{p+q} \Rightarrow D'_3$ and $D_f \Rightarrow D_{f+1}$ for $f \in [p+q]$, where $p+q+1 = 1$.

(iv) If there is any arc from D_i to D_j with $i \in [\lambda_2 - 1]$ and $j \in [\lambda_2, p-1]$, then $D_a \Rightarrow D_b$ for all $a \in [i, \lambda_2 - 1]$ and $b \in [\lambda_2, j]$.

(v) If there is any arc from $D_k to D_l$ with $k \in [p+1, p+q]$ and $l \in [\lambda_2 - 1]$, then $D_a \Rightarrow D_b$ for all $a \in [k, p+q]$ and $b \in [l]$.

Chapter 3

Path Cover and Hamilton Cycles

In this chapter, the focus is the Hamilton cycle problem, where we know that if we can solve the path covering problem, then we can solve the Hamilton cycle problem for quasi-transitive digraphs.
In the first section we are going to cover what a Hamilton path is, and a Hamilton cycle Since these to problems are well known as **NP-Complete** which will shortly be introduced too.
The next section is about path-mergeable digraphs and that locally semicomplete digraphs are a subclass of these, and how this helps in the path covering problem and Hamilton cycle problem. The following section is covering quasi-transitive digraphs and whether or not there exists a Hamilton cycle in those. Here, the decomposition of the quasi-transitive digraphs is going to be a crucial part of proving this.

3.1 The Hamilton Path and Cycle Problem

The Hamilton cycle problem in a digraph is well known, but here is a short explanation of what that is. When we define what a Hamiltonian digraph is, we first have to explain what a Hamilton cycle is. A Hamilton cycle is a directed cycle C_H in a digraph that contains every vertex in the digraph $\forall v \in V(D),\ v \in C_H$.

Definition 3.1.1. *A Hamiltonian digraph is a graph containing a Hamilton cycle.*

We can also define digraphs called traceable digraphs:

Definition 3.1.2. *A traceable digraph is a digraph containing a Hamilton path.*

A Hamilton path is a path containing all vertices of the digraph.
It is NP-complete to find out whether an arbitrary given digraph is traceble or Hamiltonian. Before going into why the problems are NP, we are going to state some obvious conditions for graphs to be traceable or Hamiltonian.
For a digraph to be traceable, it needs to be semi-connected, and for a digraph to be Hamiltonian, it needs to be strong. Since a Hamilton cycle is a cycle factor in a digraph D hence a condition for a digraph to be Hamiltonian is that it need to contain a cycle factor.

This is all explained and proven in the book 'Digraph', written by Bang-Jensen and Gutin [1].

3.2 Hamiltonian Locally Semicomplete Digraphs

Recall that a locally semicomplete digraph is both locally in-semicomplete and locally out-semicomplete. Before this gets relevant, we are going to introduce a class of digraphs called path-mergeable.
A definition of path mergeable digraphs: is if there exists a pair of distinct vertices $x, y \in V(D)$ and any two disjoint (x, y)-paths, there exists a new path from x to y a union of the vertices used in the two vertex-disjoint paths (ending up with a "merge" path of the two given path).
These digraphs are easy to recognize with the following corollary. We can do it in polynomial time too, and the following theorem gives us a nice property of path-mergeable digraphs.

Corollary 3.2.0.1. *[1] Path-mergeable digraphs can be recognized in polynomial time.*

Theorem 3.2.1. *[1] A digraph D is path mergeable if and only if for every pair of distict vertices $x, y \in V(D)$ and every pair $P = xx_1 \dots x_r y$, $P' = xy_1 \dots y_s y$, $r, s \geq 1$ of internally disjoint (x, y)-paths in D, either there exists an $i \in \{1, \dots, r\}$, such that $x_i \to y_1$, or there exists a $j \in \{1, \dots, y_j \to x_1\}$.*

Theorem 3.2.1 tells us, that for every path mergeable digraph in every two disjoint (x, y)-path there has to be from one of the path a vertex that dominates the first vertex after x in the other path. This has to hold for every distict pair of vertices x and y.
It turns out that in these digraphs, we can easily determine whether it is a Hamiltonian digraph.

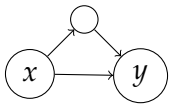
Theorem 3.2.2. *[8] A path-mergeable digraph D of order $n \geq 2$ is Hamiltonian if and only if D is strong and $UG(D)$ is 2-connected.*

Corollary 3.2.2.1. *[8] There is an $O(nm)$-algorithm to decide whether a given strong path-mergeable digraph has a Hamiltonian cycle and find one if it exists.*

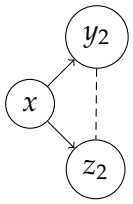
So it turns out that for path-mergeable digraphs this problem is polynomial solvable, and a subclass of these path-mergeable digraph is namely the locally semicomplete digraphs. If we can prove this, we do not only know that we can solve the Hamilton cycle in polynomial time. Since the locally semicomplete digraphs is a special subclass of path-mergeable digraph we have an even better time for these.

Propersition 3.2.3. *[8] Every locally in-semicomplete (out-semicomplete) digraph is path-mergeable.*

Proof. First we prove it is true for out-semicomplete and there after for in-semicomplete digraphs. So letbus assume that D is an out-semicomplete digraph, and we take x and y where we say that these two have 2 vertex disjoint (x, y)-paths called P and Q.
Let $P = y_1 y_2 \dots y_k$ and $Q = z_1 z_2 \dots z_s$ where $y_1 = x = z_1$ and $y_k = y = z_s$. We want to show that there exists a path R where $V(R) = V(P) \cup V(Q)$, if $|A(P)| + |A(Q)| = 3$. It is clear from Figure 3.1a that we can just choose the longest of the paths and we have all vertices included from both paths. So we assume that $|A(P)| + |A(Q)| \geq 4$, and ince D



(a) A visual example of two vertex disjoint paths P and Q where $|A(P)| + |A(Q)| = 3$.



(b) Clearly, from the definition of out-semicomplete, the dashed line needs to be an arc in either direction or both directions.

is out-semicomplete we know that either $y_2 \to z_2$ or $z_2 \to y_2$ or both has to be true. For confirmation see Figure 3.1b. This must be true for every pair of vertices x and y where there are two distict (x, y)-paths. The rest of this part of the proof is from Theorem 3.2.1, which conludes the proof for out-semicomplete digraph.

Now suppose that D is in-semicomplete. Then reversing the arcs will make it out-semicomplete. Denote this digraph D-reverse. Now for two distict vertices x and y where there exists two distinct (x, y)-path P and Q in D, then in D-reverse there must exist two distinct (y, x)-paths P-reverse Q-reverse.
Since D-reverse is out-semicomplete, we can find a path R where $V(R) = V(P\text{-reverse}) \cup V(Q\text{-reverse})$ in D-reverse. Then in D we have a (x, y)-path R-reverse where $V(R\text{-reverse}) = V(P) \cup V(Q)$. Making every in-semicomplete digraph path-mergeable. □

It turns out that Theorem 3.2.2 and Corollary 3.2.2.1 can be improved if we are only looking at the locally in-semicomplete digraphs, since the locally semicomplete digraphs are a subclass of these, and it is the ones we are interested in, in this thesis. It turns out that every strong in-locally semicomplete digraph has a 2-connected underlying graph, which means the only thing we need to check is whether it is a strong digraph.

Theorem 3.2.4. *[9] A locally in-semicomplete digraph D of order $n \geq 2$ is Hamiltonian if and only if D is strong.*

It turns out that when looking at the strong locally in-semicomplete digraphs out of the path-mergeable digraph finding the Hamiltonian cycle can be done in polynomial time by a theorem discovered by Bang-Jensen and Hell in [10].

Theorem 3.2.5. *[10] There is an $O(m + n \log n)$-algorithm for finding a Hamiltonian cycle in a strong locally in-semicomplete digraph.*

This ends the section about Hamiltonian locally semicomplete digraphs, now we want to know about traceable locally semicomplete digraphs.
First we need to know that an **out-branching** for a digraph D is where for one specific vertex, the root, have arcs only going out of it, and for all other vertices they have only one arc going in and the number of arcs going out is ≥ 0. An in-branching is the above explaination where all arcs are reversed.

Lemma 3.2.6. *[1] Every connected locally in-semicomplete digraphs D has an out-branching*

Theorem 3.2.7. *[9] A locally in-semicomplete digraph is traceable if and only if it contains an in-branching.*

This also means that reversing the arcs, that a locally out-semicomplete digraph is traceable if and only if it contains an out-branching. If this out-branching or in-branching exists for locally out-semicomplete or locally in-semicomplete digraphs respectively, we want to find the longest path in these and then we have the wanted Hamilton path.

Theorem 3.2.8. *[10] A longest path in a locally in-semicomplete digraph D can be found in time $O(m + n \log n)$.*

And again, this is also true for locally out-semicomplete digraphs. Connecting Theorem 3.2.7 and Theorem 3.2.6, we know that a locally semicomplete digraph is both locally in-semicomplete, meaning from Theorem 3.2.6 it contains an out-branching if it is connected, and it is locally out-semicomplete. So if it contains an out-branching it is traceable.

Theorem 3.2.9. *A locally semicomplete digraph has a Hamiltonian path if and only if it is connected.*

The hamilton path can be found in time $O(m + n \log n)$ from Theorem 3.2.8.

3.3 Hamiltonian Quasi-transitive Digraphs

First of all, we have to recall Theorem 2.2.3 since it is the key theorem to solve the Hamilton cycle problem in polynomial time.
Remember that a condition for a digraph to be Hamiltonian is that it need to be strong, so for finding a Hamilton cycle in a quasi-transitive digraph, we are not interested in the non-strong digraphs. Leaving only the strong quasi-transitive digraphs with decomposition $S[Q_1, \dots Q_s]$ from Theorem 2.2.3. The given decomposition of a strong quasi-transitive digraph has a semicomplete digraph as the quotient. This is why we need some insight to these before the main solution in this subsection can be proven. Another composition of semicomplete digraphs is the extension of these, called extended semicomplete digraphs. An extension of a digraph is a composition of the given digraph S where the houses of the composition is either a single vertex or independence sets.
Before we explain when we can find a Hamilton cycle in strong quasi-transitive digraphs, we need to recall what a cycle factor is. From section 1.1, we shortly explained that a cycle factor is when we can find $C_1, \dots C_k$ cycles in D contaning all vertices of D.

Theorem 3.3.1. *[11] An extended semicomplete digraph D is Hamiltonian if and only if D is strong and contains a cycle factor. One can check whether D is Hamiltonian and construct a Hamilton cycle of D (if one exsists) in time $O(n^{2.5})$.*

Theorem 3.3.2. *[8] A strong quasi-transitive digraph D with a canonical decomposition $D = S[Q_1 \dots, Q_s]$ is Hamiltonian if and only if it has a cycle factor $\mathcal{F}$ such that no cycle of $\mathcal{F}$ is a cycle of some Q_i.*

Proof. Since a Hamiltonian cycle needs to cover all vertices in a digraph, we know that it must cross every Q_i. Moreover the Hamilton cycle is a cycle factor not fully contained in any Q_i. So we only need to show that if we have a cycle factor $\mathcal{F}$, where no cycle is in any Q_i, then D is Hamiltonian. $\forall i\ V(Q_i) \cap \mathcal{F} = \mathcal{F}_i$, there can not be any cyrcle in this and since every vertex is in $\mathcal{F}$, all vertices in Q_i must be contained in $\mathcal{F}_i$ and there is no cycle contained in $\mathcal{F}_i$ which makes it a path factor of Q_i.
Figure here
For all paths in $\mathcal{F}_i$, we make a path contraction. After contraction or before we delete

the remaining arcs if this is done before its the arcs going from the end of a path to a beginning of another path. This action will make Q_i an independent set $\forall i \in [s]$. Since S is a semicomplete digraph, the digraph D would then because of the independence of each Q_i after the path contractions be an extended semicomplete digraph S'. Since we have only made path contractions along the cycles in the cycle factor of D and not deleted any arcs that are a part of the cycle factor, S' contains a cycle factor. Then by Theorem 3.3.1, we know that S' contains a Hamilton cycle. Adding the deleted arcs in each q_i does not change the fact that it contains a Hamilton cycle. We now insert the paths we contracted instead of a node, this makes the cycle longer but it still contains every vertex in the digraph, given a Hamilton cycle in D. □

A Hamilton path does not have the same condition for a digraphs to be strong meaning we are also interested in the non-strong quasi-transitive digraphs $T[H_1, \ldots, H_t]$. The next theorem is proven in much the same as Theorem 3.3.2.

Theorem 3.3.3. *[8] A quasi-transitive digraph D with at least two vertices and with canonical decomposition $D = R[G_1, G_2, \ldots, G_r]$ is traceable if and only if it has a 1-path-cycle factor $\mathcal{F}$ such that no cycle or path of $\mathcal{F}$ is completely in some $D\langle V(G_i)\rangle$.*

The proof of Theorem 3.3.2 shows that there is a correlation between the cycle factor of D a quasi-transitive digraph and the path cover of Q_i and therefore we have correlation between the path cover of each Q_i and the hamilton path or cycle. to further explore that we have the following theorem. but first we need to establish some notation, $pc(D)$ is the path covering number of a digraph D.

Theorem 3.3.4. *Let D be a strong quasi-transitive digraph with canonical decompostion $D = S[Q_1, Q_2, \ldots, Q_s]$. Let $n_1, \ldots n_s$ be the size of the digraphs $Q_1, Q_2, \ldots, Q_s$ respectively. Then D is hamiltonian if and only if the extended semicomplete digraph $S' = S[\overline{K}_{n_1}, \ldots, \overline{K}_{n_s}]$ has a cycle subdigraph which covers at least $pc(Q_j)$ vertices of $\overline{K}_{n_j}$ for every $j = 1, 2, \ldots, s$.*

Proof. Supose D has a hamilton cycle H. $V(Q_j) \cap H$ is a k_j-path factor $\mathcal{F}_j$ of Q_j. $pc(Q_j)$ is the minimum number of path covering Q_j so $k_j \geq pc(Q_j)$. For every $j = 1, 2, \ldots, s$ we use the path contraction of $\mathcal{F}_j$ and delete the remmaining arcs in Q_j results as argued in the proof of Theorem 3.3.2 in a extended semicomplete digraph S' having at least $pc(Q_j)$ vertices in $\overline{K}_{n_j}$ for every $j = 1, 2, \ldots, s$.
We are now going to show that the other way. Suppose S' contains a cycle subdigraph $\mathcal{L}$ containing $p_j \geq pc(Q_j)$ vertices of $\overline{K}_{n_j}$ for every $j = 1, 2, \ldots, s$. By **??** S' has a cycle C containing all vertices of the cycle subdigraph $\mathcal{L}$. $V(C) = V(\mathcal{L})$. Hence there is a cycle C covering p_j vertices of each Q_j. Since $p_j \geq pc(Q_j)$ there is a p_j-path factor in Q_j for every $j = 1, 2, \ldots, s$. We then replace the vertices of p_j of $\overline{K}_{n_j}$ with the p_j-path factor in C, we do this for all $j = 1, 2, \ldots, s$ constructing C'. Since replacing a vertex in a cycle with a path is still a cycle we have that C' is a cycle and is the hamilton cycle of D. □

We know that the canonical decomposition of a quasi-transitive digraph can be found in polynomial time. We can also find the Hamilton cycle in a quasi-transitive digraph in polynomial time, but also verify if it does not exists for the given graph. This result was proved by Gutin. And idear behind it comes from the proof **??** finding the longest cycle in an extended semicomplete digraph from a cycle subdigraph takes time $O(n^3)$ and with the extra operation of finding the coresponding p_i-path cover of the houses Q_i we see this is the case.

Theorem 3.3.5. *[12] There is an $O(n^4)$ algorithm which, given a quasi-transitive digraph D, either returns a Hamiltonian cycle in D or verifies that no such cycle exists.*

Part II

Linkage and Weak Linkage

Chapter 4

Disjoint Paths in Decomposable Digraphs

In this chapter we will cover the linkage problem given a decomposable digraph D and a set of terminals Π to be linked. We will explain the problem and shortly show why the problem is NP-complete. In section 4.2 we will show that the linkage problem is polynomially solvable in ϕ-decomposable digraphs if ϕ adheres to sertaint properties. After that we will show that ϕ_1 also adheres to these properties, meaning that the linkage problem is solveable for quasi-transitive digraphs (all of this is only true if the number of pairs we want to link is fixed). Then in section 4.3 we will cover the 3 different subclasses a locally semicomplete digraph can be a part of, and solve the k-linkage problem for those in polynomial time for a fixed k.

4.1 The Linkage Problem

Given a digraph D and two distinct vertices s and t, we want to make a path from s to t denoted by P. Recall that in this case s will be the source of P and t the sink. Now let $s_1, t_1, \ldots, s_k, t_k$ be distinct vertices of D, then the **k-linkage** problem is to decide if there exist paths $P_1, \ldots, P_k$ linking the vertices so P_i link the pair (s_i, t_i) $\forall i \in [k]$ and each path P has to be vertex disjoint from the others. This problem is also sometimes just called **the linkage problem**. The problem is actually NP-complete already when $k = 2$, so we will be focusing on the problem when k is fixed. The vertices $s_1, t_1, \ldots, s_k, t_k$ are called **terminals** and $(s_1, t_1); \ldots; (s_k, t_k)$ are called **terminal pairs**. The notation for this problem in this thesis would be using k as the natural number of pairs of terminals, and the set of these terminals is denoted $\Pi = \{(s_1, t_1); \ldots; (s_k, t_k)\}$. As we have done up till now we will still use D as the main digraph we are looking at unless anything else is specified. L is used as a collection of paths $P_1, \ldots, P_l$. If L is the solution to our linkage problem it means $l = k$ and the path P_i links the pair (s_i, t_i) for all $i \in [k]$. If L upholds the above conditions we say that L is a Π-linkage, or L is the linkage of (D, Π).
Recall that a quasi-transitive digraph is constructed by either a transitive acyclic digraph or semicomplete digraph as the quotient of the decomposition. And for these two classes of digraphs, we can solve the k-linkage problem in polynomial time for a fixed k. Fixed k means that for an algorithm given a digraph and a natural number k can solve the k-linkage problem in polynomial time (it is possible that the algorithm needs more information). So

when calculating the running time k is treated as a constant. When k is not fixed then it is already NP-complete for tournaments. Since tournaments are a very strict class, we will only focus on when k is fixed.

4.2 Solving the Linkage Problem in ϕ-decomposable Digraphs

From Theorem 2.2.3, we know that a quasi-transitive digraph is a composition of acyclic transitive digraphs and semicomplete digraphs. We know that ϕ_1 is the union of acyclic and semicomplete digraphs, which means that every quasi-transitive digraph is ϕ_1-decomposable as described in chapter 2.

Theorem 4.2.1. *[3] For every fixed k, there exists a polynomial algorithm for the k-linkage problem on acyclic digraphs.*

Theorem 4.2.2. *[13] For every fixed k, there exists a polynomial algorithm for the k-linkage problem on semicomplete digraphs.*

Note that this means that there exists polynomial algorithm for a fixed k to solve the k-linkage problem for digraphs in ϕ_1.
For a decomposition $D = S[M_1, \dots M_s]$ and a set of terminal pairs, we can split the set into two different sets of terminals. The set of **internal pairs** Π_i, where internal pair means that both s_i and t_i is in the same house, and the set of **external pairs** Π_e which is the rest such that $\Pi = \Pi_i \cup \Pi_e$.

Lemma 4.2.3. *[3] Let $D = S[M_1, \dots, M_s]$ be a decomposable digraph and Π a set of pairs of terminals. Then (D, Π) has a linkage if and only if it has a linkage whose external paths do not use any arc of $D\langle M_i \rangle$ for $i \in [s]$.*

Proof. One direction is trivial since a linkage where external paths uses no arcs inside any house is still a (D, Π)-linkage. So now we assume that L is a (D, Π)-linkage that uses the smallest amount of vertices possible. We claim that no external path of L uses any arcs inside any house. Now we assume that if this is not the case, then there must exist a path $P \in L$ where an arc uv of P is contained in a house $uv \in A(P) \cap A(D\langle M_i \rangle)$ for some $u, v \in V(P)$ and some $i \in [s]$.
Since P is external, there is at least one vertex outside the house, ($z \in V(P) - V(M_i)$) either zu or vz is an arc of P. Without loss of generality say vz is the arc, then since v and u are in the same house $uz \in A(D)$and we can make $P' = P - \{uv, vz\} + uz$, Then we can construct a new linkage $L' = L - P + P'$ which indeed is a (D, Π)-linkage with $V(L') < V(L)$, which is a contradiction since L was supposed to be the linkage with the smallest number of vertices. (for formality say zu was the arc then $P' = P - \{zu, uv\} + zv$ and $L' = L - P + P'$a (D, Π)-linkage where $V(L') < V(L)$). □

This means that the external paths do not use arcs inside the houses, only the arcs between the houses (arcs from the quotient digraph S). Be awere that internal pairs can be linked by an internal path or an external path going out of the house and later in again, where of course external pairs have to be linked by external paths.

Before getting into the algorithm for solving the k-linkage problem for ϕ-decomposable digraphs, we need to set some conditions for the set ϕ. When a set of digraphs ϕ upholds

these conditions, we are going to say that ϕ is a linkage ejector. But first we need to establish that a set of digraphs can be closed with respect to blow-up. **Blow-up** means replace a vertex v, with a digraph K (Replacing v with the digraph K). When a set of digraphs ϕ is closed with respect to this operation it means that for a digraph $D \in \phi$ there exists a digraph K such that after K has replaced v the digraph is still a part of the set ϕ. This definition brings this nice lemma.

Lemma 4.2.4. *If a class ϕ is closed with respect to the blowing-up operation $S \in \phi$ and $D = S[M_1, \dots M_s]$, then it is possible to replace the arcs in the digraph M_i with other arcs, so that the resulting digraph is in ϕ.*

This brings us to the definition of a linkage ejector. This definition is a reformulation of the one given in article [3].

Definition 4.2.1. *[3] A class of digraphs ϕ that is closed with respect to blow-up is a linkage ejector if the following conditions are true:*

1. *There exists a polynomial algorithm $\mathcal{A}_\phi$ to find a total ϕ-decomposition of every totally ϕ-decomposable digraph.*
2. *There exists a polynomial algorithm $\mathcal{B}_\phi$ for a fixed k, for solving the k-linkage problen on ϕ.*
3. *There exists a polynomial algorithm $\mathcal{C}_\phi$ that given a totally-decomposable digraph $D = S[M_1, \dots, M_s]$ constructs a digraph of ϕ by replacing the arcs inside each M_i for $i \in [s]$ as in Theorem 4.2.4.*

Now we are going to introduce the theorem that proves that for some classes ϕ, we can solve the k-linkage problem in polynomial time.

Theorem 4.2.5. *Let ϕ be a linkage ejector. For every fixed k, there exists a polynomial algorithm to solve the k-linkage problem on totally ϕ-decomposable digraphs.*

The proof of this theorem is obviously the algorithm $\mathcal{M}$ which we state in Algorithm 1, but to be sure it does what it is supposed to, we are in the proof going to prove that it works and that it solves the problem in polynomial time for a fixed k. Before proving Algorithm 1 we are going to explain the steps in the algorithm in more depth. Since the algorithm for arc-disjoint path (the weak k-linkage) Algorithm 2 involves more complicated steps, but is overall much like Algorithm 1. The biggest difference is that Algorithm 2 need to keep track of the arcs already used. So the example in section 5.2 would hardly be as if we had an example for this algorithm.

Step 1-3 Are the trival steps in the algorithm if there are no pair of terminals we are already done.

Step 4-7 Calls an algorithm to find the decomposition of the digraph D. If it is trivial we call the algorithm $\mathcal{B}_\phi$ which solves the problem and we are done (at least with this part of the graph, since it could be a recursive call).

Step 8 We split the pairs into internal and external pairs compared to the decomposition we found in step 4.

Procedure 1 Algorithm $\mathcal{M}$ for k disjoint paths

Input: Digraph D, a set of terminal pairs Π.
Output: Either "NO" or "YES"

if $\Pi = \emptyset$ **then**
  output YES
end if
Run $\mathcal{A}_\phi$ to find a total ϕ-decomposition of $D = S[H_1, \ldots, H_s]$.
if this decomposition is trivial that is $D \in \phi$ **then**
  Run $\mathcal{B}_\phi$ to solve the $(D,)\Pi)$-linkage.
end if
Let $\Pi^e \subset \Pi$ ($\Pi^i \subset \Pi$) be the list of external (internal) pairs $(s_q, t_q) \in \Pi$.
Assume that $M_1, \ldots, M_l$ is the houses with the internal pairs.
for every partion of $\Pi^i = \Pi_1 \cup \Pi_2$, look for external paths linking the pairs in $\Pi^e \cup \Pi_1$ and internal pairs in Π_2 **do**
  if $\Pi^e \cup \Pi_1 = \emptyset$, then for $i = 1, \ldots, l$: **then**
    run $\mathcal{M}$ recursively on input $(D\langle M_1\rangle, \Pi_2 \cap (V(M_1) \times V(M_1))), \ldots, (D\langle M_l\rangle, \Pi_2 \cap (V(M_l) \times V(M_l)))$.
    if all are linked **then**
      output YES
    end if
  end if
  if $\Pi^e \cup \Pi_1 \neq \emptyset$ **then**
    for each possible choice of l vertex sets $(V_1, \ldots V_l)$ and nonnegative numbers $n_1, \ldots, n_l \leq k$ such that $|V_i| = n_i$ and $V(\Pi^e \cup \Pi_1) \cap V(M_i) \subseteq V_i \subseteq V(M_i) - V(\Pi_2)$ **do**
      let $S' \in \phi$ be the the result of running the algorithm $\mathcal{C}_\phi$ on $S[I_{n_1}, \ldots, I_{n_l}, M_{l+1}, \ldots, M_s]$, where I_{n_j} is the digraph on n_j vertices with no arcs ($V(I_{n_j}) = V_j$).
      Run B_ϕ on $(S', \Pi^e \cup \Pi_1)$.
      if $\Pi^e \cup \Pi_1$ is linked **then**
        run $\mathcal{M}$ recursively on input $(D\langle V(M_1) - V_1\rangle, \Pi_2 \cap (V(M_1) \times V(M_1))), \ldots, (D\langle V(M_l) - V_l\rangle, \Pi_2 \cap (V(M_l) \times V(M_l)))$.
      end if
      if These pairs are linked **then**
        output YES
      end if
    end for
  end if
end for
output NO.

Step 9 Here we seperate the houses of the internal pairs from the houses that only contains the external pairs or no pairs at all. We are not intersted in the other houses since we do not need to use any arcs inside any of the houses for the external pairs, we know this from Theorem 4.2.3.

Step 10 Make a partition of the digraphs internal pairs. We pair the one partition of the internal pairs with the external pairs, since internal pairs can have external paths.

Step 11-16 If there is no terminal pairs in the union, we have only internal pairs that we want to link internally. So when linking the pairs we do not need to consider the rest of the digraph but only the house where the terminals we are considering is inside. So we call the algorithm again recursivly on the houses with the internal pairs $M_1, \ldots, M_l$, and the terminals from Π_2 that is a part of that house $\Pi_2 \cap (V(M_i) \cup V(M_i))$.

Step 17 Now when there are pairs we want to link externally we continue from here.

Step 18-20 We make some smaller set of vertices inside the houses, we do not need any arcs inside the houses for the external paths. There are k pairs of terminals and therefore at most k external pairs, so we do not need more than max k vertices in each house. Since it does not matter how many we use in the houses without internal pairs we only make smaller vertex sets for the once where we may need some of the vertices for the internal paths and that is only needed in the houses $M_1, \ldots M_l$. Of course we should not have the vertices that are terminals of Π_2 as a part of the new vertex sets but just as we do not need those we have to have the terminals of the ones we try to link $\Pi^e \cup \Pi_1$. Since we do not need any arcs inside these houses, we make independent set out of these vertex sets $V_1, \ldots V_l$ and construct a new digraph S. Then we run $\mathcal{C}_\phi$ such that we know have a digraph $S' \in \phi$ which means we can run $\mathcal{B}_\phi$ on $(S', \Pi^e \cup \Pi_1)$.

Step 21-23 Since D is totally ϕ-decomposable and that is a hereditary property from Theorem 2.1.1, meaning every induced subdigraph of D is totally ϕ-decomposable, so each $D\langle V(M_i) - V_i\rangle$ is totally ϕ-decomposable and we know that when we can call $\mathcal{M}$ it will accept the digraph and return an answer.

Step 24-29 If all external pairs are linked, we go into the if-statement in step 21. and we only return if these are also linked so we can go into this statement and output that a linkage exists. If not then the pairs are not linked and we go to step 30.

Step 30 We have not been able to link the pairs in any partition of the internal pairs and we output that it does not exists.

Proof. We are going to show that the algorithm works by induction on $|V(D)| + k$, where k is the number of terminal pairs.
If $k = 0$ we are done. If $\mathcal{A}_\phi$ returns $D \in \phi$, then since $\mathcal{B}_\phi$ is correct we are done. So we assume $k > 0$ and $D \notin \phi$, then we assume that D has a Π-linkage, and we will show that in every case the algorithm will output yes. Since D has a Π-linkage it means there exists some choice of Π_1, Π_2 and there exists possible choices for $V_1, \ldots, V_l$ such that $\Pi^e \cup \Pi_1$ is linked and by induction hypothesis the Algorithm 1 links $D\langle V(M_i) - V_i\rangle$ for every recursive call on $i \in [l]$. Again, by the induction hypothesis the Algorithm 1 $D\langle M_i\rangle$ for every recursive call on $i \in [l]$. Then in both cases the algorithm output yes.

We only need to prove that the algorithm is polynomial. the running time of the algorithm depends on the size of $n = |V(D)|$ and k the number of terminal pairs to be linked,

the algorithm is polynomial as long as k is fixed. We let $T(n,k)$ be the running time for the algorithm for D we are going to show that $T(n,k)$ is $O(n^{d(k)})$ for some funktion on $d(k)$.
The first steps are obviusely constant. Step 4 is where we find the decomposition it is a polynomial algorithm $\mathcal{A}_\phi$ and finding the the external and internal paths in D in step 8 is also polynomial, so we say that step 4 and 8 combined takes time $O(n^{a(k)})$.
Running $\mathcal{B}_\phi$ is also polynomial. Let this be $O(n^{b(k)})$. We also have a part of the algorithm where we first run $\mathcal{C}_\phi$ followed by $\mathcal{B}_\phi$, both polynomial algorithms, meaning the product is also polynomial. Say this is $O(n^{c(k)})$. Step 11-15 is a recursive call on K_i $\forall i \in [l]$ so step 11-15 takes $\sum_{i=1}^{l} T(n_i,k_i)$ where $n_i = |V(K_i)| < n$ and $k_i = |\Pi_2 \cap (V(K_i) \times V(K_i))| \leq k$ by induction hypothesis each of these recursive calls takes $O(n_i^{d(k_i)})$ so run time is $\sum_{i=1}^{l} n_i^{d(k_i)}$. Since we entered the if statement in step 11 we know that $\sum_{i=1}^{l} k_i = k$ and in worst case $\sum_{i=1}^{l} n_i = n$ such that step 11-15 takes at most $O(n^{d(k)})$.
Step 14-26 Is a bit more tricky, we still have in worst case $\sum_{i=1}^{l} n_i = n$. But first, we need to make the vertex sets V_i $\forall i \in [l]$ which can be of size between 1 and k. In worst case we have to go through all of the possibilities for these vertex sets. For one set V_i we need all the possible combinations $\sum_{j=1}^{k} \binom{n}{j}$. The worst possibility is if all k pairs are internal and all in seperate houses, so we need to create these k times since there is at most k houses and they can be combined in any way $\left(\sum_{j=1}^{k} \binom{n}{j}\right)$. For all these choices of vertex set V_i we have to go through step 19 and 20 which takes $O(n^{c(k)})$ and the recursive calls $\sum_{i=1}^{l} T(n_i,k_i)$. We define $d(k) = k^3 + a(k) + c(k)$ and we are going to show that what ever we come up with is going to be smaller than $O(n^{d(k)})$ which is clearly polynomial.
$\sum_{j=1}^{k} \binom{n}{j} = n^2 - 1$, $(n^2)^k = n^{k^2}$ since we entered step 17 there is at least one pair in $\Pi^e \cup \Pi_1$ meaning in the worst case the recursive calls in step 21-23 is $T(n,k-1)$, so step 17-23 takes $O(n^{k^2})(T(n,k-1) + O(n^{c(k)}))$. We split this up into $O(n^{k^2})T(n,k-1)$ and $O(n^{k^2})O(n^{c(k)})$ $T(n,k-1)$ takes $O(n^{d(k-1)})$ so $O(n^{k^2})T(n,k-1) = O(n^{k^2})O(n^{d(k-1)}) = O(n^{k^2+d(k-1)})$ we are going to show that $k^2 + d(k-1) \leq d(k)$:

$$d(k-1) = (k-1)^3 + a(k-1) + b(k-1) = k^3 - 3k^2 + 3k - 1 + a(k-1) + c(k-1) \quad (4.1)$$

$$k^2 + d(k-1) = k^3 - k^2 + 3k - 1 + a(k-1) + c(k-1) \leq k^3 + a(k) + c(k) = d(k) \quad (4.2)$$

We also need to show that for the other half, $O(n^{k^2})O(n^{c(k)}) = O(n^{k^2+c(k)})$, we definitely have that $k^2 + c(k) \leq d(k)$. So step 17-23 takes $O(n^{d(k)})$ Since there is 2^k choice of partitioning the set Π^i into Π_1 and Π_2 so 10-29 takes $O(2^k n^{d(k)})$ since we treat k as fixed it is considered a constant and the running time of $T(n,k)$ is $O(n^{a(k)}) + O(n^{b(k)}) + O(n^{d(k)}) = O(n^{d(k)})$. □

4.2.1 Linkage for Quasi-transitive Digraph

To prove that for quasi-transitive digraphs we can solve the linkage problem in polynomial time, we just need to prove that ϕ_1 is a linkage ejector. Since extended semicomplete digraphs and other classes are also a part of the totally ϕ_1-decomposable digraphs, this will then also prove that the linkage problem can be solved in polynomial time for these.

Lemma 4.2.6. *[3] The class ϕ_1 is a linkage ejector.*

Proof. First we have to make sure that ϕ is closed with respect to blow-ups. If we blow-up the vertices with a transitive tournament. Then if $D \in \phi_1$ is semicomplete, then since tournaments are semicomplete, D after blow-up is still semicomplete. Then if $D \in \phi$ is

acyclic and the vertices are blown-up by a transitive tournament then it is still acylic since a transitive tournament is acylic, which we shortly prove.
Let us assume that a transitive tournament is not acyclic, then for some acylic ordering $v_1, v_2, \ldots, v_n$ we have what we call a backwards arc which is an arc going back in the ordering. Let us say that the first backwards arc in the ordering goes from v_y. We know that it is transitive, so if $v_z \to v_x$ and $v_x \to v_y$ Then $v_z \to v_y$. Since it is a tournament we have for v_{y-1} that every vertex $v_1, v_2, \ldots v_{y-2}$ dominates v_{y-1}. If not, then we will have a backwards arc from a vertex earlier in the ordering than the first one from v_y, a contradiction. So if v_{y-1} dominates v_y, then by the transitive property $v_1, v_2, \ldots v_{y-2}$ also dominates v_y. So the only way we can have a backwards arc is if v_y dominates v_{y-1} but since $v_1, v_2, \ldots v_{y-2}$ also dominates v_{y-1}, v_y would be earlier than v_{y-1} in the acylic ordering a contradiction. And therefore a transitive tournament is acylic.
This indicates that ϕ_1 is closed to blow-ups if the digraphs that the vertices is blown-up with is a transitive tournament. From Theorem 2.1.2, we have the polynomial algorithm $\mathcal{A}_{\phi_1}$, meaning we only need the function $\mathcal{B}_{\phi_1}$ and $\mathcal{C}_{\phi_1}$.
The algorithm $\mathcal{B}_{\phi_1}$ is a algorithm that determines the k-linkage problem for a fixed k on digraphs in ϕ_1 by Theorem 4.2.1 we have a polynomial algorithm for acyclic digraph and by Theorem 4.2.2 we have a polynomial algorithm for semicomplete digraphs such combining these we have an algorithm for solving the k-linkage problem on a digraph $D \in \phi_1$ meaning we have $\mathcal{B}_{\phi_1}$.
For the last algorithm $\mathcal{C}_{\phi_1}$ it takes for every decomposition $D = S[M_1, M_2, \ldots, M_s]$ each M_i for $i = [s]$ and delete and add arcs so each M_i is a transitive tournament. □

Now we have proved that ϕ_1 is a linkage ejector and in section 2.2 that a quasi-transitive digraph is totally ϕ_1-decomposable such for any quasi-transitive digraph we can use Algorithm 1.

4.3 Solving Linkage Problem in Locally Semicomplete Digraphs

A locally semicomplete digraph is either round decomposable, semicomplete or neither. We have in section 2.3 called these evil locally semicomplete digraph or just evil. The semicomplete part is solved from Theorem 4.2.2 but the theorem will also be important in this section. First we will look at the evil semicomplete digraph where we need to recall Equation 2.3.7 (a) where we can see that an evil semicomplete digraph can be partitioned into into maximum 4 semicomplete digraphs S, D'_1, D'_2, D'_3 which lead us to the next theorem.

Theorem 4.3.1. *[14] For every fixed pair of positive integers c,k there exists a polynomial algorithm for the k-linkage problem on digraphs whose vertex set is partinionable into c sets inducing semicomplete digraphs.*

Let $c = 4$ in Theorem 4.3.1. Then we know from Equation 2.3.7 that every evil locally semicomplete digraph has a polynomial algorithm for the k-linkage problem when k is fixed.

The remaning class of digraphs inside the class of locally semicomplete digraphs is the class of round decomposable digraphs. Recall the class $\phi_2 = \{\text{semicomplete digraphs}\} \cup \{\text{round digraphs}\}$ from section 2.1. As we did in section 4.2, we will in the end prove that ϕ_2 is a linkage ejector and since round decomposable digraphs are totally ϕ_2-decomposable,

we would have proven that there exists a polynomial algorithm for them. To prove that ϕ_2 is a linkage ejector, we know from item 4.2.1 that it needs 3 algorithms $\mathcal{A}_{\phi_2}, \mathcal{B}_{\phi_2}$, and $\mathcal{C}_{\phi_2}$. For the algorithm $\mathcal{B}_{\phi_2}$ we only need it for round digraphs.

Theorem 4.3.2. *For every fixed k, there exists a polynomial algorithm to solve the k-linkage problem on round digraphs.*

Proof. D is round so let $v_1, \ldots, v_n$ be the round ordering and $\Pi = \{(s_1, t_1), \ldots (s_k, t_k)\}$ the set of pairs of terminals. Given $j \in [n-1]$, we say that an arc $v_a v_b$ is **over** another arc $v_j v_{j+1}$ if $v_b \in \{v_j + 1, \ldots v_{a-1}\}$. We are now going to show that for a (s_i, t_i)-path, it only needs to use one arc over $v_j v_{j+1}$. Let us assume this is not the case and that the (s_i, t_i)-path uses two arcs over $v_j v_{j+1}$. Call these two arcs $u_1 w_1$ and $u_2 w_2$. There are four ways these vertices can be placed in relation to each other in the ordering and still be arcs over $v_j v_{j+1}$. See figure lav figur. Let us say without loos of generality that the (s_i, t_i)-path first the arc $u_1 w_1$ and then later $u_2 w_2$. We can in all cases of constellations of the vertices mentioned in ref figur blalbalbal make the (s_i, t_i)-path shorter by use of other arcs. If we are in either case 1 we can make the path shorter by using the arc $u_1 u_2$ and therefore not need to use the arc $u_1 w_1$, since $u_1 u_2$ is not an arc over $v_j v_{j+1}$ the (s_i, t_i)-path only uses one arc over. If we instead have the constellation in case 2, case 3 and case 4, we can use the arc $u_1 w_2$ which is an arc over $v_j v_{j+1}$, but it means that the path does not use any of the two over arcs $u_1 w_1$ or $u_2 w_2$. This proves that any (s, t)-path only needs to use max one arc over $v_j v_{j+1}$. Hence each path in the k-linkage only uses maximum k arcs over $v_j v_{j+1}$. We also know that deleting all arcs over $v_j v_{j+1}$, we get an acyclic digraph and from Theorem 4.2.1 that we can solve the k-linkage problem in polynomial time on this digraph.
Since the digraph is not acylic, some of the paths can and may need to use an arc over $v_j v_{j+1}$ so we make a combination of some of the pairs, not nessesary all in order, so we rename the h chosen pairs $(s_{i_1}, t_{i_1}), \ldots, (s_{i_h}, t_{i_h})$ where $0 \leq h \leq k$ where i_k is a function mapping i_1 to $z \in [k]$ the first chosen piar of the k terminal piars.
These are the pairs we predict uses the arcs $\{u_1 w_1, \ldots, u_h w_h\}$ which are all arcs over $v_j v_{j+1}$. Then we construct D' by deleting all arcs over $v_j v_{j+1}$ then we have the $2h$-linkage for the pairs $(s_{i_1}, u_1), (w_1, t_{i_1}), \ldots, (s_{i_h}, u_h), (w_h, t_{i_h})$ and the rest of the original pairs $k - h$-linkage, then we use the algorithm for acyclic digraphs and solve the $k + h$-linkage on D'. Do this for all combinations of h pairs. If there exists a k-linkage in D, there exists some $k + h$-linkage in D' for some combination of h pairs. When the $k + h$-linkage is found, we add the arcs $u_1 w_1, \ldots, u_h w_h$ to the linkage and create a path P_{i_j} from the path (s_{i_j}, u_j)-path then the arc (u_j, w_j) and at last (w_j, t_{i_j})-path which is a (s_{i_j}, t_{i_j})-path. This creates the k-linkage for D. □

The last algortihm will be in the proof of the next theorem which will end the part about round decomposable digraphs.

Theorem 4.3.3. *For every fixed k, there exists a polynomial algorithm to solve the k-linkage problem on round decomposable digraphs.*

Proof. First we know from section 2.3 that round-decomposable digraphs are totally ϕ_2-decomposable. So if ϕ_2 is a linkage ejector then we can use Algorithm 1 to find the k-linkage of a round decomposable digraph. This means all that is left to prove is that ϕ_2 is a linkage ejector, for this we need to prove that ϕ_2 is closed with respect to blow-ups. The semicomplete digraphs we know from the proof of Theorem 4.2.6 that we can blow it up with a transitive tournament. A transitive tournament is also a round digraph, which means that we can blow up a vertex in a round digraph and it is still round. A short argument of

this:
We know from the proof of Theorem 4.2.6 that a transitive tournament is acyclic meaning we have an acyclic ordering of the vertices $v_1, v_2, \ldots, v_n$. Since it is a tournament, we know taht a vertex v_i is dominated by all vertices before in the acylic ordirng $v_1, \ldots v_{i-1} = N^-(i)$ and dominate $v_{i+1}, \ldots v_n = N^+(i)$. This is true for all $i \in [n]$. Thus the acyclic ordering is also the round ordering.
So now we know that ϕ_2 is closed w.r.t. blow-ups as long as it blows-up to a transitive tournament. The algorithm $\mathcal{A}_{\phi_2}$ is covered by Theorem 2.1.2. Now for algorithm $\mathcal{B}_{\phi_2}$ we have for the semicomplete digraphs a polynomial algorithm for the k-linkage problem by Theorem 4.2.2 and for round digraphs we have the algorithm for the k-linkage problem by Theorem 4.3.2, thus combining these theorems we have $\mathcal{B}_{\phi_2}$. The last algorithm $\mathcal{C}_{\phi_2}$ delete and add arcs from each M_i in a decomposition $R[M_1, M_2, \ldots, M_r]$ so it becomes a transitive tournament. This proves that the class ϕ_2 a is likage ejector. □

Now we have an algorithm for all locally semicomplete digraphs and therefore to end this section we have the following theorem:

Theorem 4.3.4. *For every fixed k, there exists a polynomial algorithm to solve the k-linkage problem on locally semicomplete digraphs.*

Chapter 5

Arc-disjoint Paths in Decomposable Digraphs

5.1 The Weak Linkage Problem

This problem is much like the problem we just went through, except instead of linking terminals with vertex disjoint paths, these path only need to be arc disjoint. Which makes the problem appear more likely in digraphs but also harder to control since there are other checks to go through.
Given a set of terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$, finding arc-disjoint paths between each pair is called the weak k-linkage problem. where a terminal pair is a source and a sink in the paths of the solution of the linkage problem.
The weak linkage problem is also NP-complete because the linkage problem is. By vertex splitting, we can make a linkage problem to a weak linkage problem. The notation in this chapter is much like in the last: D as the digraph we are examining and Π is the set of pairs of terminals, where k is the number of pairs of terminals. When we consider the linkage problem for decomposable digraphs, we can have houses with terminals in and some without any terminals. The houses contaning no terminals are called **clean houses**. Then a terminal pair can either be inside the same houses or in different houses. We adopt these definitions from the linkage problem **internal pairs (and paths)** and **external pairs (and paths)**.
Since we are focusing on arcs, then letter F will be the main notation of a set of arcs from the digraph D, ($F \subseteq A(D)$). F is usually used for arcs that we do not want a part of the linkage we are focusing on, mostly because the arcs are already used to link some other pairs. Therefore, when focusing on a vertex out- and in-degree compared to the set F, it is usually bound by the number of pairs already linked. Since the paths do not need to be vertex-disjoint we can end up in the same vertex as another path that links another pair, then we need to somehow ensure that we do not choose the same arc as we did linking the other pair and that is here the set F comes in. This is important since deleteing arcs could change the class the digraph belongs to.

5.2 Weak Linkage in ϕ-decomposable Digraphs

In this section, we need to establish some new properties for the class ϕ. Much like in section 4.2, the class needs to have these proporties to be relevant for solving the weak k-linkage problem.
For an integer c, the class denoted $D(c)$ is a digraph D, where there is added as many parallel arcs, to the already existing arcs in D, then blow-up b vertices where $0 \leq b \leq c$. The digraph that is blown up has to have size $\leq c$.

Definition 5.2.1. *[3] We say that a class of digraphs ϕ is bombproof is there exsists a polynomial algorithm $\mathcal{A}_\phi$ to find a total ϕ-decomposition of every totally ϕ-decomposable digraph and, for every integer c, there exists a polynomial algorithm $\mathcal{B}_\phi$ to decide the weak k-linkage problem for the class*

$$\phi(c) := \bigcup_{D \in \phi} D(c). \tag{5.1}$$

The clean houses (D, Π) actually have an important role, namely that we do not need them for weak linking of the pairs Π in D.

Lemma 5.2.1. *[3] Let D be a digraph, Π a list of k terminal pairs, and $H \subset D$ a clean house with respect to Π. Let D' be the contraction of H into a single vertex h. Then D has a waek Π-linkage if and only if D' has a waek Π-linkage.*

Proof. Maybe prove □

The external pairs do not need the same amount of vertices and arcs inside a house as maybe the internal pairs. It turns out that in [4], we bound the number of vertices for the external paths. The lemma below is a reformulation of the lemma from [4].

Lemma 5.2.2. *Let $D = S[H_1, \dots H_s]$ be a decomposable digraph. Let Π' be a list of h terminal pairs and let F be a set of arcs in D satisfying that $d_F^-(v), d_F^+(v) \leq r, \forall v \in V(D)$. If $(D \backslash F, \Pi')$ has a weak linkage $\mathcal{L} = P_1, \dots, P_h$, then for any external path $P_i \in \mathcal{L}$ we have $|A(P_i \cap H_j)| \leq 2(h + r)$ and $|V(P_i \cap H_j)| \leq 2(h + r)$ for every $j \in \{1, \dots, s\}$.*

As shortly explain above we sometimes have to control the set of arcs already used by F, but as we remove these arcs from the digraph it may longer belong to the class it did before. We therefore need to make sure removale of some arcs from the digraph does not affect that we have an algorithm for the weak linkage if we had one before removing the arcs.

Lemma 5.2.3. *Let $\mathcal{C}$ be a class of digraphs for which there exists an algorithm $\mathcal{A}$ to decide the weak k-linkage problem, whose running time is bounded by $f(n, k)$. Let $D = (V, A)$ be a digraph, Π a list of k pairs of terminals and $F \subseteq V \times V$ such that $D' := (V, A \cup F)$ is a member of $\mathcal{C}$. There exists an algorithm $\mathcal{A}^-$, whose running time is bounded by $f(n, k + |F|)$, to decide whether D has a weak Π-linkage.*

Proof. Let D be the digraph and $F = \{s_1't_1', \dots, s_{k'}'t_{k'}'\}$ be the set of arcs missing in D so $D' = D(V, A \cup F)$ is in the class $\mathcal{C}$. Let the number of arcs in F be denoted by the non-negative number k'. Then we create a set of terminals based on every arc in F by the arcs

tail and head as the pair of terminals in the set $\Pi' = \{(s'_1, t'_1), \ldots, (s'_{k'}, t'_{k'})\}$. We claim that D has a weak Π-linkage if and only if D' has a weak $\pi \cup \Pi'$-linkage. Which will also prove the theorem. First, if D has a weak Π-linkage, we just add the arcs from F as the Π'-linkage resulting in a $\Pi \cup \Pi'$-linkage in D'. For the other way, we assume that D' has a weak $\Pi \cup \Pi'$-linkage deleting the arcs in F we would still have a weak Π-linkage. There are two possibilities: either the linkage Π does not use any of the arcs in F and we can delete them without problems. The second possibility is that the weak Π-linkage uses an arc of F. If this is the case, then let us say it is the arc $s'_i t'_i$. Since (s'_i, t'_i) is a terminal pair of Π' these have to be linked through some other arcs. Since the arc $s'_i t'_i$ is already used, it can't be used again, otherwise it is not a solution for the $(D', \Pi \cup \Pi')$ linkage problem. This means we substitute the path P'_i which links (s'_i, t'_i) with the arc $s'_i t'_i$ in P_j, which links s_j, t_j, which is still a weak $\Pi \cup \Pi'$-linkage in D', we do this for all arcs that are used by the weak Π-linkage in D' then delete all arcs in F and we have the weak Π-linkage in D. □

Now we are going to state the theorem that is used for the existence of the of our main algorithm in this section. This result is found by ... in .... .

Theorem 5.2.4. *Let ϕ be a bombproof class of digraph. There is a polynomial algorithm $\mathcal{M}$ that takes as input a 5-tuple $[D, k, k', \Pi, F]$ where D is a totally ϕ-decomposable digraph, k, k' are natural numbers with $k' \leq k$, Π is a list of k' terminal pairs and $F \subseteq A(D)$ is a set of arcs satisfying*

$$d_F^-(v), d_F^+(v) \leq k - k' \text{ for all } v \in V(D) \tag{5.2}$$

$$|F| \leq (k - k')2k,$$

and decides wheter $D \backslash F$ contains a weak Π-linkage.

To prove Equation 5.2.4, we state Algorithm 2 then we prove that it works, and finally that the run time for the algorithm is polynomial. The existence of the algorithm lies in the proof that it works and it is polynomial. We will first explain step by step what happens in the algorithm, then an example of the choices that it makes and when. Then we will prove that is indeed the algorithm mentioned in Equation 5.2.4.

First, we describe with words what the input and output of the algorithm is. The output is already written in words and is very undestandable.
The input can be elaborated somewhat more, first $\mathcal{M}$ is polynomial but also recursively defined. It decides whether (D, Π) has a weak-linkage on k terminals. Since the algorithm is recursive it does not find all the solutions in one go and therefore we define k' as the number of terminals that we still need to find a weak linkage for, and F is a part of the solution of at most $k - k'$ already found weak-linkages of D, Π is the set of terminals that we want to find the weak linkage for.
So when we first cal the algorithm we have $F = \emptyset$ and $k = k'$. This will help with the understanding of the algorithm.

Step 1: "$\Pi = \emptyset$" makes sure that if we call the algorithm $\mathcal{M}$ with no pairs, then there exists the solution with zero arcs to solve the weak-linkage problem.

Step 2: Recall that the digraph is totally ϕ-decomposable and ϕ is bomproof and from Equation 5.2.1, we know that the digraph has a algortihm $\mathcal{A}_\phi$ that gives the ϕ-decomposition of the digraph.

Procedure 2 The main algorithm $\mathcal{M}$

Input: Digraph D, two natural numbers k and k' where $k' \leq k$, a list of k' terminal pairs Π, A set of arcs $F \subseteq A(D)$ satisfying:

$$d_F^-(v), d_F^+(v) \leq k - k' \ \forall v \in V(D)$$
$$|F| \leq (k - k')2k$$

Output: Either "no weak-linkage exists" or "there exists a weak-linkage in (D, Π) with arc set F."

1: **if** $\Pi = \emptyset$ **then**
2: output that a solution exists and return
3: **end if**
4: Run $\mathcal{A}_\phi$ to find a total ϕ-decomposition of $D = S[H_1, \ldots, H_s]$.
5: **if** this decomposition is trivial that is $D = S$ **then**
6: $D \in \phi \subset \phi(1)$, so run $\mathcal{B}_\phi^-$ on $(D \backslash F, \Pi)$ to decide the problem and return.
7: **end if**
8: Find among $H_1, \ldots, H_s$ those houses $K_1, \ldots, K_l$ that contain at least one terminal. Let D' be obtained by contracting all the clean houses. Let F' be the set of arcs obtained from F after the contraction.
9: Let $\Pi^e \subset \Pi$ ($\Pi^i \subset \Pi$) be the list of external (internal) pairs $(s_q, t_q) \in \Pi$.
10: **for** every partion of $\Pi^i = \Pi_1 \cup \Pi_2$, look for external paths linking the pairs in $\Pi^e \cup \Pi_1$ and internal pairs in Π_2 **do**
11: **if** $\Pi^e \cup \Pi_1 = \emptyset$, then for $i = 1, \ldots, l$: **then**
12: run $\mathcal{M}$ recursively on input $[K_i, k, k_i', \Pi \cap K_i, F \cap A(K_i)]$, where $\Pi \cap K_i$ denotes the list of terminal pairs that lie inside K_i and k_i' is the number of those pairs.
13: **end if**
14: **if** $\Pi^e \cup \Pi_1 \neq \emptyset$ **then**
15: let k_i' be the number of pairs in $\Pi_2 \cap K_i$
16: **for** each possible choice of l vertex sets $W_i \subseteq V(K_i), i = 1, \ldots, l$ of size $\min\{|V(K_i)|, 2(k' - k_i')(k - k')\}$ and arc sets $F_i \subseteq A(K_i \langle W_i \rangle) \backslash F$, $i = 1, \ldots, l$ with F_i satisfying

$$d^-_{F_i \cup (F \cap A(K_i))}(v), d^+_{F_i \cup (F \cap A(K_i))}(v) \leq k' - k_i'. \tag{5.3}$$
$$|F_i| \leq 2(k' - k_i')(k - k') \tag{5.4}$$

do
17: **for** every K_i **do**
18: remove all vertices of $V(K_i) \backslash W_i$ and then delete all remaining arcs except those in F_i ($A(D \langle W_i \rangle) \backslash F_i$).
19: **end for**
20: Define D'' to be the digraph obtained from D' with this procedure.
21: Run B_ϕ^- on $(D'' \backslash F', \Pi^e \cup \Pi_1)$.
22: **for** $i = 1, \ldots, l$ **do**
23: run $\mathcal{M}$ recursively on input $[K_i, k, k_i', \Pi_2 \cap K_i, F_i \cup (F \cap A(K_i))]$.
24: **end for**
25: **end for**
26: **end if**
27: **if** in step 11 the if-statement have examined all instances and the are linked **or** in the if statment at step 14 there is a choice of $W_i, F_i, i = 1, \ldots, l$ such that all instances examined are linked **then**
28: output that a weak linkage exists and return.
29: **end if**
30: **end for**
31: output that no weak linkage exists.

Figure 5.1: The internal pair (s,t) in the house H_i linked internal and linked external

.

Step 3-5: From Equation 5.2.1 we know that $\mathcal{B}_\phi$ decides a weak-linkage for $D \in \phi$, since we can not guarentee that $D\backslash F \in \phi$ we use Theorem 5.2.3 that tells us that $\mathcal{B}_\phi^-$ can decide a weak-linkage in $D\backslash F = (V, A\backslash F)$ if $D' \in \phi$, $D' = (V, (A\backslash F) \cup F) = (V, A) = D \in \phi$.

Step 6: Here we find all the non-clean houses from $H_1, .., H_s$ and contract all the clean houses w.r.t. Π. We make a new enumeration of all the non-clean houses $K_1, \ldots, K_l$ of D w.r.t. Π. Since by Theorem 5.2.1 we know that contracting one clean house in D if it has a linkage so does our new digraph, then use this lemma again and again until there are no more clean houses. This is our new digraph D' with non-clean houses $K_1, \ldots, K_l$ and if we find a weak linkage w.r.t. Π in D' we know that D has a weak linkage from continueing using Theorem 5.2.1. We also let $F' = F \cap A'$ where A' is the arcset of D'.

Step 7: Recall an internal pair is where both vertices are in the same house and an external pair is where the vertices are two different houses.

Step 8: This for loop is looking for two different kinds of paths between internal pairs since the path for an internal pair can be an internal path (fully containt in the house) or an external path going out of and later into the house. For simplification look at Figure 5.1

Step 9-11: if $\Pi^e \cup \Pi_1 = \emptyset$, either we have already found all external paths or there is none. Either way, all terminal pairs left are internal hence $\Pi = \Pi_2$. So we are only interested in finding the internal path of the internal pairs, which is why we can call $\mathcal{M}$ on each house for itself. Each K_i could be a big graph in itself that is decomposable with at least some house H_i where $|H_i| \geq 2$, if this is not the case the algorithm returns after step 3, and continue with the next. If $\mathcal{M}$ has already found some external paths, F might not be empty and may use some arcs inside K_i therefore $F \cap A(K_i)$. $\Pi \cap K_i$ is because we are not interested in the terminal pairs that are not a part of the graph we are looking at (pairs inside K_j where $j \neq i$).

Step 12: Looking for external paths in a big graph is a bit more defficult since we do not know which arcs and vertices not to use.

Step 13-14: First we find all the pairs that are internal pairs, that we want to link as internal paths, the number of these is k_i' for each $i = 1, \ldots l$. Then we choose a very specific size of vertex sets W_i and loop over every choiche of these. This vertex set induces a subdigraph, where we make a possible arc set F_i containg no arcs of F. We make this set as big as we need to link the external pairs and some internal pairs (those we want to find external paths of $\Pi^e \cup \Pi_1$) the number of those is $k' - k_i'$ since every pair maybe has to go through the house we are looking at.

Step 15-19: For each house, we remove all vertices not in the vertex set W_i. After removing these vertices, we remove all remaining arcs except those arcs in F_i. This is defined in the algorithm as D''. We can show that $D'' \in \phi(2k^2)$. First we know that since D is totally decomposable $S \in \phi$ and from Equation 5.2.1 and the definition of $D(c)$, we can take S and add as many parrallel arcs as we want(no more than a multiplicity of k is

needed). We only need to blow up l vertices those houses of D that are not clean we know that there are k' terminal pairs and that $k' \leq k$ meaning $l \leq 2k$ these l vertices needs to be blown up and from Theorem 5.2.2 let us say that we want to find $k'' \leq k'$ external paths in D ($|\Pi^e \cup \Pi_1| = k''$). Then we are only looking at k'' terminals, meaning in every blow up we need at most $2k''(k'' + (k - k'))$. Since $k'' \leq k'$ we have $2k''(k'' + (k - k')) \leq 2k''k$ vertices in W_i and $2kk'' \leq 2kk' \leq 2k^2$, which is the biggest number we will need to blow up the l vertices meaning $c = 2k^2$ so $D'' \in \phi(2k^2)$.

Step 20-24: We need to make sure that the tuple $[K_i, k, k'_i, \Pi_2 \cap K_i, F_i \cup (F \cap A(K_i))]$ upholds every condition for every choice of that tuple. Since we are not focusing on loops, we know that the max number of arcs is bounded by the max number of vertices $|F_i| \leq 2kk''$. The rest of the terminals is the number of internal pairs which we in the algorithm denote k'_i. we know that $k'_i \leq k' - k''$ meaning $k'' \leq k' - k'_i$. we start calculating the two demands of F given in the tuple. Note that $d_{(F \cap A(K_i))}(v) = d_F(v)$, $\forall v \in V(K_i)$ and we also know $d_{F_i}(v) \leq k''$ so

$$d^+_{F \cup F_i}, d^-_{F \cup F_i} \leq k - k' + k'' = k - (k' - k'') \leq k - k'_i \tag{5.5}$$

$$|(F \cap A(K_i)) \cup F_i| \leq |F| + |F_i| \leq 2k(k - k') + 2kk'' \tag{5.6}$$

$$\leq 2k(k - k') + 2k(k' - k'_i) = 2k(k - k'_i). \tag{5.7}$$

Clearly the tuple for F holds for all its conditions.

Example 5.2.5. *This example is based on Figure 5.2. The whole figure is considered one digraph D and the set $\Pi = \{(s_1, t_1), \ldots, (s_8, t_8)\}$. D is totally ϕ-decomposable and contains a Π-linkage. Step 4 returns the houses that is the outer red circles in the figure. Since the decomposition is not trival, $D' = D$. We look for clean houses for which there are none. So after step 8 we split Π up to external pairs $\Pi^e = \{(s_1, t_1), (s_2, t_2), (s_5, t_5)\}$ and internal pairs $\Pi^i = \{(s_3, t_3), (s_4, t_4), (s_6, t_6), (s_7, t_7), (s_8, t_8)\}$. In this example, the partition of the internal pairs are going to be focusing on internal paths before external paths, meaning $\Pi_1 = \emptyset$ first, then all set combination of one pair than two pairs and so on. So first we have $\Pi_1 = \emptyset$ and $\Pi_2 = \Pi^i$. Since $\Pi^e \cup \Pi_1 \neq \emptyset$, we enter the if-statement in step 14. Then we make the vertex sets W_i which we do not go deep into in this example.*
Let us say that that we succesfully link the external paths and we now call $\mathcal{M}$ recursively on each house starting with the house in the upper left corner. Since we have linked the pairs that was present in this house, $\Pi = \emptyset$ and we return. The algorithm now calls itself recursively on the house in the upper right corner. Let us say this house $H_2 \in \phi$. Since there are two external terminals in house originally F is properly not empty but it does not matter since we call $\mathcal{B}^-_\phi$ that accounts for this. In this case, $\mathcal{B}^-_\phi$ succesfully link the pair (s_4, t_4). The next house will be the one under, let us say that the decomposition of this also is trivial. $\mathcal{B}^-_\phi$ can not link (s_4, t_4), meaning the algorithm returns and makes a new partition $\Pi_1 = \{(s_3, t_3)\}$ and the rest in Π_2. Since there is no difference when it comes to the house H_3 the algorithm end up returning agian after all the same steps. The algorithm makes a new partition $\Pi_1 = \{(s_4, t_4)\}$ and $\Pi_2 = \{(s_3, t_3), (s_6, t_6), (s_7, t_7), (s_8, t_8)\}$. All the external pairs are linked including the pair (s_4, t_4). We call the 3 first houses and like before exept now H_3 has no terminals that needs to be linked, so we return and continue with the house to the left.
H_4 has unlinked terminals and the ϕ-decomposition is not trivial, we see the houses in Figure 5.2. The green house is a clean house and so is the house contaning t_2. Since it is already linked, in step 8 we contract these two sets. In step 9 we split the pairs into external and internal pairs $\Pi^e = \{(s_3, t_3)\}$ and $\Pi^i = \{(s_6, t_6)\}$, so again since $\Pi^e \cup \Pi_1 \neq \emptyset$. So we enter the if-statement in step 14 link the external pair and then call the algorithm recursively on the houses. Except the houses we have contracted. Either we end up linking the pair as an internal path or we return and link it as an external path. We return all the way to the main digraph and call $\mathcal{M}$ on the last house.

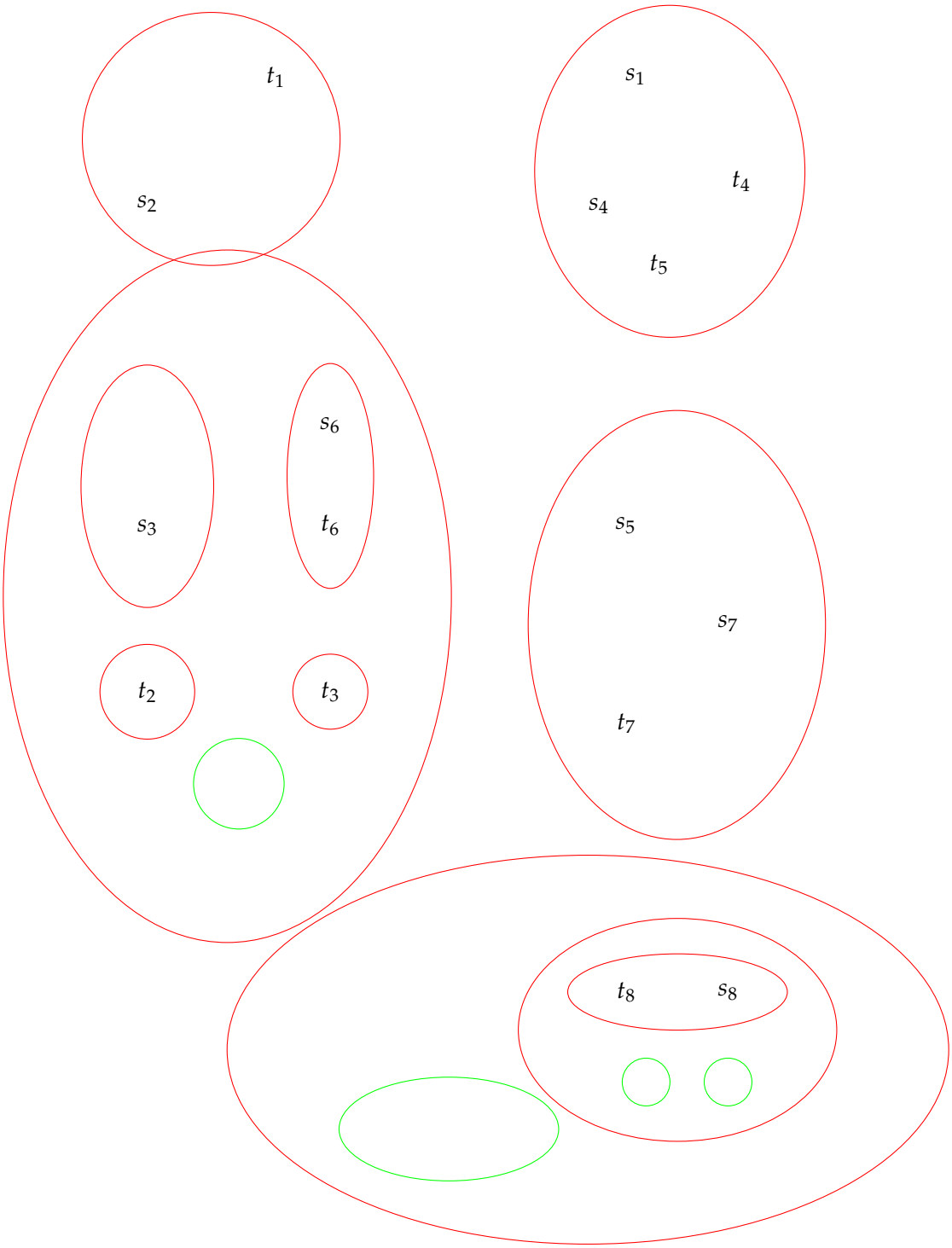


Figure 5.2: Example for the algorithm $\mathcal{M}$. The Figure is a digraph where the red circles is the totally ϕ-decompostion each house H_1, H_2, H_3, H_4, H_5 is either a digraph in ϕ (H_1, H_2, H_3) or totally ϕ-decomposable(H_4, H_5). For H_4 the red and green circles inside is the houses of the totally ϕ-decomposition. Same goes for H_5. $s_1, \ldots, s_8$ is the source vertices of each terminal pair and $t_1, \ldots, t_8$ are the sink vertices of the terminal pairs. $\Pi = \{(s_1, t_1), \ldots, (s_8, t_8)\}$.

This is not a trival decomposition as there is only an internal pair no external pairs, we contract the green clean house and instead of entering the if-statement in step 14, we enter the if-statement in step 11. In this if-statement we do not have to construct a set of arcs for the external paths since there are none. We directly call $\mathcal{M}$ recursively on the house, there is only one house, and this house does not have a trivial decomposition. Then we contract the two green clean houses and enter the if-statement at step 11, since $\Pi_1 = \emptyset$ and $\Pi_2 = \{(s_8, t_8)\}$. It turns out that there only is a (t_8, s_8)-path but no (s_8, t_8)-path so we return and make a new partition $\Pi_1 = \{(s_8, t_8)\}$ and $\Pi_2 = \emptyset$ and enter step 14 instead and we end up linking the piar external.
We enter step 27 and return and enter step 27 and return and now we are at the main digraph (the root level). Enter step 27 and output that a linkage exists.

Proof. We have now proven and explained each step in the algorithm, each step does what it is supposed to. Now we need to check whether given your favorite digraph, that upholds the conditions, the algorithm gives the right result. If the digraph do not terminate before examining list $\Pi^e \cap \Pi_1$ of k'' terminal pairs if $k'' = 0$ we enter step 9 and $F_i = \emptyset$, for $i = 1, \dots l$, and by the induction hypothesis we can assume that if there exists a weak linkage in each K_i the algorithm would find it. Now if this is not the case and $k'' > 0$, step 12 is then entered and we construct D'' which as described belong to $\phi(2k^2)$. then we can use B^-_ϕ which is correct by Equation 5.2.1. So the algorithm will find a weak Π''-linkage if it exists in $D''\backslash F'$. After all this there is made a recursive call on each K_i finding k'_i weak linkages and by the above proof of step 20-24 we know we can and by the induction hypothetise it returns with the linkage.
So since B^-_ϕ correctly finds the weak linkage inside $D''\backslash F'$ using only arcs from F_i $\forall i \in [l]$, then each K_i is recursively called from D'' and do not use any of the arcs in F_i by the induction hypothesis of the algorithm. So we can easily construct to D' from D'' since the weak k'_i-linkage inside each K_i not using any of the arcs from F_i and the external path do only use the arcs of F_i. We know that together these still form the seperated weak linkages. By Theorem 5.2.3 we know that we can find a weak linkage in $D\backslash F$ if we can find it in $D''\backslash F'$ which we just proved we can, returning a perfekt weak Π-linkage of D.

Now we prove that the running time is polynomial. Let $T(n, k, k')$ the upper bound running time of the algorithm. We assume that $T(n, k, k')$ by induction on n and k' is $O(n^{c(k')})$ for fixed function $c(k')$. If $n = 0$ time is constant same if $k' = 0$. Algorithm $\mathcal{A}_\phi$ in step 4 is polynomial from definition and the same is $\mathcal{B}^-_\phi$ in step 6. Contracting the clean houses is also polynomial. Let say that all these steps together takes $O(n^{b(k')})$ for some fixed function $b(k')$. All combinations of Π_1, Π_2 is $O(2^k)$ which we see as constant, since k is fixed. Step 12: are recursive call on the Algorithm 2 hence the time of this step is all the calls $\sum_{i=1}^{l} T(n, k, k'_i)$ with $n_i = |V(K_i)|$. In worst case there are no clean houses so $\sum_{i=1}^{l} n_i = n$. Since we have entered the if-statement in step 11, all terminal pairs are internal and $\sum_{i=1}^{l} k'_i = k'$. By induction hypothesis $\sum_{i=1}^{l} T(n, k, k'_i)$ takes $\sum_{i=1}^{l} O(n_i^{c(k'_i)})$.
The if-statement in step 14 we choose $W_1, \dots W_l$, $F_1, \dots, F_l$ combinations of these is at most $\left(\binom{n}{2k'^2} \cdot \binom{4k'^4}{2k'^2}\right)^{2k'}$, we will prove that this is the same as $O(n^{4k'^3})$. ' Since k' is fixed $\binom{4k'^4}{2k'^2}$ can be treated as a constant. $\binom{n}{2k^2} = \frac{n!}{(2k^2)!(n-2k^2)!}$ and we have $\lim_{n\to\infty} \frac{\frac{n!}{(2k'^2)!(n-2k'^2)!}}{n^{2k'^2}} = \frac{1}{2k'^2}$ which is a constant so we have that $\binom{n}{2k'^2} = O(n^{2k'^2})$. Thus

$$\left(\binom{n}{2k'^2} \cdot \binom{4k'^4}{2k'^2}\right)^{2k'} = \left(O(n^{2k'^2})\right)^{2k'} = O(n^4 k'^3).$$

This is how many times we have to run step 21: which is polynomial by correctness of $\mathcal{B}_\phi^-$ say it takes time $O(n^{d(k')})$ and the recursive calls in step 23: $\sum_{i=1}^{l} T(n_i, k, k_i')$. The whole if-statement takes time

$$O(n^{4k'^3})(\sum_{i=1}^{l} T(n_i, k, k_i') + O(n^{d(k')})). \tag{5.8}$$

Now we use the same argument as in the proof of Theorem 4.2.5 and get

$$O(n^{4k'^3})(2k' \cdot T(n, k-1, k'-1) + O(n^{d(k')})). \tag{5.9}$$

We let $c(k') := 10^{k'} + d(k') + b(k')$ and we will see that both if-statements take at most $O(c^k)$. The first if-statement in step 11.

$$\sum_{i=1}^{l} O(n_i^{c(k_i')}) = O(\sum_{i=1}^{l} n_i^{c(k_i')}) = O((\sum_{i=1}^{l} n_i)^{c(k')}) = O(n^{c(k')}). \tag{5.10}$$

We now make the same calculations for the if-staement in step 14 which takes time at most

$$O(n^{4k'^3})(2k' \cdot T(n, k-1, k'-1) + O(n^{d(k')})) = O(n^{4k'^3})(O(n^{c(k'-1)}) + O(n^{d(k')})) = \tag{5.11}$$

$$O(n^{4k'^3 + 10^{k'-1} + d(k'-1) + b(k'-1)}) + O(n^{4k'^3 + d(k')}). \tag{5.12}$$

We just need these inequalities to hold

$$4k'^3 + 10^{k'-1} + d(k'-1) + b(k'-1) \leq 10^{k'} + d(k') + b(k') \tag{5.13}$$

clearly $d(k-1) + b(k-1) \leq d(k') + b(k)$ left we have $4k^3 + 10^{k-1} \leq 10^{k'}$ which is obviously also true. We also clearly have $d(k) + 4k^3 \leq c(k')$. Thus the whole loop takes at most $O(n^{c(k)})$. (recall that $k' \leq k$ so in worst case which is what we are focusing on $k = k'$). And left we have that

$$T(n, k, k') \leq O(n^{c(k)}) + O(n^b(k)) = O(n^c(k)). \tag{5.14}$$

□

5.2.1 k-linkage problem for quasi-transitive digraphs

We have already established in section 2.2 that quasi-transitive digraphs are totally ϕ_1-decomposable. It turns out that we just have to prove that ϕ_1 is bombproof. For that we need the two polynomial algorithms $\mathcal{A}_{\phi_1}$ and $\mathcal{B}_{\phi_1}$. Recall that ϕ_1 is built up by semicomplete and acyclic digraphs so we need to establish some theorems for the weak k-linkage problem on semicomplete and acyclic digraphs.

Theorem 5.2.6. *The weak k-linkage problem is polynomially solvable for every fixed k when the input is an acyclic digraph.*

Theorem 5.2.7. *The weak k-linkage problem is polynomial for every fixed k, when we consider digraphs that are obtained from a semicomplete digraph by replacing some arcs with multiple copies of those arcs and adding any number of loops.*

Since the bombproof class allows the digraph to no longer be a part of that class, we need to consider that an acyclic digraph can get a cycle when blowing up a vertex.

Theorem 5.2.8. *For every natural number p, the weak k-linkage problem is polynomial for every fixed k, when we consider digraphs with most p directed cycles.*

Now we can prove that ϕ_1 is bombproof and therefore that quasi-transitive digraphs have a polynomial solution for the weak k-linkage problem, when k is fixed.

Theorem 5.2.9. *The class ϕ_1 is bombproof.*

Proof. For ϕ_1 to be bombproof it has to adhere the properties of Equation 5.2.1, the totally ϕ_1-decomposition can be found in polynomial time for any ϕ_1-decomposable digraph by Theorem 2.1.2. Now we only need the algorithm $\mathcal{B}_{\phi_1}$ for this we need to look at the construction of $D(c)$ where $D \in \phi_1$. Let $D' \in D(c)$. Either D is semicomplete or D is acylic.
If D is semicomplete, D' has at most c blown-up vertices $H_1, \dots H_c$ of D to size at most c. If these H_i are independent, we do not need more then c^2 arcs, for each H_i to be semicomplete. So there are at most c^3 arcs missing from D' for it to be semicomplete, then by Theorem 5.2.3, we can find a weak k-linkage for D' if D is semicomplete. Now suppose D is an acyclic blowing up c vertices with at a size at most c, we have inside these of the acyclic digraph no more than $O(c^c)$ cycles pressent in the house. Since D is acyclic there are no cycles between the houses. There are at most k parallel arcs since no more are needed. Which brings up the number of cycles in the house to $O((ck)^c)$ so there are at most $O(c \cdot (ck)^c)$ cycles in D'. Then we can use Theorem 5.2.8 where we let $p = c \cdot (ck)^c$. So for all possibilities of D and all cases of $D' \in D(c)$ we have a polynomial algorithm that solves the k-linkage problem, meaning the $\mathcal{B}_{\phi_1}$ algorithm exists and ϕ_1 is a bombproof class. □

5.3 Weak Linkage in Locally Semicomplete Digraphs

Locally semicomplete digraphs can be round-decomposable. It turns out that we can from the independence number $\alpha(D)$ tell whether a digraph is round-decomposable or not. Recall independence number from section 1.1. The theorem below is from [4] where we omit some part of it since we have it stated elsewhere in the thesis.

Theorem 5.3.1. *[4] A locally semicomplete digraph D havinng idependece number $\alpha(D)$ at least 3 is round decomposable with a unique round-decomposition.*

This means when considering all other locally semicomplete digraphs, it has an independence number $\alpha(D) \leq 2$, which means for all non round-decomposable locally semicomplete digraphs, we can use the algorithm in Theorem 5.3.2 to solve the weak k-linkage problem when k is fixed.

Theorem 5.3.2. *For every natural number α the weak k-linkage problem is polynomial for every fixed k, when we consider digraphs with independence number at most α.*

For solving the weak k-linkage problem in locally semicomplete digraphs, we now only need to find a polynomial algorithm for the round-decomposable digraphs. Before going into this, we have to introduce something called the cutwidth. This definition of cutwidth is inspired by the describtion of the cutwidth in [4].
Given a digraph D and an ordering of the vertices $O = v_1, \dots, v_n$ we say that the ordering O has a **cutwidth** at most θ if $\forall j \in \{2, 3, \dots n\}$. There are at most θ arcs u, v with $u \in \{v_1, \dots, v_{j-1}\}$ and $v \in \{v_j, \dots, v_n\}$ see .... Say we have another ordering O' of the same

digraph D, if O' has a cutwidth at most θ for all possible orderings O' of D, then D is said to have a **cutwidth** at most θ.
The minimum natural number θ such that D has a cutwidth at most θ, we call θ the **cutwidth** of D. When we know the cutwidth of the digraph we can solve the weak k-linkage problem for those in polynomial time.

Theorem 5.3.3. *[4] For every natural number θ, the weak k-linkage problem is polynomial for every fixed k, when we consider digraphs with cutwidth at most θ.*

For the rest of this section **interval** will be used with respect to the round ordering of a round digraph. An **interval** is a subset of vertices $v_i v_{i+1} \dots v_{j-1} v_j$ where the vertices are consecutive compared to the round ordering. In this case, the intervals left and right endpoints is v_i and v_j respectively. From a round digraph D, we are going to construct another digraph called the **compression of** D **with respect to** Π and is denoted D_Π.
We will now introduce disjoint intervals $I_1, \dots, I_l$, where all terminals are contained in their union ($\Pi \subseteq \bigcup_i^l I_i$) and the left endpoint of each I_i is a terminal. Also, the next $6k$ vertices on the left of the interval I_i are not terminals. The next $6k$ vertices on the right of the interval are not terminals either. This is true for all $i \in [l]$. let I_1 be the interval which left endpoint is a terminal with the lowest possible number in the round ordering that adheres to the properties of the intervals endpoints. This condition enforces uniqueness.
This can be done unless the digraph is smaller than $12k^2$. How we find these intervals will be described later in this section. First we want to introduce L_i and R_i, which are both intervals of the round ordering and L_i is the $3k$ vertices left of I_i and R_i is the $3k$ vertices right of I_i. Now we have the rest of the vertices that are not in any interval and we define W_i to be the interval of the vertices between R_i and L_{i+1}. See figure in [4] page 103.
Now the compression of D with respect to Π is the digraph obtained from D by contracting W_i and if necessary we delete arcs such that only k multiple arcs is left (the maximum multiplicity of an arc is k).

We want to show that finding a weak Π-linkage in a round digraph D you can just as well find a weak Π-linkage in its compression with respect to Π. we are only focusing on round digraphs with cutwidth at least $\Theta = k(6k + 36k^2(2k+1)^2)$. Since the cutwidth is so large we can clearly see that the size of D is not smaller than $12k^2$ so we can construct D_Π.

Lemma 5.3.4. *Let D be a round digraph with round ordering O and cutwidth at least Θ. Let Π be a list of terminal pairs. D has a weak Π-linkage if and only if its compression with respect to Π, D_Π, has a weak Π-linkage.*

The construction of D_Π is based on the intervals I_i. For the first interval I_1, we find the first terminal τ where any of the $6k$ vertices left of τ are not terminals. We now make τ the left endpoint of I_1 and look at the $6k$ vertices to the right. If they contain another terminal τ', we let every vertex up till τ' including τ' be in I_1 and look right on the next $6k$ vertices from τ'; if it contains a terminal include it in I_1 as we did with τ'. If no we have I_1 and we know that the next terminal $\tau*$ right of I_1 has at least $6k$ vertices to the left that are not terminals. We now make $\tau*$ the left endpoint of I_2 then we do with I_2 as we did with I_1. We keep doing this until all terminals are a part of an interval I_i. Then we can easily construct L_i and R_i from I_i and from this we can find W_i if it exists (is not empty) do this forall $i \in [l]$. then we constract W_i and we now have constructed D_Π.
In this thesis, we are focusing on the decomposable digraphs and we can define a compression for the round decomposable digraphs too. As the compression for round digraphs the compression of round decomposable digraphs, is both defined in [4] page 102 - 105. So we

assume $D = R[H_1, \ldots, H_r]$ is round decomposable. Then we contract the clean houses so we now have $D' = R'[H'_1, \ldots, H'_r]$, which is the digraph after the contraction. The only difference between R' and R is the multiplicity of the arcs, we will construct Π' from Π where for each pair, $(s_i, t_i) \in \Pi$ where $s_i \in H_z$ and $t_i \in H_q$, we make a pair $(v_z, v_q) \in \Pi'$ where $v_z, v_q \in V(R')$. We now make a compression of R' with respect to Π', $R'_{\Pi'}$. Let $v_{j_1}, \ldots, v_{j_p}$ be the vertices of the compression $R_{\Pi'}$.
Now we define the compression of D with respect to Π to be the digraph $D_\Pi = R'_{\Pi'}[H'_{j_1}, \ldots, H'_{j_p}]$. The intervals I_j are the only ones with terminals in and therefore the only intervals of $R'_{\Pi'}$ that have some blown-up vertices in D_Π. We know from Theorem 5.2.1 that we can contract the clean houses and in the proof of Theorem 5.3.4, which can be found in [**?**] page 103-104, that the paths are split up in (s_i, σ_i)-path, (σ_i, τ_i)-path, (τ_i, t_i)-path, that obviouse joined together is an (s_i, t_i)-path. The (σ_i, τ_i)-path is not inside any I_b interval and follows therefore by lemma 5.5 in [4]. Both the (s_i, σ_i)-path and the (τ_i, t_i)-path do not use the property of I_b being round and can therefore be linked in the same way for $D_\Pi = R'_{\Pi'}[H'_{j_1}, \ldots, H'_{j_p}]$. Which brings us to this lemma.

Lemma 5.3.5. *Let D be a digraph of the form $D = R[H_1, \ldots H_r]$, where R is round and has cutwidth at least Θ. Let Π be a list of piars of terminals. D has a Π-linkage if and only if D_Π, has a Π-linkage.*

Now we can use all this to prove that the class ϕ_2, which is defined in section 2.1, is bombproof and recall that round-decomposable digraphs is totally ϕ_2-decomposable.

Lemma 5.3.6. *The class ϕ_2 is bombproof.*

Proof. For ϕ_2 to be bombproof we need to find $\mathcal{A}_{\phi_2}$, which we have from Theorem 2.1.2. we have already proven the existence of $\mathcal{B}_{\phi_2}$ if $R \in \phi_2$ is semicomplete - for this, see the proof of Theorem 5.2.9. Therefore assume that R is round. We want to show that the weak k-linkage problem is polynomial on $R(c)$ for a positive integer c. Let a digraph $D \in R(c)$. Now recall $\Theta = k(6k + 36k^2(2k+1))^2$ then when R is round we will base the proof on two cases one where R has a cutwidth at least Θ and another where R has cutwidth at most Θ.
In both cases we have $D = R[H_1, \ldots, H_r]$ where at most c of the H_i houses has $|V(H_i)| > 1$ and R has an ordering O, $v_1, \ldots, v_r$ where H_i in D corresponds to v_i in R.

Case 1 When considering a digraph $D = R[H_1, \ldots, H_r]$ with size $|V(R)| \geq 12k^2$, we can create a compression of R with respect to some $\Pi*$ created from Π, and therefore we can construct the compression of D with respect to Π, D_Π. As we know from the way we constructed D_Π, the size is only dependent on c and k, since $R_{\Pi*}$ has a size depending on $|\Pi * | \leq k$, and we blow up at most k vertices to a size at most c. Since c and k are both fixed natural numbers, we use a brute-force algorithm (an algorithm that checks all possibilities) to solve the weak Π-linkage problem on D_Π and from Theorem 5.3.5 D has a weak Π-linkage if and only if D_Π has a weak Π-linkage.
Since c and k are fixed, the brute-force algorithm is polynomial and the construction of D_Π is also polynomial.

Case 2 R has a cutwidth at most Θ for the round ordring O so for $D = R[H_1, \ldots, H_r]$ we construct an ordering O', where for every $u \in H_i$ and $z \in H_j$ with $i \neq j$, we have that $u < z$ in O' if $v_i < v_j$ in O. The ordering of the vertices inside a house H_l is not important for the proof. Now cutwidth θ' of O' is at most $k(c^3 + c^2 \cdot \Theta)$. To calculate this we know that there are at most c houses H_i where $|V(H_i)| > 1$. These houses have size at most c. There are at most c^2 arcs inside a house with possible multiplicity k

since we are not interested in more. we know now that the arcs inside the houses that can contribute to the cutwidth is at most $c^2 \cdot k \cdot c$. The other arcs that can contribute to the cutwidth θ' are the arcs between the houses H_i and H_j where $v_i < v_j$ in O. The number of arcs between two such given houses is $c \cdot c \cdot k$ since both have a size on maximum c and the multiplicity of these arcs is at most k. We can not have more than Θ cases of such two houses since they represent vertices of R with cutwidth at most Θ. So $\theta' \leq c^3 \cdot k + c^2 \cdot \Theta \cdot k = k(c^3 + c^2 \cdot \Theta)$. Therefore, we can use the algorithm from Theorem 5.3.3 to solve the k-linkage problem of $D = R[H_1, \ldots, H_r]$ ($R(c)$).

Thus we have found $\mathcal{B}_{\phi_2}$. □

As mentioned above and proven in section 2.3 ,round-decomposable digraphs are totally ϕ_2-decomposable and we have just proved that ϕ_2 is bombproof, so by the Algorithm 2 for bombproof classes, every round-decomposable digraph now have a polynomial algorithm to solve the weak k-linkage problem.

Theorem 5.3.7. *For every fixed k there exists a polynomial algorithm for the weak k-linkage problem for round-decomposable digraphs.*

This ends the part for round-decomposable digraph and in the begining of this section we proved that all other locally semicomplete digraphs than the round-decomposable ones have a polynomial algorithm for the weak k-linkage problem. We can now state this.

Theorem 5.3.8. *For every fixed k there exists a polynomial algorithm for the weak k-linkage problem for locally semicomplete digraphs.*

Bibliography

[1] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[2] Jørgen Bang-Jensen and Jing Huang. Quasi-transitive digraphs. *Journal of Graph Theory*, 20(2):141–161, 1995.

[3] Jørgen Bang-Jensen, Tilde My Christiansen, and Alessandro Maddaloni. Disjoint paths in decomposable digraphs. *Journal of Graph Theory*, 85(2):545–567, 2017.

[4] Jørgen Bang-Jensen and Alessandro Maddaloni. Arc-disjoint paths in decomposable digraphs. *Journal of Graph Theory*, 77(2):89–110, 2014.

[5] Jørgen Bang-Jensen, Yubao Guo, Gregory Gutin, and Lutz Volkmann. A classification of locally semicomplete digraphs. *Discrete Mathematics*, 167:101–114, 1997.

[6] Jørgen Bang-Jensen and Jing Huang. Decomposing locally semicomplete digraphs into strong spanning subdigraphs. *Journal of Combinatorial Theory, Series B*, 102(3):701–714, 2012.

[7] Tilde My Christiansen. *Algorithmic and structural problems in digraphs*. PhD thesis, 2018.

[8] Jørgen Bang-Jensen. Digraphs with the path-merging property. *Journal of Graph Theory*, 20(2):255–265, 1995.

[9] Jørgen Bangjensen, Jing Huang, and Erich Prisner. In-tournament digraphs. *Journal of Combinatorial Theory, Series B*, 59(2):267–287, 1993.

[10] Jørgen Bang-Jensen and Pavol Hell. Fast algorithms for finding hamiltonian paths and cycles in in-tournament digraphs. *Discrete applied mathematics*, 41(1):75–79, 1993.

[11] G Gutin. Characterization of vertex pancyclic partly oriented k-partite tournaments, vestsi acad. navuk bssr ser. fiz. *Mat. Navuk N*, 2:41–46, 1989.

[12] Jørgen Bang-Jensen and G Gutin. *Vertex Heaviest Paths and Cycles in Quasi-Transitive Digraphs*. Odense Universitet. Institut for Matematik og Datalogi, 1994.

[13] Maria Chudnovsky, Alex Scott, and Paul Seymour. Disjoint paths in tournaments. *Advances in Mathematics*, 270:582–597, 2015.

[14] Maria Chudnovsky, Alex Scott, and Paul Seymour. Disjoint paths in unions of tournaments. *Journal of Combinatorial Theory, Series B*, 135:238–255, 2019.