Karlsruhe Institute of Technology

Embedding into Graph Products: Computational Complexity and Algorithms

Bachelor's Thesis of

Antonia Heiming

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer:	T.T.-Prof. Dr. Thomas Bläsius
Second reviewer:	Dr. rer. nat. Torsten Ueckerdt
Advisor:	Marcus Wilhelm

December 2024 – March 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, March 2025

Antonia Heiming

. . . .

(Antonia Heiming)

Abstract

We consider the computational complexity of embedding graphs and especially trees into various sorts of grids. To do this, we denote that a graph G is embeddable into a graph H if and only if there is a map that maps vertices of G to vertices of H and edges of G to edges of H. We understand a grid as a product of two paths. We investigate multiple embedding problems of graphs and trees. We alternate this grid by using the cartesian product and the strong product for the grid. We observe that deciding whether embedding a tree in the product of two paths of infinite and a clique is NP-complete for an arbitrary clique size. However, we find that it is decidable in linear time whether a caterpillar is embeddable into such a grid. Moreover, we also find that embedding a graph in the strong product of a path of infinite length and a path of given length k is possible in $O(n^{O(k)})$. Finally, we give a linear time algorithm for embedding a tree in the strong product of a path of infinite length and a path of length 1.

Zusammenfassung

In dieser Arbeit betrachten wir die Komplexität, zu entscheiden ob ein Graph und vor allem ein Baum in verschiedene Arten von Gittern einbettbar ist. Wir definieren, dass ein Graph G in einen Graphen H einbettbar ist, falls wir eine Abbildung finden, die die Knoten aus G auf Knoten aus H abbildet und die Kanten von H auf Kanten von G.

Defür betrachten wir mehrere explizite Einbettungsprobleme, wobei wir verschiedene Variationen von Gittern untersuchen, in die Graphen oder Bäume eingebettet werden. Wir interpretieren das Gitter als einen Produktgraphen aus zwei Pfaden und betrachten sowohl das Kartesische als auch das Strong Produkt. Für beide Produktarten multiplizieren wir das jeweilige Gitter mit einem vollständigen Graphen beliebiger Größe. Wir stellen fest, dass die Entscheidung ob ein Baum in dieses Gitter einbettbar ist, NP-vollständing ist. Trotzdem ist es in Linearzeit entscheidbar, ob ein Caterpillar-Baum in ein solches Gitter einbettbar ist. Schließlich schränken wir das Gitter ein. Wir sehen, dass zu entscheiden ob es möglich ist einen beliebigen Graphen in das Strong Produkt aus einem unendlich langem Pfad und einem Pfad der Länge k einzubetten, in $O(n^{O(k)})$ möglich ist. Außerdem stellen wir einen Algorithmus vor, welcher in Linearzeit entscheidet, ob ein Graph in das Strong Produkt eines unendlich langen Pfades und einem Pfad der Länge 1 einbettbar ist.

Contents

1 Introduction

We say that a graph G is embedded into a graph H, if there is a subgraph of H that is isomorphic to G. Another way to look at the definition of embedding is to have a mapping that maps edges of G to edges of H and vertices of G to vertices of H. Another possibility is to have a map that maps edges of G to paths in H. For this definition we can find many different variations of the problem whether a graph is embeddable into another graph, for example the VLSI-Layout problem [KV84].

Embedding graphs is not only interesting for VLSI-Layouts but also for finding subgraphs, which is possible with our definition of embedding. Embeddings can also be used to gain a better understanding of the graph to embed, because it may inherit properties of the graph it is embedded in, for example fulfilling Brouwser's conjecture [TT24].

In this thesis we look at embedding graphs and trees into various forms of grids. So we look at related problems to the VLSI-embedding but with a different definition of embedding. The grids are product graphs defined as being composed of two paths, where the vertex set is the cartesian product of the vertex sets of the paths. The edge set of a product graph depend on the product we use. We consider the strong product and the cartesian product. The cartesian product of two paths is a grid, while the strong product also contains diagonal edges in the grid. Finding an embedding into product graphs is interesting, because they provide some regularity because of their construction.

It is already investigated that embedding a tree in both products is NP-complete ([BEU23], [Gre89]). Since a tree is a special case of a graph, the general problem of embedding a graph is also NP-complete. To get a better understanding where the complexity comes from, when embedding a graph into a grid, we look at the computational complexity of different related problems, where we both alternate the grid and the graph to embed. We alternate the grid and see that if we restrict the grid such that it has a constant height k, deciding whether an arbitrary graph G is embeddable is possible in $O(n^{O(k)})$. When considering $k = 1$ we give a linear time algorithm that provides an embedding of a graph G if G is embeddable. We also enlarge the grid by multiplying it with a clique. We observe that even deciding wether a tree is embeddable in this enlarged grid is NP-complete. We further restrict the graph to embed in. Here we see that embedding a caterpillar in such an enlarged grid is possible in linear time.

In the following we first give an overview to related problems. Then, we introduce notation and general concepts relevant for this thesis. Chapter 3 provides results on embedding trees in the product of two paths of infinite length and a clique. We first show for each product that the embedding problem is NP-complete for a clique size of 2 and generalize the result by using an arbitrary clique size but 2. In Chapter 4 we show that embedding a caterpillar in a product of two paths of infinite length and a clique is possible in polynomial time. Before we draw a conclusion, we show that embedding a graph in the grid of limited hight is possible in polynomial time and give an algorithm to find an embedding of a graph G in linear time if G is embeddable, when restricting the height to 1.

1.1 Related Work

In this thesis we investigate the embedding of trees (and graphs) in different (multilayered) grids. We understand grids as product graphs. We use the definition of embedding where vertices are mapped to vertices and edges to edges. In literature one finds a lot of related problems with focus on variations of the problem of embedding graphs into graphs. First, one can change the definition of embedding, by mapping edges to paths. In this case of embedding the problem of finding an VLSI-Layout is widely investigated [KV84]. The VLSI-Layout problem is about finding an embedding of a graph in a grid, that uses as little space as possible. Kramer and Van Leeuwen show that the VLSI-Layout problem is NP-complete. Starting from the VLSI-Layout problem many further investigations for different variations of the problem are made [KV84]. For VLSI-Layouts it is of great interest to have an embedding with the fewest possible bending paths [AKS91]. Formann and Wagner show that the VLSI-Layout problem is still NP-complete if the intersecting property of the paths is altered [FW91]. Some restrictions to the problem were made by only embedding planar graphs if a planar embedding is given [Tam87] or to only embed trees [BLSS04] even in multilayered grids [Iva16]. For each of these restrictions they observe that the embedding problem is solvable in polynomial time. But the main question in these kinds of embedding problems remains the space complexity.

Another definition of embedding is the embedding we use. Here vertices are mapped to vertices again, but edges are mapped to edges and not to paths. In this case space complexity is less important, since the graph G to embed already gives the space of the embedding if G is embeddable. We have that the general embedding problem is NP-complete [CGL17]. When looking on restrictions of the embedding problem, we can look at product graphs to embed in. Embedding graphs into product graphs is interesting since product graphs may inherit properties of the graphs they consists of or further constraints can be put on them [TT24]. There is a lot of research on embedding a graph G into the strong product of a path, a clique and a graph with bounded treewidth ([UWY22], [Dvo+21], [HW25]). There, G or the product graph is further restricted and they show that the embedding problem is still NP-complete. When restricting the graph to embed into to a multilayered grid, embedding a graph is still NP-complete [Tik16]. We investigate a similar problem but try to embed a tree. Gregori observes that embedding a tree in a grid is NP-complete [Gre89]. Biedl, Eppstein, and Ueckerdt show that embedding tree in the strong product of two paths is NP-complete [BEU23].

We generalize two results stated by Biedl, Eppstein, and Ueckerdt in Chapter 3 and Chapter 4. In Chapter 3 the reduction follows the same idea as in [BEU23] for constructing a new tree and reducing the problem of embedding a tree into a product of two paths and a clique to the problem of embedding a tree in the cartesian product of two paths of infinite length. Chapter 4 consist of two statements and proofs which are adapted from [BEU23]. They show a similar statement but for embedding the caterpillar into the strong product of infinite length, which is a special case for Section 4.1 when choosing clique size 1.

Besides restricting either the graph to embed into or the graph to embed one can investigate properties of both graphs. Matoušek and Thomas investigates such properties at looking at different definitions of embedding [MT92]. Sudakov and Vondrák give some properties for graphs such that a tree could be embedded into them, for example that the graph to embed in does not contain cycles with to small diameter [SV10].

2 Preliminaries

Let a *graph* G be a tuple (V, E) consisting of a set of *vertices* V and a set of *edges* $E \subseteq \binom{V}{2}$ connecting these vertices. If we refer to the vertices or edges of G we write $V(G)$ or $E(G)$ to specify the graph. We define a *subgraph* $S \subseteq G$ of a graph G as a graph, with $V(S) \subseteq V(G)$ and $E(S) \subseteq E(G)$ and $E(S) \subseteq \binom{V(S)}{2}$. Moreover for simplification we also denote by S a subgraph of G, if there is a subgraph G' of G that is isomorphic to S. So we want that the labeling of the vertices is not relevant for defining a subgraph. We define a restriction to an edge set of a graph as follows: Let $G = (V, E)$ be a graph and $X \subseteq E$. We define $G|_X = (V(G|_X), X)$ with $V(G|_X) = \{x \mid xy \in X, x, y \in V(G)\}$, as the subgraph of G that only contains vertices of G that are part of an edge in X as the vertex set of $G|_X$ and X as the edge set.

In the following we define some properties of graphs. Let G be a graph and $v, w \in V(G)$. We define the *distance between v and w in G* $\operatorname{dist}_G(v, w)$ is the length of a shortest path in G between the given vertices v and w. We use the distance to define the *k-hop-neighborhood of v in G*, $N_k^G(v)$ as the set of all vertices in G which have distance k to v, i.e. $N_k^G(v) = \{w \in V(G) \mid \operatorname{dist}(v, w) = k\}$. Sometimes we talk about the *at-most-k-hop-neighborhood of v in G*, which includes all vertices in G with distance at most k to v. i.e. $N_{\leq k}^G(v) = \{w \in V(G) \mid \operatorname{dist}(v, w) \leq k\}$.

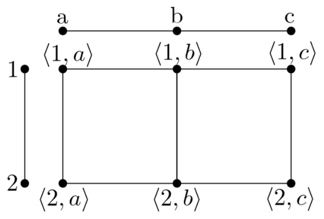
We omit the graph G in all of the previous definitions if G is clear from the context.

In the following we state some common graph concepts. A *tree* T is a connected and acyclic graph. A path P is a tree where every vertex has at most degree 2. We write uv-path, if there is a path P, where u and v are the only vertices with degree 1. We define u and v as the *end points* of P. To join two paths P and Q we write PQ. This only make sense when P and Q share exactly one end point x, where P and Q get connected. We identify a the vertices of a path with integers and we use P_k as the path of length k. Normally we refer to the vertices of P_k in an enumerated way starting at 0. Note that $k = \infty$ is possible. In this case we assume that the path is infinite in both directions. A *clique of size ω*, K_ω is a fully connected graph of size ω.

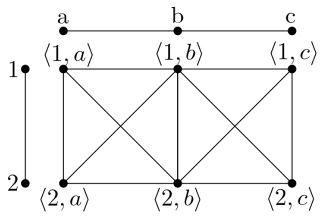
We define a *product graph* G as the product of at least two graphs $G_1, \ldots, G_n$ in the following way, which follows closely the definitions of Hickingbotham and Wood [HW21]. Since only the edge set of a product graph is determined by the type of product we first introduce the vertex set for all used products. The vertex set of G is the cartesian product of the vertex sets of the graphs $G_1, \ldots, G_n$. We call the vertices of the product graph *composed vertices*. We have that every vertex of a product graph consists of one vertex of each graph. We write $\langle u_1, \ldots, u_n \rangle \in V(G_1) \times \cdots \times V(G_n)$, where each $u_i \in V(G_i)$. We use this notation if it is important to know which vertices the composed vertex consists of. In this thesis we focus on the cartesian product and on the strong product.

Let $G = G_1 \square \cdots \square G_n$ the *cartesian product* of the graphs $G_1, \ldots, G_n$. As defined above let $V(G) = V(G_1) \times \cdots \times V(G_n)$. We then have the edge set

$$E(G) = \{\langle v_1^1, \ldots, v_n^1 \rangle \langle v_1^2, \ldots, v_n^2 \rangle \mid \exists! i \in [1, n] : v_i^1 v_i^2 \in E(G_i) \wedge \forall j \neq i : v_j^1 = v_j^2\}$$



(a) cartesian product



(b) strong product

Figure 2.1: Both figures show an example of the respective product of two paths of length 2 and 3 and with named vertices.

. Two vertices u, v in G are connected if there is exactly one graph G_i, where the vertex components of u and v have an edge in G_i and all other vertex components are the same for u and v. Figure 2.1a gives an example of a product graph for a cartesian product of two paths. For the *strong product* let $G = G_1 \boxtimes \cdots \boxtimes G_n$ be a product graph with $V(G) = V(G_1) \times \cdots \times V(G_n)$. The edge set contains all edges of the cartesian product as well as the set

$$\{\langle v_1^1, \dots, v_n^1\rangle\langle v_1^2, \dots, v_n^2\rangle \mid \forall i \in [n] : v_i^1 v_i^2 \in E(G_i)\}.$$

Figure 2.1b shows an example for the strong product of two paths.

If nothing else is stated, all used graphs are simple, undirected and finite. The only case when we have infinite graphs is graphs to embed into. Since we only embed finite graphs, it would be sufficient to look at “large enough” finite graphs but we do not want to define what “large enough” means in every context.

We define an *embedding* φ of a graph G into another graph A as an injective function $\varphi : V(G) \to V(A)$ such that if there is an edge $uv \in E(G)$ there is also an edge $\varphi(u)\varphi(v) \in E(A)$. For a graph S we define $\varphi(S)$ as the embedding of all vertices of S.

3 Embedding a Tree in Different Products of Two Paths and a Clique

In this chapter we show that deciding whether a tree is embeddable in a product of two infinite paths and a clique is still NP-complete. Using similar arguments as Biedl, Eppstein, and Ueckerdt we generalize their result [BEU23]. Biedl, Eppstein, and Ueckerdt state that embedding a tree in either cartesian or the strong product of two paths of infinite length is NP-complete.

First, we introduce the graphs we try to embed trees in. The *ω-rectangle grid R_ω* is the cartesian product of two paths of infinite length and a clique of size ω, i.e., $R_\omega = P_\infty \Box P_\infty \Box K_\omega$. We call the strong product of two paths of infinite length and a clique of size ω a *ω-full grid* $F_\omega = P_\infty \boxtimes P_\infty \boxtimes K_\omega$ for $\omega \in \mathbb{N}$. Note that for $\omega = 1$ each vertex of the respective grid has $V(K_1)$ as vertex component and this is the case that Biedl, Eppstein, and Ueckerdt showed [BEU23].

Let $G_\omega \in \{F_\omega, R_\omega\}$. We also introduce notation for some parts of G_ω. A *layer* is a subgraph of a ω-rectangle/full grid where all composed vertices contain the same vertex of K_ω as vertex component.

For the following definition of the partitions of the edge set of G_ω we use $G \in \{P_\infty \Box P_\infty, P_\infty \boxtimes P_\infty\}$. By using G, we define edge sets which are needed for both the cartesian and the strong product.

All edges in the same layer are *horizontal*. The set of all horizontal edges is defined as

$$H = \{\langle a, y\rangle\langle b, y\rangle \mid a, b \in V(G) \wedge y \in V(K_\omega) \wedge ab \in E(G)\}. \tag{3.1}$$

All composed vertices that consists of the same vertices of the infinite paths, but different vertices of K_ω are *vertical copies* of each other. Edges connecting vertical copies are *vertical*. We define the set of all vertical edges

$$S = \{\langle a, x\rangle\langle a, y\rangle \mid a \in V(G) \wedge x, y \in V(K_\omega) \wedge xy \in E(K_\omega)\}. \tag{3.2}$$

Now we have that $R_\omega = P_\infty \Box P_\infty \Box K_\omega$ with the vertex set $V(R_\omega) = V(P_\infty) \times V(P_\infty) \times V(R_\omega)$ and the edge set $E(R_\omega) = H \cup S$. In Figure 3.1 S and H are color coded for R_2 and also different layers are color coded.

In the ω-full grid there are still other edges which are neither horizontal nor vertical. The remaining edges in F_ω are *slanted* and we define the set of all slanted edges

$$A = \{\langle v, u\rangle\langle x, y\rangle \mid vx \in E(F_1) \wedge uy \in E(K_\omega)\}. \tag{3.3}$$

Then we have $F_\omega = P_\infty \boxtimes P_\infty \boxtimes K_\omega$ with the vertex set $V(F_\omega) = V(P_\infty) \times V(P_\infty) \times V(K_\omega)$ and the edge set $E(F_\omega) = H \cup S \cup A$. Figure 3.2 shows a section of a 2-full grid where H, S and A are color coded as well as different layers.

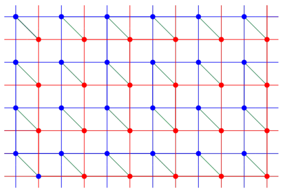


Figure 3.1: A section of a 2-rectangle grid. We have that all edges of S are colored green and we have two layers, where one is colored blue and the other one red. Note that both the red and the blue edges are all edges of H

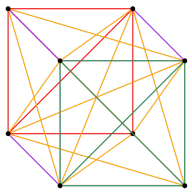


Figure 3.2: A section of a 2-full grid. Here all edges are color coded. Vertical edges are purple and thus of S. Red and green edges belong to H but we see a red and a green layer. All orange edges are slanting edges and thus belong to the set A. Note that some edges are not visible in the figure because of the perspective of the drawing.

Note that F_1 is isomorphic to $P_\infty \boxtimes P_\infty$ and R_1 to $P_\infty \square P_\infty$ respectively, since every vertex of G_1 contains the vertex of K_1 as vertex component. By looking at the edge set we see that K_1 does not provide any edges at all. Thus for convenience we do not make a difference between using the product without or with K_1. i.e. we use F_1 and $P_\infty \boxtimes P_\infty$ and R_1 and $P_\infty \square P_\infty$ respectively interchangeably.

Note that H, S and A are all pairwise disjoint, since none of R_1, F_1 and K_ω have self-loops and thus an edge cannot be in two sets of H, S and A.

In this chapter we prove for the presented grids that embedding trees is NP-complete. To do this, we reduce this from the problem $\mathscr{P}$ of embedding a tree in a grid to embedding a tree in the cartesian product of two paths of infinite length, because $\mathscr{P}$ is NP-complete. First, we construct a new tree T' from the given tree T we want to embed. We call T' the *gadget tree* for T. We construct T' by replacing every vertex u of T by some *gadget* $G(u)$ and connect these gadgets $G(u)$ through *connecting vertices*. For constructing this gadget tree, let T be a tree with degree at most four. Let $G(w)$ be a rooted tree with root w and four pairwise different connecting vertices $a_1, \dots, a_4$, which are all different from w. We construct the gadget tree T' by replacing every vertex v of T by $G(v)$. Replace every edge $uv \in E(T)$ by a *connecting edge* between $G(u)$ and $G(v)$ by connecting a_i and a_j for some $i, j \in \{1, \dots, 4\}$ and $a_i \in V(G(v))$ and $a_j \in V(G(u))$. We first show that the gadget tree is indeed a tree, so that we only have to specify the gadgets and the connecting vertices for every following reduction in this chapter.

Note that every tree having at least one vertex of degree at least five, does not have an embedding in R_1 since every vertex in R_1 has degree 4.

Lemma 3.1: *Let T be a tree where every vertex has at most degree four. Let $G(v)$ be a rooted tree with root v and four pairwise different connecting vertices $a_1, \dots, a_4$, which are all different from v and T' the gadget tree as defined above.*

Then T' is a tree.

Proof. To show that T' is a tree, we show that T' contains $|V(T')| - 1 = n - 1$ edges and is connected. Let T' be constructed as described above.

First we show that T' is connected. Assume for the sake of contradiction that T' contains at least two connected components. Then there are two gadgets $G(u)$ and $G(v)$, with $u, v \in V(T)$ which are in different connected components of T'. Thus there is no path connecting u and v in T'. Since edges are only present if there is a corresponding edge in T there cannot be a uv-path in T. This is a contradiction to T being a tree.

Now we show that T' has $n - 1$ edges. Since every gadget is a tree, we have that for all $v \in V(T)$ the gadget $G(v)$ contains $|V(G(v))| - 1$ edges. Since every connecting edge e connects two different connected components C_1, C_2, we have that the connected component consisting of C_1, C_2 and e has $|V(C_1)| + |V(C_2)|$ vertices and $(|V(C_1)| - 1) + (|V(C_2)| - 1) + 1 = |V(C_1)| + |V(C_2)| - 1$ edges and thus is a tree. We have that each of the connecting edges connects different components otherwise T would not be a tree.

Having that T' is connected and contains $|V(T')| - 1$ edges, T' is a tree.

■

For showing that a problem is NP-complete we show that the problem is in NP and that we can reduce the problem from another NP-complete problem. For showing the reduction we use the problem of deciding whether a tree is embeddable in R_1. Every section in this chapter uses the following idea for its respective proof of the reduction, which closely follow the proof by Biedl, Eppstein, and Ueckerdt [BEU23]. For the reduction, we build a gadget tree T' from a tree T and show that the problem of deciding whether there is an embedding T' in the respective product graph G is equivalent to deciding whether there is an embedding T in the 1-rectangle grid. For the proof of the equivalence we want to sketch the idea of "if there is an embedding of a gadget tree, we find an embedding of a tree". To show the implication it is sufficient to show that the construction of a gadget and thus the gadget tree forces a shortest path P in the respective grid connecting two gadget roots u, v to always have the same length. In this case we have that these paths P are some sort of straight paths and of the same length. We identify the respective gadget with a vertex in R_1 and each of those paths P with an edge. The definition of a straight path and the embedding of the gadgets is specified in the respective proof.

For showing that the problem of embedding a tree in the respective grid is in NP, we give a more general statement.

Lemma 3.2: *Let T be tree. Let G be a graph. Then deciding whether T is embeddable in G is in NP.*

Proof. We claim for a map $\varphi : V(T) \rightarrow V(G)$ to be an embedding of a tree T in G.

To justify whether φ embeds T in G we check that the definition set of φ is the whole vertex set of T. Next we check that φ is injective by counting the vertices in the image of φ. We have that φ is injective if and only if the image is of size $|V(T)|$. Then we consider every edge $uv \in E(T)$ and check if there is an edge $\varphi(u)\varphi(v)$.

All these checks are possible in polynomial time. Checking the size of the image set and the definition set is possible in linear time. Checking if there is always a corresponding edge is also possible in polynomial time.

Thus we have that deciding whether T is embeddable in G is in NP. ■

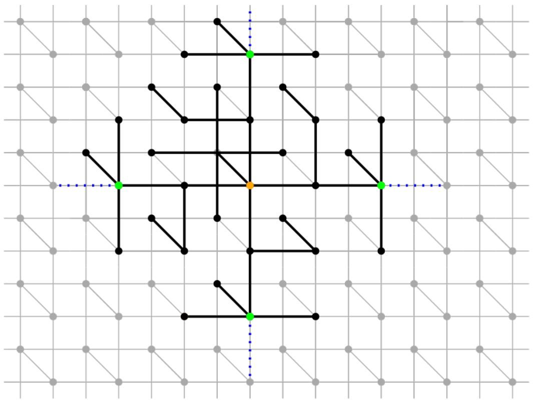


Figure 3.3: An embedding of the gadget $G(v)$ in R_2. The vertex v is colored in orange. The connecting vertices are colored in green. The blue dotted lines suggest where other gadgets can be connected.

3.1 Embedding a Tree in the 2-Rectangle Grid

We show that deciding whether there is an embedding of a tree into the cartesian product of two paths of infinite length and a clique of size 2, R_2 is NP-complete. To do this, we reduce this problem from the problem of deciding whether a tree is embeddable into the 1-rectangle grid $R_1 = P_\infty \square P_\infty$. We only need to show that the problem of deciding whether a gadget tree T' is embeddable in the 2-rectangle grid is an equivalent problem to decide whether a tree T is embeddable into the R_1 and that constructing the gadget tree is possible in polynomial time. The gadget tree T' is constructed of T and a given gadget.

We distinguish two sorts of paths within the ω-rectangle grid. A path P is called *straight* if every vertex on P contains the same vertex component of one of the two P_∞. If the vertex component changes we say that P *bends.*

We use a gadget tree T' constructed of the original tree by using Lemma 3.1. Let T be a tree without loss of generation with maximal degree 4. Then we construct a gadget $G(v)$ by using $v \in V(T)$ as a root of $G(v)$. This gadget is shown in Figure 3.3. Let v have five child vertices $w_1, \dots, w_5$. Let w_1 have four own children. Let $w_2, \dots, w_5$ have two children w_j^1, w_j^1 for $j \in \{2, \dots, 5\}$, where w_j^1 has one child itself and w_j^2 has three own children. We have that $G(v)$ is a tree by construction and $G(v)$ has four vertices of degree 4, i.e. w_j^2 for $j \in \{2, \dots, 5\}$. Use w_j^2 for $j \in \{2, \dots, 5\}$ as connecting vertices. Now we can apply Lemma 3.1 to construct T'.

We see that constructing T' from T is possible in polynomial time, since every vertex of T only gets replaced by a constant amount of vertices. In particular the enlargement of each vertex of T does not depend on the size of T itself. With this, we show the needed equivalence.

Theorem 3.3: *Let T, T' be trees, where T' is the gadget tree as defined above.*
Then it holds that $T \subseteq R_1$ if and only if $T' \subseteq R_2$.

Proof. First, let $T \subseteq R_1$ and $V(K_2) = \{a, b\}$. Now we show that we find an embedding of T' in the 2-rectangle grid R_2.

To do this, we construct an embedding $\varphi : V(T') \to V(R_2)$. The embedding φ maps every vertex of T to a vertex with vertex component a. To embed T' each gadget is placed as shown in Figure 3.3. Place the gadgets in such a way that for every edge $uv \in E(T)$, the respective path $\varphi(u)\varphi(v)$ is a straight path of length 5 in R_2. Since the gadgets only cover the at-most-2-neighborhood of each vertex in R_2 all vertices of each gadget can be placed. The connecting edges are placed between two gadgets by construction of the embedding. Thus φ is an embedding of T' in R_2.

Now let $T' \subseteq R_2$ with an embedding φ. For showing that there is an embedding of T in R_1 it is sufficient to show, that the construction of the gadget enforces that the shortest path to connect two gadgets is a straight path with length 5 between their roots. Since we only have four connecting vertices at each gadget $G(v)$ we can map each of these shortest paths to an edge in R_1 and each root v the respective vertex v. In this case we have at most four outgoing edges of the same length for each vertex. Since each root vertex of T' is also a vertex of T we have an embedding of T in R_1. To show that the construction of T' enforces straight paths between the roots of the gadgets in T', we show that the whole at-most-2-hop neighborhood of a vertex $v \in V(T)$ is covered by $\varphi(G(v))$.

To do this, we see that $|N^{T'}_{\leq 2}(v)| = 18$. This is exactly the number of vertices that have distance at most two to $\varphi(v)$ in R_2. Thus we have that φ uses $N^{R_2}_{\leq 2}(\varphi(v))$ to embed $N^{T'}_{\leq 2}(v)$. For each $uv \in E(T)$ there is a path P of length 5 in T'. Let P be $uw_1w_2w_3w_4v$. Here, $w_1, w_2 \in V(G(u))$ and $w_3, w_4 \in V(G(v))$. By construction of the gadget, w_2 has three neighbors in $G(u)$ besides w_1 as well as w_3 has three neighbors in $G(u)$ besides w_4. The $\varphi(w_2)\varphi(v)$-path P is straight and of length 2, because w_2 has four neighbors in T' as well as $\varphi(w_2)$ in R_2. One of the neighbors of w_2 or $\varphi(w_2)$ is w_1 or $\varphi(w_1)$ respectively. We have that $\varphi(w_1)$ and w_1 have distance 1 to u and $\varphi(u)$ respectively. The remaining three neighbors have distance 3 to u or $\varphi(u)$ respectively. From this it follows that $\varphi(w_1)$ is connected to all its neighbors in R_2. If P contains a vertex which is obtained from $b \in V(K_2)$, so the edge changes the layer. Then $\varphi(w_2)$ has a neighbor $\varphi(w')$ with $dist(\varphi(w'), \varphi(u)) = 2$. Since both vertex components of u changed once each. We have that $w'u \notin E(T')$, because otherwise there would be a cycle. If P bends but stays in the same layer, $\varphi(w_2)$ has neighbor $\varphi(w'')$ with distance 1 to $\varphi(u)$. Again, $w_1w'' \notin E(T')$ because it would close a cycle. So $\varphi(G(u))$ for an $u \in V(T)$ is unambiguous. The edge $\varphi(w_2)\varphi(w_3)$ extends P in a straight way, because otherwise there are not six vertices, which are adjacent to one of w_2 and w_3 and have distance at least 3 to both u and v. Thus, we have that the gadget $G(w)$ covers exactly the at-most-2-hop neighborhood of their respective vertex w in T and the only edges connecting this vertices are the four connecting edges, such that every two root of adjacent gadgets have distance 5 in R_2 and are connected through a path of length 5. ∎

Having this equivalence it remains to show that embedding a tree is indeed in NP.

Theorem 3.4: *Deciding whether a tree T is embeddable in R_2 is NP-complete.*

Proof. To show that the given problem $\mathcal{R}$ is NP-compete we have to show that deciding whether a tree T is embeddable in R_2 is in NP and that we find a polynomial reduction from another NP-hard problem. By Lemma 3.2 we have that $\mathcal{R}$ is in NP. Now we reduce $\mathcal{R}$ to the problem $\mathcal{P}$ of deciding whether T is embeddable into R_1. Remember that $\mathcal{P}$ is NP-complete. Let T be a tree. We construct a gadget tree T' as explained above. As already seen, this is possible in polynomial time. By Theorem 3.3 we have that $\mathcal{P}$ is equivalent to the deciding whether the gadget tree T' is embeddable in R_2.

Thus we have that $\mathcal{R}$ is NP-complete. ∎

3.2 Embedding a Tree in the ω-Rectangle Grid

We generalize Section 3.1, by showing that deciding whether a tree is embeddable in R_ω but R_2 is still NP-complete. To do this, we construct a gadget tree T' with help of Lemma 3.1 with a gadget $G_\omega(v)$, which may depend on the used clique size ω. We show that the problem $\mathcal{P}$ of deciding whether a tree is embeddable into the 1-rectangle grid is equivalent to the problem $\mathcal{R}$ of deciding whether a gadget tree is embeddable into the ω-rectangle grid. We make sure that the construction of the gadget tree only needs polynomial time. We also use the same definition for straight and bending paths as in Section 3.1. Before looking at $\mathcal{R}$, we show some neighborhood properties of the ω-rectangle grid.

Lemma 3.5: *Let $\omega \in \mathbb{N}$, $R_\omega = R_1 \square K_\omega$, then for all $v \in V(R_\omega)$ holds:*

(1) $|N_1^{R_\omega}(v)| = 4 + (\omega - 1)$

(2) $|N_2^{R_\omega}(v)| = 8 + 4(\omega - 1)$

Proof. Before looking at both statements we look at the partition of the edge set $E(R_\omega) = S \cup H$ given by Equation (3.1) and Equation (3.2). Remember that the sets of vertical edges S and horizontal edges H are disjoint. Thus, to determine the neighborhood of an arbitrary vertex v in R_ω, it is sufficient to determine the neighborhood of v in each of $R_\omega|_S$ and $R_\omega|_H$. Let $v = \langle a, b, c\rangle \in V(R_\omega)$, with $\langle a, b\rangle \in V(R_1)$ and $c \in V(K_\omega)$.

Now we show (1). Since $R_\omega|_S$ connects a vertex with its copies, each connected component in $R_\omega|_S$ is isomorphic to K_ω. Therefore $|N_1^{R_\omega|_S}(v)| = |N_1^{K_\omega}(c)| = \omega - 1$. We also have that $R_\omega|_H$ only connects vertices within a layer, thus it holds that $|N_1^{R_\omega|_H}(v)| = |N_1^{R_1}(\langle a, b\rangle)| = 4$. Hence, $|N_1^{R_\omega}| = |N_1^{R_\omega|_S}(v)| + |N_1^{R_\omega|_H}(v)| = (\omega - 1) + 4$.

For showing (2) we see that each path starting from v to a vertex of $N_2^{R_\omega}(v)$ contains at least one horizontal edge. Otherwise, assume for sake of contradiction that there is a vertex $w \in N_2^{R_\omega}(v)$ and both edges e, f on the vw-path are vertical. Thus e and f lay in the same connected component of $R_\omega|_S$. Since every connected component of $R_\omega|_S$ is isomorphic to K_ω, there have to be the edge $vw \in E(R_\omega)$, which contradicts the assumption of $w \in N_2^{R_\omega}(v)$.

So, to determine $N_2^{R_\omega}(v)$ there are two sorts of possible paths. First consider the paths $\mathscr{P}$ consisting of two horizontal edges. In this case the paths $\mathscr{P}$ stay in the same layer as v and we have $|N_2^{R_1}(v)| = 8$. Now consider a path P consisting of one horizontal and one vertical edge. In R_ω it does not matter whether the vertical edge is the first to take. Without loss of generality, the vertical edge is the first to pick and the second one is the horizontal. Since v has $\omega - 1$ vertical copies P can reach $4(\omega - 1)$ different vertices. Theses both cases are disjunct, because they all take place in different layers and only count neighbors within one layer. Hence, $|N_2^{K_\omega}(v)| = 8 + 4(\omega - 1)$. ■

Now we construct the gadget $G_\omega(v)$ for $\omega \in \mathbb{N}$, as shown in Figure 3.4.

Let v have $4 + (\omega - 1)$ children $u_1, \ldots, u_{4+(\omega-1)}$. Let $u_5, u_6, \ldots, u_{4+(\omega-1)}$ have four children each. Let $u_1, \ldots, u_4$ have two children u_j^1 and u_j^2 for $j \in [4]$ each with u_j^1 having $\omega - 1$ own children and u_j^2 having $\omega + 1$ own children. We have that $G_\omega(v)$ is a tree by construction. We have four vertices with degree $\omega + 2$, i.e. $u_1^2, \ldots, u_4^2$. We use $u_1^2, \ldots, u_4^2$ as connecting vertices for applying Lemma 3.1 and constructing a gadget tree T'. Note that the gadget tree is constructed in polynomial time, since every vertex of T is enlarged by a constant number of vertices, since ω is given by $R\omega$. Next, we show the needed equivalence.

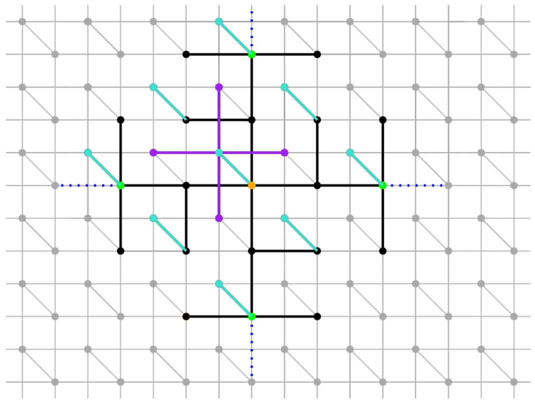


Figure 3.4: The embedding of the gadget $G(v)$ in K_ω. This is quite the same gadget as seen in Figure 3.3. The turquoise edges connect all different layers and v' is the vertical copy of v in each layer. For better readability only two layers are displayed. The purple edges and vertices show the structure which is on every layer with an v'. Again the orange vertex is v and the green vertices are connecting vertices. The blue dotted lines suggest where to connect other gadgets.

Theorem 3.6: *Let T be a tree and T' a gadget tree as defined above. Let $\omega \in \mathbb{N}$. Then it holds that $T \subseteq R_1$ if and only if $T' \subseteq R_\omega$.*

Proof. First, let $T \subseteq R_1$. Now we construct an embedding $\varphi : V(T') \longrightarrow V(R_\omega)$. Let $v \in V(T)$. Let $G(v)$ be a gadget as constructed above. Embed all gadgets such that their roots are all in the same layer. For each vertex $w \in V(G(v))$ with four children $x_1, \ldots, x_4$ let $\varphi(w)$ be a vertical copy of $\varphi(v)$ and $x_1, \ldots, x_4$ in their respective layer. Embed the remaining four children $u_1, \ldots, u_4$ of v in the same layer as $\varphi(v)$. Let u_i^1 the child of u_i with $\omega - 1$ own children for $i \in [4]$. Embed u_i^1 such that the $\varphi(u_i^1)\varphi(v)$ path bends. Embed all $\omega - 1$ as vertical copies of u_i^1. Let u_i^2 the other child of u_i. Embed u_i^2 such that the $\varphi(u_i^2)\varphi(v)$ path is straight. Embed $\omega - 1$ of u_i^2 children as vertical copies of $\varphi(u_i^2)$ and the last two in a bending way for all $i \in [4]$. To connect two gadgets $G(v), G(w)$, there is an path of length 5 by and using the connecting vertices. Therefore all gadgets and edges are embedded. Figure 3.4 shows the φ of a gadget. We have that the whole at-most-2-hop neighborhood of a vertex of T in R_ω is covered and at most four edges may connect different gadgets. The path connecting two root vertices of the gadgets are straight. So we have that there is an embedding of T' in R_ω.

To show the other direction, let $T' \subseteq R_\omega$ with an embedding φ. To obtain an embedding of T in R_1 it is sufficient to show that the construction of T' forces that, if two gadgets are connected their roots are connected through shortest straight paths of length 5. If this is the case there is an embedding of T in R_1 because every root of a gadget is connected to at most four other roots through straight paths P of length 5. Thus we can identify P with an edge in R_1. To show that the construction of T' enforces straight paths, we show that each gadget covers the complete at-most-2-hop-neighborhood of a vertex of T embedded in R_ω.

To do this, choose an arbitrary $v \in V(T)$. With Lemma 3.5 (1), we have that $|N_1^{R_\omega}(\varphi(v))| = 4 + (\omega - 1) = |N_1^{G_\omega(v)}(v)|$, by construction of $G_\omega(v)$. The whole 1-hop-neighborhood of $\varphi(v)$ is covered by $\varphi(G_\omega(v))$. We have that $|N_2^{G_\omega(v)}(v)| = 4(\omega - 1) + 4 \cdot 2 = 4(\omega - 1) + 8 = |N_2^{R_\omega}(\varphi(v))|$, with Lemma 3.5 (2). Again we have that the $N_{\leq 2}^{R_\omega}(\varphi(v))$ is covered by $\varphi(G(v))$.

There is a vertex $w \in N_2^{G_\omega}(v)$ such that $\varphi(w)$ has only one adjacent vertex u in R_ω, such $\operatorname{dist}(\varphi(u), \varphi(v)) \le \operatorname{dist}(\varphi(w), \varphi(v))$. Assume for the sake of contradiction that $\varphi(w)$ is not on the same layer as $\varphi(v)$. Then $\varphi(w)$ is neighboring a vertical copy of $\varphi(v)$ and the vertical copy of $\varphi(w)$ on the layer of $\varphi(v)$. Both of these vertices have distance one to $\varphi(v)$. This contradicts the assumption. From this follows that the connecting vertices are on the same layer as $\varphi(v)$. We have that $\varphi(w)\varphi(v)$-path is straight, since we only change the vertex component of R_1. Note that changing the vertex component of K_ω does not extend the distance, since all vertices with the same vertex component of R_1 but different vertex component of K_ω have distance 1 to v. We have four straight paths starting at $\varphi(v)$ and having distinct endpoints w_i for $i \in \{1, \dots, 4\}$. Since the connecting vertices have $\omega + 1$ children they are w_i, since w_i is adjacent to $\omega + 1$ vertices in R_ω with distance 3 to $\varphi(v)$. When connecting two gadgets $G_\omega(u)$ and $G_\omega(v)$, the path $\varphi(u)\varphi(v)$ has length 5 and it is straight. Let $\varphi(u)a_1a_2a_3a_4\varphi(v)$ be this path. As already shown $\varphi(u)a_1a_2$ and $a_3a_4\varphi(v)$ have to be straight paths. The vertices a_2 and a_3 have $2(\omega + 1)$ neighbors, which have distance at least 3 to both $\varphi(u)$ and $\varphi(v)$. Thus each of a_2 and a_3 can be adjacent to most one vertex except a_2 or a_3 which has distance 2 to either $\varphi(u)$ or $\varphi(v)$. This enforces the edge a_2a_3 to be in a straight path. ■

With this equivalence we show the NP-completeness of the problem of deciding whether a tree is embeddable in the ω-rectangle grid.

Theorem 3.7: *Deciding whether a tree T is embeddable in R_ω is NP-complete.*

Proof. To show that the problem $\mathcal{R}$ of deciding whether a tree T is embeddable in R_ω is NP-complete, we have to show that $\mathcal{R}$ is NP and reduce it from another NP-complete problem. By Lemma 3.2 we have that $\mathcal{R}$ is in NP. For the reduction we use the problem $\mathcal{P}$ of deciding whether a tree T is embeddable in R_1. We construct a gadget tree of T' as described above, which is possible in polynomial time, as mention above. With Theorem 3.6 we have that $\mathcal{P}$ is equivalent to deciding whether T' is embeddable in R_ω. Thus $\mathcal{R}$ is Np-complete. ■

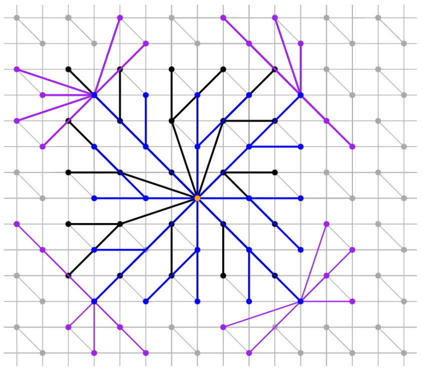
3.3 Embedding a Tree in the 2-Full Grid

We show that deciding whether a tree is embeddable into the 2-full grid is NP-complete. Biedl et al. have already shown that the problem of deciding whether a tree is embeddable into the 1-full grid is NP-complete [BEU23].

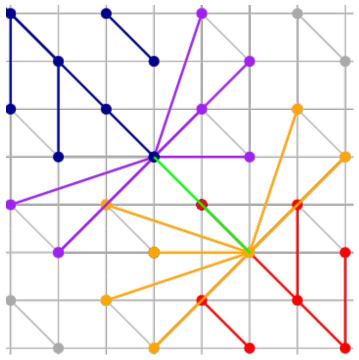
For the reduction part, we show that deciding whether a gadget tree T' embeddable into F_2 is equivalent to the problem of deciding whether a tree embeddable in R_1, which we know that this is NP-complete by [BEU23]. We also remark that constructing the gadget tree is possible in polynomial time.

First, we construct the gadget $G(v)$ shown as in Figure 3.5 We choose v as a root with 17 child vertices $w_1, \dots, w_{17}$. Let $w_5, \dots, w_{16}$ have two child vertices each. Let $w_1, \dots, w_4$ also have two children w_i^1, w_i^2 for $i \in [4]$ and let w_i^1 have six children itself. Note that w_{17} is a leaf. We have at four vertices of degree 7, i.e. $w_1^1, \dots, w_4^1$, which we use as the connecting vertices in Lemma 3.1 and refer to them as j-vertices in the following. By Lemma 3.1 we get a gadget tree T'. This gadget is an alternation of the gadget for F_1 presented in [BEU23]. Since every vertex of T is enlarged by a constant amount of vertices, the construction of the gadget tree is possible in polynomial time. Now we show the needed equivalence.

Theorem 3.8: *Let T be a tree and T' a gadget tree constructed as above.*
$T \subseteq R_1$ if and only if $T' \subset P_\infty \boxtimes P_\infty \boxtimes K_2$



(a) The embedding of $G(v)$ in F_2. v is marked orange. For readability all edges ending in the same layer as v are painted blue. All others are painted black. The purple edges are these edges that provide unique possibility for connecting two gadgets.



(b) An embedding where of two edges of the adjacent gadgets. The green edge connects two connecting vertices. The blue and purple edges belong to one gadget, the orange and red edges to another.

Figure 3.5: For a better understanding of the picture all edges that distinguish R_2 and F_2 are omitted. The edges restricting the connecting edges are colored in another color.

Proof. First, let $T \subseteq R_1$. Now we show that there is an embedding φ of T' in F_2. We construct T' as described above and embed it as shown in Figure 3.5. Since each gadget has four j-vertices and the maximum degree of T is at most 4, there is an embedding $\varphi(T')$.

To show the other direction, let $T' \subseteq F_2$ with an embedding φ. It is sufficient to show that the construction of the gadgets forces an embedding such that for adjacent gadgets $G(v)$ and $G(u)$ with $u, v \in V(T)$ that $\varphi(u)$ and $\varphi(v)$ have distance 5 and the $N^{F_2}_{\leq 2}(v) \cap N^{F_2}_{\leq 2}(u) = \emptyset$. We also find that there are only four possibilities to build connecting paths. If this is the case we can identify each gadget with its root in T and each path connecting two roots with an edge in T. To do this, let $v \in V(T)$ arbitrarily chosen. We have $|N^{F_2}_1(\varphi(v))| = 17 = |N^{T'}_1(v)|$ thus $\varphi(N^{G(v)}_1(v))$ covers $N^{F_2}_1(\varphi(v))$. The vertical copy of $\varphi(v)$ has only neighbors in F_2 which are either $\varphi(v)$ or in $N^{T'}_1(\varphi(v))$. There are 16 vertices $u_1, \ldots, u_{16}$ in $N^{T'}_1(v)$, that have a neighbor with distance 2 to v. Each of $u_1, \ldots, u_{16}$ has exactly two children by construction, so there are $|N^{T'}_1(v)| \leq 2 \cdot 16 = 32$. Since T' is a tree, every vertex (except the root) has exactly one parent vertex, thus $|N^{T'}_2(v)| = 32$. We also have that $|N^{F_2}_2(\varphi(v))| = 32$. From this follows that $N^{T'}_2(v)$ covers of $N^{F_\omega}_2(\varphi(v))$ completely. Note that embedding all vertices of $N^{T'}_2(v)$ in F_2 is possible as shown in Figure 3.5a, limited to vertices of distance at most 2 to $\varphi(v)$.

The edge $uv \in E(T)$ corresponds to a path P of length 5 in T'. Let P be $uw_1w_2w_3w_4v$, where $w_1, w_2 \in V(G(u))$ and $w_3, w_4 \in V(G(v))$, because $G(u), G(v)$ contain each the 2-hop-neighborhood of their respective root u, v. To connect $G(u)$ and $G(v)$ two connecting vertices are connected and these connecting vertices are w_2 and w_3. We claim that for all $x \in (N^{G(u)}_1 \setminus \{w_1\})$ we have $\operatorname{dist}(u, x) = 3$. Since the path uw_1w_2 is in $G(u)$ and the parent vertex of each vertex in a tree is unique, each child of w_2 has distance 3 to u. Each connecting vertex, has six children outside of $N^{G(u)}_{\leq 2}(u)$. This holds, because $N^{F_2}_{\leq 2}(u)$ is covered by $\varphi(N^{G(u)}_{\leq 2}(u))$. We identify F_2 by a $\mathbb{N} \times \mathbb{N} \times 2$ grid. So we have that a neighborhood of a vertex is always a cuboid. We see that each gadget looks like a cuboid with some additional edges coming out of it.

There are three possibilities where connecting vertices can be placed Let $G(v)$ be a gadget. First: A connecting vertex w is embedded on the side of a gadget but not in a corner. Then w has 11 neighbors are in the at-most-2-hop-neighborhood of $\varphi(v)$ in the gadget. Thus $\varphi(w)$ is only adjacent to six vertices outside of $N^{F_2}_{\leq 2}(v)$. In this case it is impossible to embed all child vertices of w. Second: Assume for the sake of contradiction that two connecting vertices w, y lay in one corner of$N^{F_2}_{\leq 2}(v)$ and $\varphi(w)$ and $\varphi(y)$ are vertical copies of each other. The 1-hop-neighborhood for $\varphi(w)$ and $\varphi(y)$ that is outside of $N^{F_2}_{\leq 2}(\varphi(v))$ contains ten vertices. It is not possible to embed all child vertices of w and y since they have in total 12 child vertices. The last possibility is that only one connecting vertex w that is embedded in the corner of $N^{F_2}_{\leq 2}(\varphi(v))$. Then v has ten neighbors outside $N^{F_2}_{\leq 2}(\varphi(v))$, so there is enough space to place all six child vertices and a connecting edge is also embeddable. From this follows that all connecting vertices are placed in a corner of $N^{F_2}_{\leq 2}(\varphi(v))$. When having a connecting edge w_2w_3, there are 12 vertices in F_2 that are adjacent to either $\varphi(w_2)$ or $\varphi(w_3)$ and have at least distance 3 to $\varphi(u)$ and $\varphi(v)$. This is the case if $\varphi(w_2)\varphi(w_3)$ is diagonal. Note that this edge can change the layer. So we have that every two adjacent gadgets $G(v)$ and $G(w)$, v and w are connected through a path of length 5. We also have that the whole $N^{F_2}_{\leq 2}(v)$ is covered by $\varphi(G(v))$. ■

With this equivalence we can show the NP-completeness.

Theorem 3.9: *Deciding whether a tree T is embeddable into F_2 is NP-complete.*

Proof. To show the NP-completeness, we first note that the problem $\mathcal{R}$ of deciding whether a tree T is embeddable into F_2 is in NP by Lemma 3.2. Next, reduce $\mathcal{R}$ to the problem $\mathcal{P}$ of deciding whether a tree is embeddable into R_1. To do this, we construct a gadget tree T' of T. As mentioned above, we have that the construction of the gadget tree is possible in polynomial time. With Theorem 3.8 we have the needed reduction. Thus $\mathcal{R}$ is NP-complete. ■

3.4 Embedding a Tree in the ω-Full-Grid

We generalize the result of Section 3.3 by showing that deciding whether a tree is embeddable into R_ω but R_1 is still NP-complete. We show that the problem of deciding whether a gadget tree T' is embeddable into the ω-full grid is equivalent to the problem of deciding if a tree is embeddable into K_1. We show that constructing a gadget tree T' is possible in polynomial time. To do so, we introduce a gadget $G_\omega(v)$ and construct a gadget tree T' with Lemma 3.1. First, we introduce some properties of the neighborhood in the ω-full grid, to justify the proof of the following theorem.

We have a closer look at the vertices and edges of F_ω and use the definition presented in Equation (3.1), Equation (3.2) and Equation (3.3). Remember that the sets of horizontal edges H, vertical edges S and slanted edges A are pairwise disjointed. We have that $|E(F_\omega)| = |S| + |H| + |A|$.

Corollary 3.10: *If v and w adjacent in F_ω then v is also adjacent to every vertical copy of w.*

Proof. Let $v = \langle a, b, c\rangle, w = \langle x, y, z\rangle \in V(F_\omega)$ with $\langle a, b\rangle, \langle x, y\rangle \in V(F_1)$ and $c, z \in V(R_\omega)$. Assume that they are on the same layer. i.e. $c = z$. Then we have $vw \in H$. Let $w' = \langle x, y, z'\rangle$, $z' \in V(R_\omega)$ be a vertical copy of w. Thus we have that $ww' \in S$. By definition of S and H we have $\langle a, b\rangle\langle x, y\rangle \in E(F_1)$ and $zz' = cz' \in E(R_\omega)$. By Equation (3.3) we have that $vw' \in A$.

Assume that v, w are not on the same layer. Then there is a vertical copy w' of w which is on the same layer as v, i.e. $w' = \langle x, y, c\rangle$. By definition ww' is vertical and thus in S. The edge vw' is in H. Thus, we have that $\langle a, b\rangle\langle x, y\rangle \in E(F_1)$ and $cz \in E(R_\omega)$, so $vw \in A$. Since v, w were arbitrarily chosen, it holds. ■

With this corollary we look at most of properties just in one layer and adjust the result with ω in the correct way. We have that the neighborhood of a vertex in the ω-full grid is a rectangle cuboid. We already showed that it holds for $\omega = 2$ and with Corollary 3.10 we can extend the cuboid.

We show some properties concerning the size of neighborhoods in F_ω.

Lemma 3.11: *Let $\omega \in \mathbb{N}$, $v, u \in V(F_\omega)$ and $F_\omega = F_1 \boxtimes K_\omega$, then:*

(1) $|N_1^{F_\omega}(v)| = 8 + 9(\omega - 1)$

(2) $|N_2^{F_\omega}(v)| = 16\omega$

(3) The amount of vertices which are in the neighborhood of a corner of $N_2^{F_\omega}$, but not in $N_{\leq 2}^{F_\omega}(v)$ is 5ω

(4) The amount of vertices which are in the neighborhood of a side of $N_2^{F_\omega}$, but not in $N_{\leq 2}^{F_\omega}(v)$ is 3ω.

(5) Amount of vertices needed to enforce that the only possibility to connect $N_{\leq 2}^{F_\omega}(u)$ and $N_{\leq 2}^{F_\omega}(v)$ by a diagonal edge between corners is 6ω.

Proof. Let $v = \langle a, b, c\rangle \in V(F_\omega)$. For proving the first statement we look at the different neighborhoods determined by S, H and A. The size of the neighborhood of v can be determined as follows:

$$|N_1^{F_\omega}(v)| = |N_1^{F_\omega|_S}(v)| + |N_1^{F_\omega|_A}(v)| + |N_1^{F_\omega|_H}(v)|$$

. First we look at S. We have $|N_1^{F_\omega|_S}(v)| = |N_1^{K_\omega}(c)| = \omega - 1$. The first equality holds, because only the vertex component of K_ω changes in a connected component C in $F_\omega\ |_S$. We have that C is isomorphic to K_ω. For H we have $|N_1^{F_\omega|_H}(v)| = |N_1^{F_1}(\langle a, b\rangle)| = 8$, since only the vertex component of F_1 changes in a connected component C in $F_\omega\ |_H$. Thus C is isomorphic to F_1. Finally we have for A that $|N_1^{F_\omega|_A}(v)| = |N_1^{K_\omega}(c)||N_1^{F_1}(\langle a, b\rangle)| = 8(\omega - 1)$. Each neighbor in K_ω has an edge to each neighbor in F_1, with Corollary 3.10. So we have $|N_1^{F_\omega}(v)| = (\omega - 1) + 8 + 8(\omega - 1) = 8 + 9(\omega - 1)$.

To prove the second statement, we have that $|N_2^{F_1}(\langle a, b\rangle)| = 16$. To extend this to all layers, we use Corollary 3.10. There are ω layers in total, since each vertex of K_ω is in its own layer. Thus there are $\omega - 1$ vertical copies of each vertex in F_ω.

$$|N_2^{F_\omega}(v)| = |N_2^{F_1}(\langle a, b\rangle)|(\omega - 1) + |N_2^{F_1}(v)| = \omega|N_2^{F_1}(\langle a, b\rangle)| = 16\omega$$

.

We use this argumentation again for the third and fourth statement. For the third statement, let $w = \langle x, y, z\rangle \in V(F_\omega)$ such that $\langle x, y\rangle$ is in a corner of $N_{\leq 2}^{F_1}(\langle a, b\rangle)$-cuboid. The vertex $\langle x, y\rangle$ has five neighbors, which are not in $N_{\leq 2}^{F_1}(\langle a, b\rangle)$. With Corollary 3.10 we have that for $v \in V(F_\omega)$ and w in the corner of $N_{\leq 2}^{F_\omega}(v)$ that w has 5ω neighbors in F_ω.

For the fourth statement, let $w = \langle x, y, z\rangle \in V(F_\omega)$ such that $\langle x, y\rangle$ is not a corner but on the edge of an $N^{F_1}_{\le 2}(\langle a, b\rangle)$-cuboid. Then $\langle x, y\rangle$ has three neighbors outside of $N^{F_1}_{\le 2}(\langle a, b\rangle)$. With Corollary 3.10 we have that w has 3ω neighbors in F_ω.

To prove the fifth statement we use this strategy, but we also use the same argumentation for space around the wanted vertices as presented by Biedl, Eppstein, and Ueckerdt [BEU23]. Let $u = \langle x, y, z\rangle \in F_\omega$ such that $\langle a, b\rangle, \langle x, y\rangle$ are in the middle of two 5×5 squares in F_1. Let w, h be two vertices in the corners, which are connected through an edge. Biedl, Eppstein, and Ueckerdt state that six vertices that are adjacent to either w or h and have at least distance 3 to $\langle x, y\rangle$ and $\langle a, b\rangle$ to force that the edge wh is diagonal [BEU23]. With Corollary 3.10, we need 6ω vertices to force wh to be aslope or diagonal. Since we do not know more about these blocks, the 6ω vertices should be distributed equally over the two blocks. So 3ω vertices are used per block and ω vertices have to be free, so that the corner of an other block can fit in. ■

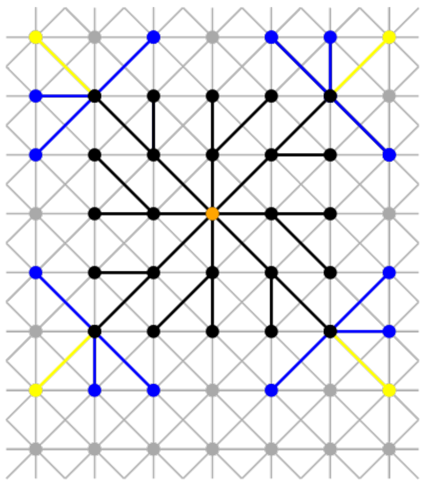
Now we construct a gadget $G_\omega(v)$ for a gadget tree T'. Figure 3.6 shows $G_\omega(v)$ embedded in F_ω. The idea and the gadget for $\omega = 1$ is the same as presented by Biedl, Eppstein, and Ueckerdt for proving that embedding a tree in F_1 is NP-complete [BEU23]. We apply Corollary 3.10. We construct a rooted tree $G_\omega(v)$. Let $\omega \in \mathbb{N}$ and v the root of $G_\omega(v)$. Let v have $(\omega - 1) + 8\omega$ children $u_1, \dots u_{(\omega-1)+8\omega}$. Let $u_1, \dots u_{8\omega-4}$ have two children each. Let $u_{8\omega-3}, \dots, u_{8\omega}$ have two children u^1_j, u^2_j each with u^1_j having 3ω children for $j \in \{8\omega - 3, 8\omega - 2, 8\omega - 1, 8\omega\}$. To construct the gadget tree T' we use Lemma 3.1. Note that G_ω is a tree by construction. We have four vertices of degree $3\omega + 1$, i.e. w^1_j which we use as connecting vertices. Note that, every gadget contains a constant number of vertices, since ω is given by F_ω. Thus, constructing T' from T is possible in polynomial time.

Theorem 3.12: *Let T be a tree, $\omega \in \mathbb{N}$ arbitrary and T' a gadget tree as defined above. Then it holds*

$T \subseteq R_1$ if and only if $T' \subseteq F_\omega$.

Proof. Let $T \subseteq R_1$. We show that there is an embedding φ of T' in F_ω. To embed a gadget $G(v)$. We embed every child of v that is a leaf as vertical copy of $\varphi(v)$. Let $w_1, \dots, w_4$ be the connecting vertices. Embed every vw_i- path P_i such P_i is in the same layer $\varphi(v)$ and $\varphi(w_i)$ is in a corner of $N^{F_\omega}_{\le 2}(v)$. Let z_i the vertex on P_i with degree 2. We have that z_i has another child z^1_i in $G(v)$. Embed z^1_i in the same layer as z_i and such that $\varphi(z^1_i)$ is in $N^{F_\omega}_2(v)$ and in the right side of w_i. Embed all children of w_i outside of $N^{F_\omega}_{\le 2}(v)$ as shown in Figure 3.6a. Embed the remaining children $u_1, \dots, u_{8\omega-4}$ of v arbitrary, since they all have two children u^1_i, u^2_i each. Embed u^1_i u^2_i as shown in Figure 3.6. Two gadgets are connected through a diagonal or slanted edge. Thus φ is an embedding of T' in F_ω.

To show the other direction let $T' \subseteq F_\omega$ with an embedding φ. To show that T is embeddable into R_1 it is sufficient to show that the construction of the gadgets and T' already enforces, that for every two adjacent gadgets $G(u)$ and $G(v)$ the $\varphi(u)\varphi(v)$-path is a shortest path of length 5. Having this assumption one can identify each gadget with its root in T and the connecting paths with edges in T. Since we only have four connecting vertices per gadgets, each gadget has at most four adjacent gadgets. To do that, let $v \in V(T)$. With Lemma 3.11 (1), we have that $|N^{F_\omega}_1(\varphi(v))| = 8 + 9(\omega - 1) = |N^{G_\omega(v)}_1(v)|$. Thus the complete 1-hop-neighborhood of $\varphi(v)$ in F_ω is covered by $\varphi(N^{G_\omega(v)}_1(v))$. Since 8ω children of v have two children, we have that $|N^{G_\omega(v)}_2(v)| = 2 \cdot 8\omega = 16\omega$. By Lemma 3.11 (2) we have that this is exactly $N^{F_\omega}_2(v)$. Thus the $\varphi(N^{G_\omega(v)}_{\le 2}(v))$ covers $N^{F_\omega}_{\le 2}(\varphi(v))$ completely. We have that $N^{F_\omega}_{\le 2}(\varphi(v))$ is a cuboid of size


(a) The embedding of $G(v)$ within the layer where v is embedded. Note that v is presented in orange. The blue edges are all incidient to the connecting vertex. The yellow edge shows where a connecting edge might be.

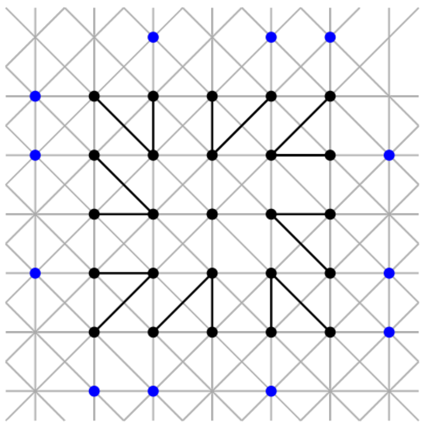

(b) The embedding of $G(v)$ on a layer where v is not embedded. Note that the vertical copy of v is connected to v and the whole 1-hop neighborhood is also connected to v.

Figure 3.6: The embedding a gadget $G_\omega(v)$ divided into different sorts of layers. The blue vertices of Figure 3.6b are vertical copies of the blue vertices of Figure 3.6a and are all connected to the connecting vertices in Figure 3.6a. The whole 1-hop-neighborhood of v is connected to v.

$5 \times 5 \times \omega$, if we identify a layer with a $\mathbb{N} \times \mathbb{N}$. Every connecting vertex has $3\omega + 1$ adjacent vertices outside of $N_{\leq 2}^{F_\omega}(\varphi(v))$. With Lemma 3.11 (3) and (4) we have that each connecting vertex can only be placed in the corner of $N_{\leq 2}^{F_\omega}(\varphi(v))$. With Lemma 3.11 (3) we have that two connecting vertices cannot be placed in the same corner, since they need $2 \cdot 3\omega$ adjacent vertices outside $N_{\leq 2}^{F_\omega}(v)$ but there are only 5ω. From this follows that every gadget $G(v)$ covers the whole at-most-2-neighborhood of their root v and that every two adjacent gadgets $G(u), G(v)$, $\varphi(u)$ and $\varphi(v)$ are connected through a shortest path of length 5. As already mention we find an embedding of T in R_1. ∎

With this equivalence we show that the problem $\mathcal{R}$ of deciding whether a tree is embeddable in F_ω is NP-complete.

Theorem 3.13: *Deciding whether a tree T is embeddable in the ω-full grid is NP-complete.*

Proof. Note that by Lemma 3.2 $\mathcal{R}$ is in NP. So it remains to show that $\mathcal{R}$ is NP-hard. To do this, we reduce $\mathcal{R}$ to the problem $\mathcal{P}$ of deciding whether a tree T is embeddable in R_1. For the reduction we build a gadget tree T' of T. As given above, the construction of T' is possible in polynomial time. Theorem 3.12 gives us the equivalence of $\mathcal{P}$ and deciding whether T' is embeddable into F_ω. Thus, we have that $\mathcal{R}$ is NP-complete. ∎

4 Embedding a Caterpillar in Different Products of Two Graphs and a Clique

In this chapter we show that deciding whether a caterpillar is embeddable into the ω-rectangle grid or the ω-full grid respectively is possible in linear time. By showing this, we generalize the statement that a deciding whether a caterpillar is embeddable in F_1 is possible in linear time presented by Biedl, Eppstein, and Ueckerdt [BEU23]. We use the notation for the ω-full grid and the ω-rectangle grid as in Chapter 3. We define a *caterpillar* C as a path S, where every vertex of S is adjacent to either other vertices on S or leaves. We say that S is the *spine* of the caterpillar and every vertex that is not on the spine is a *leg*. We denote the edges as *diagonal* that distinguish R_1 and F_1. For F_ω we define that an edge uv is *diagonal aslope* if u and v are in different layers and the edge uv' is diagonal for v' being the vertical copy of v on the layer as u.

The following proofs and statements follow closely the respective theorem presented by Biedl, Eppstein, and Ueckerdt, but with different values for the most parts [BEU23].

4.1 Embedding a Caterpillar in the ω-Rectangle Grid

In this section we show that embedding a caterpillar C in a ω-rectangle grid R_ω is solvable in linear time. We use a similar argument as stated by Biedl, Eppstein, and Ueckerdt [BEU23]. The main difference is, that a rectangle grid does not have diagonal nor slanted edges thus the legs of the caterpillar cannot interfere with each other.

Theorem 4.1: *Let C be a caterpillar, $\omega \in \mathbb{N}$ then the following are equivalent*

(1) $C \subseteq P_\infty \Box P_\infty \Box K_\omega$.

(2) C can be embedded such that all spine edges are horizontal and the spine is a straight path.

(3) Each vertex on the spine has at most $\omega + 3$ neighbors.

Proof. The implication (2) to (1) follows immediately since (2) already provides an embedding of C.

Now we show the implication (1) to (3). Let $v \in V(R_\omega)$. With Lemma 3.5 we have that $|N_1^{R_\omega}(v)| = (\omega - 1) + 4 = \omega + 3$. Then, each vertex on the spine of the caterpillar can have at most $\omega + 3$ neighbors and (3) holds.

In the following we show (3) to (2). Consider a caterpillar C with spine Q that satisfies (3). We show that there exists an embedding φ of C in R_ω. Embed Q as a straight path in one layer of R_ω. Note that only embedding Q without any legs of the caterpillar is possible, since Q is only a path. Let $u, v \in V(R_\omega)$ and $uv \in E(R_\omega)$ horizontal. Then we have that $N_1^{R_\omega}(u) \cap N_1^{R_\omega}(v) = \emptyset$. This holds because in one layer there are no triangles, thus u and v cannot share neighbors in their own layer. We have that u and v are only adjacent to their respective vertical copies in other layers. Since u and v are in the same layer, there is no common neighbor of $\varphi(u)$ and $\varphi(v)$ in another layer. We have that each vertex v on the spine, except the end points, have $\omega + 1$ neighbors in R_ω which are not on the spine and v has two neighbors on the spine. So v has $\omega + 3$ neighbors in C. The end vertices x, y have $\omega + 2$ neighbors not on the Q and one neighbor on Q, thus x and y also have $\omega+3$ possible neighbors. Since (3) states that each vertex on Q has at most $\omega + 3$ neighbors, C can be embedded such that $\varphi(Q)$ is a straight and horizontal path. ■

We justify the linear running time by using an adjacency list for saving the caterpillar. Then we can determine the degree of every vertex in linear time. Since Theorem 4.1 (3) states that every vertex can have at most degree $\omega + 3$, it is sufficient to look at each vertex separately.

4.2 Embedding a Caterpillar in the ω-Full Grid

We show that deciding whether a caterpillar is embeddable in the ω-full grid is solvable in linear time. The statement and proof are nearly the same as presented by Biedl, Eppstein, and Ueckerdt but here we consider ω layers, which is visible in (3) and some adjusted values in the proof [BEU23].

Theorem 4.2: *Let C be a caterpillar, $\omega \in \mathbb{N}$, then the following are equivalent:*

(1) $C \subset P_\infty \boxtimes P_\infty \boxtimes K_\omega$

(2) C can be embedded such that all spine edges are diagonal or slanted diagonal.

(3) For every subpath P of the spine S of C, it holds $\sum_{v\in V(P)} deg(v) \leq (5\omega+1)|V(P)|+4\omega-2$.

Proof. First we show the implication from (1) to (3). Let φ be an embedding of C in F_ω. Let P be any subgraph of the spine S of G and $w \in V(C)$ With Lemma 3.11 (1) we have that $|N_1^C(w)| \leq |N_1^{F_\omega}(\varphi(w))| = (\omega - 1) + 8\omega$. For any $uv \in E(F_\omega)$ with $u = \langle a, b, c\rangle, v = \langle a', b', c'\rangle$ we have that $|N_1^{F_\omega}(u) \cap N_1^{F_\omega}(v)| \geq 2\omega + 2(\omega - 1)$. This equation holds because $|N_1^{F_1}(\langle a, b\rangle) \cap N_1^{F_1}(\langle a', b'\rangle)| \geq 2$, since $\langle a, b\rangle\langle a', b'\rangle \in E(F_1)$ is diagonal, they have two vertices that are adjacent to both $\langle a, b\rangle$ and $\langle a', b'\rangle$. With Corollary 3.10 we have that u and v are also neighboring all vertical copies of the vertices in the common neighborhood of $\langle a, b\rangle$ and $\langle a', b'\rangle$. Also with Corollary 3.10 u and v are both adjacent to the vertical copies of both $\varphi(u)$ and $\varphi(v)$. Since caterpillars contain no triangles we have that for $x, y \in V(C)$ $|N_1^G(x) \cap N_1^G(y)| = 0$. Thus, for each $uv \in E(P)$ we have that their neighborhood has not full size each. So we can estimate the degrees of vertices of P.

$$\begin{aligned}\sum_{v\in V(P)} deg(v) &\leq ((\omega - 1) + 8\omega)|V(P)| - (2\omega + 2(\omega - 1))|E(P)|\\ &= ((\omega - 1) + 8\omega)|V(P)| - (2\omega + 2(\omega - 1))(|V(P)| - 1)\\ &= (6\omega - (\omega - 1)|V(P)| + 2\omega + 2(\omega - 1)\\ &= (5\omega + 1)|V(P)| + 4\omega - 2.\end{aligned}$$

Now we show the implication from (3) to (2).

Let C be a caterpillar with spine $S = v_1v_2...v_n$. Note that we identify both P_∞ components with $\mathbb{Z}$ and we numerate the vertices of the K_ω. Consider an embedding φ of C in F_ω by with $\varphi(v_i) = \langle i, i, k\rangle$. Note that the vertex component of K_ω does not matter because of Corollary 3.10, so we just leave it out of the notation. Every leaf is embedded on vertex $\langle i, j\rangle$ with $i + j$ as small as possible. We have that for the first vertex v with degree of at least

$2 + 2\omega + (\omega - 1)$ the adjacent leaves are embedded on $\langle i, i-1\rangle, \langle i-1, i\rangle, \langle i-1, i-1\rangle$, but note that on $\langle i-1, i-1\rangle$ might also be another vertex of the S. If each vertex of S has degree less than $1 + 3\omega$, one can embed C by this construction. Assume for the sake of contradiction that there is a vertex v_j of S such there is a leaf of v_j is not embedded by φ. Then we have $\deg_G(v_j) \geq 1 + 3\omega$ otherwise there is not enough space which cannot be used by any other vertex on S. Let v_i be the vertex with the largest index $i \leq j$ such that v_i has leaves embedded on $\langle i-1, i-1\rangle, \langle i, i-1\rangle, \langle i-1, i\rangle$. Note that $\deg_G(v_i), \deg_G(v_j) \geq 1+3\omega$. Let $P = v_i \dots v_j$ be the subpath of the spine. We have that every vertex in $V(\varphi(P)) \cup N_1^{F_\omega}(\varphi(P))$ is covered by a vertex of $V(P) \cup N_1^G(P)$. So, $|V(\varphi(P)) \cup N_1^{F_\omega}(\varphi(P))| = (5(j-i)+9)\omega$. This holds with $|N_1^{F_1}(P)| = (5(j-i)+9)$ and Corollary 3.10. We have that $|V(P) \cup N_1^G(P)| = \sum_{v \in V(P)} deg(v) - (j-i+1)$, since every vertex on P only appears once. Then it follows:

$$
\begin{aligned}
&|V(P) \cup N_1^G(P)| \geq |V(\varphi(P)) \cup N_1^{F_\omega}(\varphi(P))| \\
\iff &\sum_{v \in V(P)} deg(v) - (j-i+1) \geq (5(j-i)+9)\omega \\
\iff &\sum_{v \in V(P)} deg(v) \geq (5(j-i)+9)\omega + (j-i+1) = ((5\omega+1)(j-i+1) + 4\omega \\
= &(5\omega+1)|V(P)| + 4\omega.
\end{aligned}
$$

Therefore P does not satisfy the assumption from (3).

The implication from (2) to (1) holds immediate, because (2) already assumes that G is embedded in F_ω. ∎

To justify the linear running time for deciding whether a caterpillar is embeddable into F_ω, we embed use Theorem 4.2 (2). For this we use the idea of embedding the caterpillar C as presented in the proof. Since we identify P_∞ with $\mathbb{Z}$, we start by embedding the first vertex of the spine S of C on a vertex (0,0,x), $x \in [\omega]$. Let $\varphi(v) = (j, j, x)$ for each vertex v on S of C and $j > 0$. Embed every other vertex v of C in such a way that for $\varphi(v) = \langle a, b, c\rangle$ $a + b + c$ is minimal. From the proof of Theorem 4.2 (2) we have that this way of embedding provides enough space to embed all edges if C is indeed embeddable. Since we only have to look at every vertex and edge of C once to embed them, we only need linear time.

5 Embedding Graphs in the Strong Product of Two Paths

In this chapter we show that embedding a simple and connected graph G into the strong product of a path of infinite length and a path of any given constant length can be done in polynomial time if G is embeddable. Note that Biedl, Eppstein, and Ueckerdt showed that embedding a tree in a 1-full grid is NP-complete, so it is necessary that the second path is finite and the length is given ([BEU23]). We also show that embedding a tree T in a grid of height 2 is possible in linear time, if T is embeddable.

We define $P_\infty \boxtimes P_k$ as *k-wide crossed ladder* L_k for some $k \in \mathbb{N}$. We abbreviate L_2 by *crossed ladder*. We give some notation for the k-wide crossed ladder that follows from Figure 5.1.

Recall that we identify P_∞ with $\mathbb{Z}$, so we have that the vertices of the L_k numbered in ascending order from left to right. Let $u = \langle a, b \rangle, v = \langle x, y \rangle \in V(L_k)$ with $a, x \in V(P_\infty)$. We say u is *left* of v, if $a < x$. If $a > x$ we say u is *right* of v. We call u the *vertical copy* of v if $a = x$. We define a *column* by the induced subgraph K of L_k, where all vertices are vertical copies of each other. An edge connecting two vertical copies with each other we call *vertical*.

In this chapter we talk about subtrees of trees T starting at a given vertex v. By that we refer to all connected components which occur when deleting the vertex v from T and add v to each component as a leaf.

We first show that embedding a tree in the crossed ladder is possible in polynomial time. We refine the presented algorithm and show that embedding a tree in the crossed ladder is possible in linear time. After we show that deciding whether a simple and connected graph is embeddable in L_k is possible in polynomial time. We see that the running time also depends on k.

Note that both algorithms assume that the respective graph G to embed is connected. For disconnected graphs we use the algorithms on each component and G is embeddable only if every connected component of G is embeddable. This is sufficient, because the P_∞ component of L_k provides enough space to embed every graph component in a sequential way.

5.1 Embedding a Tree in the Crossed Ladder

We show that embedding a tree in a crossed ladder is possible in polynomial time. To do so, we normalize the embedding of a tree in L_2 and present an algorithm that finds an embedding if the tree is embeddable. After this, we give a modification of the presented algorithm and show that embedding T in L_2 is possible in linear time.

Let P_2 be the path of length 1 with $V(P_2) = \{a, b\}$. We define some notation for referring to different parts of the crossed ladder L_2, that is made clear by looking at Figure 5.2.

We define the *top layer* by the induced subgraph of L_2 where every vertex contains the vertex component a. We define the *bottom layer* by the induced subgraph of L_2 where every vertex contains the vertex component b. We say an edge is *diagonal* if it connects two vertices

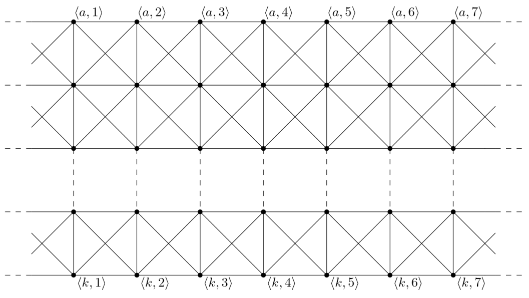


Figure 5.1: A k-wide crossed ladder.

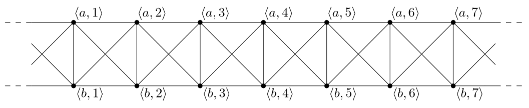


Figure 5.2: A crossed ladder or 2-wide crossed ladder. The vertices are numerated in an ascending way from left to right. This is the representation used in the proof.

in the bottom and the top layer and is not vertical. We call an edge e *horizontal* if e is in either the top or the bottom layer. We define an diagonal edge uv as *backwards*, if u is left of v. We say that uv is *forwards* if u is right of v.

The next lemma provides a normalization of the embedding of a tree in L_2 that we use as ground structure for the algorithm.

Lemma 5.1: *For every embedding φ of a tree T with a leftmost and a rightmost vertex $\varphi(\ell), \varphi(r)$ respectively in the crossed ladder, there is an embedding ψ that embeds the ℓr-path in the top layer and $\psi(\ell)$ and $\psi(r)$ are the leftmost and rightmost vertices respectively.*

Proof. Consider an embedding φ of a tree T in the crossed ladder with a leftmost and a rightmost vertex $\varphi(\ell), \varphi(r)$ respectively. Then we have that no other vertex of T is embedded on the left of $\varphi(\ell)$ nor on the right of $\varphi(r)$. Starting from φ, we give a construction of a new embedding ψ where the ℓr-path P is embedded in the top layer and every other vertex of T is embedded on the bottom layer.

We assume that $\varphi(\ell)$ is already in the top layer, otherwise we flip $\varphi(T)$ by replacing each vertex $\varphi(x)$ by its vertical copy. In the following we give a construction of φ' that ensures that P is embedded in the top layer. If P is not already embedded in the top layer, then there are vertical, diagonal backward or diagonal forward edges that change the layer. For each of the edge types, we give a transformation of φ to φ' such that this edge type is replaced by another type.

First, we look at the case that $\varphi(P)$ contains a diagonal forward edge e. We transform e to a horizontal edge. To do that, we show a more general statement: Let G be a graph. Let χ be an embedding of G in L_2 such that $\chi(x)$ is a vertical copy of $\chi(y)$, with $\chi(x)$ in the top layer and $\chi(y)$ in the bottom layer for $x, y \in V(G)$. Then we have that there is an embedding χ' such that $\chi'(x)$ is in the bottom layer and $\chi'(y)$ is in the top layer and $\chi(z) = \chi'(z)$ for all $z \in V(G)$ with $z \neq x$ and $z \neq y$.

To show the general statement, we examine the 1-hop-neighborhood of $\chi(x)$ and $\chi(y)$ in L_2. Note that $N_1^{L_2}(\chi(x)) \cup \{\chi(x)\} = N_1^{L_2}(\chi(y)) \cup \{\chi(y)\}$. Therefore changing vertical copies of each other is only a local operation, so the rest of χ, which is not in the 1-hop-neighborhood of neither $\chi(x)$ nor $\chi(y)$ just stays the same for χ'. We have that each diagonal edge incident to $\chi(x)$ or $\chi(y)$ changes to an horizontal edge and each horizontal edge to a diagonal edge, since only the P_2 component of $\chi(x)$ and $\chi(y)$ change.

We apply that result to φ having a diagonal forward edge $\varphi(u)\varphi(v)$. With this claim set $\varphi(x) = \varphi'(y)$ and $\varphi(y) = \varphi'(x)$. We do that for every forward diagonal edge in $\varphi(P)$. Then we get an embedding φ' without any forward vertical edges.

For the following two cases, assume that $\varphi(P)$ contains a vertical edge or a diagonal backward edge $\varphi(u)\varphi(v)$. We treat both cases similar. We replace a vertical edge by a diagonal forward edge and the diagonal backward edge by a vertical edge. Both of them require that $\varphi(v)$ is translated by one vertex to the right. To obtain a new embedding φ', we divide T into three subtrees L, M and R. Let u' be the left neighbor of u in P and let v' be the right neighbor of v in P. We delete the edges $u'u$ and vv' from T, then we get a forest T' with three connected components. Let L be the connected component of T' containing u', M be the connected component fo T' containing uv and R be the connected component containing v'.

In the following we construct an embedding φ' of M, L and R from φ and show that φ' is again an embedding of T in L_2. Assume for the sake of contradiction that the rightmost vertex from $\varphi(L)$ is not on the left of $\varphi(R)$. Then we find vertices a and a' such that $a \in V(\varphi(L))$ and $a' \in V(\varphi(R))$ with a and a' being vertical copies of each other. For the first case assume that $\varphi(u)\varphi(v)$ is vertical. Since $\varphi(u)\varphi(v)$ is between $\varphi(L)$ and $\varphi(R)$ and there is no edge crossing a vertical edge in L_2, that contradicts the assumption that such a and a' exist. Now assume that $\varphi(u)\varphi(v)$ is a diagonal backward edge. Then we have that the vertical copies of $\varphi(u)$ and $\varphi(v)$ both belong to $\varphi(P)$. From this we have that a and a' do not exist.

Set $\varphi'(L) = \varphi(L)$ and $\varphi'(R)$ is the translation of $\varphi(R)$, by one vertex to the right. φ' is a proper embedding of R and L, since $\varphi(R)$ is entirely to the right of $\varphi(L)$. Further shifting to the right will not interfere with $\varphi(L)$.

Now it remains to show that $\varphi'(M)$ is an embedding. Let $\varphi(u) = \varphi'(u)$ and $\varphi'(v)$ is the same as $\varphi(v)$ but translated by one vertex to the right. When looking at $\varphi(M)$, let S_L be the subtrees starting in u and $\varphi(S_L)$ is on the left of $\varphi(u)$. Then we set $\varphi'(S_L) = \varphi(S_L)$. Let S_R be the subtrees starting in v and $\varphi(S_R)$ is on the right of $\varphi(v)$, then set $\varphi'(S_R)$ is the same as $\varphi(S_R)$ but translated by one vertex to the right. The embedding of S_L and S_R is possible since the the rest of φ' is locally the same as φ.

We have that $\varphi(u)$ has one adjacent vertex on the right u_R that is not in $\varphi(P)$. Note that the vertical copy of u_R is used by $\varphi(x)$ with $x \in V(P)$. Assume that u has a child vertex z with $\varphi(z) = u_R$. Let P_R be the subpath starting in u, with second vertex z. Then we have that the vertices in $V(\varphi(P_R))$ translated by one to the right are not used in φ'. Then we have that there is a path P_{free} of length $|P_R| + 1$ in L_2 starting in $\varphi'(u)$, that is not covered by φ. Then we embed P_R in P_{free} in φ'. For v we consider a similar case: We have that $\varphi(v)$ has one adjacent vertex on the left v_L that is not in $\varphi(P)$. Note that the vertical copy of v_L is used by $\varphi(x)$ with $x \in V(P)$. Assume that v has a child vertex a with $\varphi(a) = v_L$. Let P_L be the subpath starting in v, with second vertex a. Then we have that the vertices in $V(\varphi(P_L))$ are not in φ'. We also have that $\varphi(v)$ is translated by one to the right for φ'. Note that $\varphi(v)$ is not used by φ'. Then we have that there is a path P_{free} of length $|P_L| + 1$ in L_2 starting in $\varphi'(v)$, that is not covered by φ. Then we embed P_R in P_{free} in φ'.

Now we can use these transformations to get an embedding ψ from φ, with ψ is an embedding of T where $\psi(P)$ is embedded in the top layer. To do so, make sure, that $\psi(\ell)$ is in the top layer. Then iterate from left to right in $\varphi(P)$ and use the presented transformations until each edge of P is embedded horizontally. ∎

We describe an algorithm `FindCrossedLadderEmbedding` to find an embedding φ that embeds a tree T with only a path on the top layer if T is embeddable. So, let T be a tree that should be embedded. First, check whether the maximum degree of T is at most 5. If the degree is higher, the tree cannot be embedded. Then, iterate over all tuples of vertices. For each tuple (ℓ, r) embed the ℓr-path P in the top layer. Check if P contains at least half of the vertices of T. If this is not the case continue with the next tuple. Then, we use some sort of line sweep algorithm. Starting from ℓ iterate over each vertex in P and consider the subtrees that have the current vertex as root and do not contain any vertex of P and embed them as far to the left as possible. To do that, we save two values. The first value is the length of the path in the bottom layer of L_2 that is not in the image of the current embedding and ends on the vertical copy of current vertex. We say that these vertices are *unused.* We define *needed vertices* as the number of vertices that are used in the bottom layer and are on the right of the current vertex.

Let v be the current vertex. Let $\mathcal{S}$ the set of subtrees starting in v and not containing any other vertex of P. We have that v has at most three children. If v has no children we continue with the next vertex and increase the number of unused vertices.

If v has one child w we have a closer look at the subtree S. There are two possible shapes of S. For the first shape we have that w has two children w_1, w_2 itself. Both of w_1, w_2 start a path P_1, P_2 maybe of length 1. We embed P_1 and P_2 such that as many as possible unused vertices are now in the image of φ. Next we update the unused vertices and needed vertices. For the second shape assume that S is a path. Then we look at the following vertex u on P. If this u has exactly one subtree S_u that is a path, we try to embed S and S_u such that as many unused vertices as possible are used by φ. Update the needed and the unused vertices. We do not continue with looking at u in the line sweep scheme, but with the next vertex after u. If u has a different number of children or S_u is not a path, we try to embed S such that as many vertices of the unused vertices as possible are used in φ.

If v has two children u, w we have that the subtrees S_u, S_w starting in u and w respectively are paths. Then we embed S_u, S_w such that as many unused vertices as possible are used by φ and update the unused and needed vertices.

If v has three children, let S_1, S_2, S_3 be the subgraphs starting in one of the children of v each. We check without loss of generality that S_1, S_3 are paths and S_2 is a leaf. We embed S_2 as the copy of $\varphi(v)$ and embed S_1, S_3 such that as many unused vertices as possible are used.

When arriving at r make sure that there are no needed vertices left. If this is the case, there is an embedding φ with ℓ as leftmost vertex and r as the rightmost vertex. Otherwise start with the next tuple. If all tuples are used and no embedding is found, then there is no embedding of T in L_2.

Lemma 5.2: *Let T be a tree. The algorithm `FindCrossedLadderEmbedding` described above finds an embedding φ of T if T is embeddable in $P_\infty \boxtimes P_2$.*

Proof. With Lemma 5.1 we have that it is sufficient to find an embedding φ of T that embeds a path P in the top layer of L_2 where the end points of P are embedded as the leftmost and rightmost vertices.

The algorithm iterates through all possible tuples of vertices so it checks all possible paths. For the following proof, consider a path P with end points ℓ and r.

First we distinguish the different subtrees having a vertex on P as root. Each embedding of a vertex $v \in V(P)$ has at most three children in the crossed ladder, because every vertex on the top layer has three neighbors on the bottom layer. By assumption the children of v are always embedded in the bottom layer. Each subtree S starting in a vertex of P is either a path or v has a child u that has two children itself. If u has two own children u_1 and u_2 we have that the subtrees starting in u_1 and u_2 are paths. These are the only two possibilities because every vertex on the bottom layer has only two neighbors on the bottom layer and S does not have an edge changing the layer since otherwise either $\varphi(\ell)$ or $\varphi(r)$ is not the left or rightmost vertex or there is a circle.

Note that the algorithm is greedy. We show that `FindCrossedLadderEmbedding` finds a feasible embedding and that if `FindCrossedLadderEmbedding` finds an embedding φ, then φ is optimal. To show that the solution is optimal we use the exchange argument. Let T be a tree with an optimal embedding φ_{opt}, that is not reachable by `FindCrossedLadderEmbedding`. Note that φ_{opt} also embeds only a path in the top layer with $\varphi_{opt}(\ell)$ as the leftmost vertex and $\varphi_{opt}(r)$ as the rightmost vertex for some $\ell, r \in V(T)$. Let P be the ℓr-path. We have that φ_{opt} is not reachable by `FindCrossedLadderEmbedding`, if there is a vertex a on P with a subtree S such that S is not embedded as far to the left as possible. That means, that there is another embedding that embeds S to the left of $\varphi_{opt}(x)$ for $\varphi_{opt}(x)$ being the rightmost vertex of $\varphi_{opt}(S)$. Take the leftmost vertex a for which there is an S that is not embedded as far to the left as possible by φ_{opt}. Let φ_{Gr} be a greedy embedding of T. Note that `FindCrossedLadderEmbedding` tries the tuple (ℓ, r) as leftmost and rightmost vertex for the path embedded on the top layer. Thus we consider that $\varphi_{Gr}(\ell)$ is the leftmost vertex and $\varphi_{Gr}(r)$ is the rightmost vertex of the greedy embedding of T. By assumption we have that $\varphi_{Gr}(S)$ is as far to the left as possible. In the following we divide P into two different subgraphs, so that we combine φ_{Gr} and φ_{opt} to a new embedding of T. Let P^r be a subpath of P starting at the right neighbor b of a and ending in r. Let P_ℓ be the subpath of P from ℓ to a. Construct T' by deleting uv from T. Let S^ℓ be the connected component of T' containing P^ℓ and let S^r be the connected component of T' containing P^r. We construct a new embedding ψ by joining $\varphi_{Gr}(P^\ell)$ and $\varphi_{Opt}(P_r)$. For combining $\varphi_{Gr}(P^\ell)$ and $\varphi_{opt}(P^r)$ we have to make sure that $\varphi_{Gr}(P^\ell)$ does not contain any vertices that are also in $\varphi_{opt}(P^r)$. Let v^r_{opt} be the rightmost vertex on the bottom layer of $\varphi_{opt}(S^\ell)$. Let v^ℓ_{opt} be the leftmost vertex on the bottom layer of $\varphi_{opt}(S^r)$. We define v^r_{Gr} and v^r_{Gr} like v^r_{opt} and v^ℓ_{opt} respectively. We look at two different cases. First assume that v^r_{opt} is on the left of v^ℓ_{opt}. By construction of S_ℓ we have that v^r_{Gr} is on the left of v^r_{opt}. Thus v^r_{opt} is on the left of v^ℓ_{opt}. From this follows that we can build an embedding ψ using $\varphi_{opt}(S^r)$ and $\varphi_{Gr}(S^\ell)$.

Now assume that v^r_{opt} is on the right of v^ℓ_{opt}. Let x be the first vertex on the $\varphi_{opt}(a)v^r_{opt}$-path. Then we have that $\varphi_{opt}(a)x$ is a diagonal forward edge. Otherwise it is not possible that v^ℓ_{opt} is on the left of v^r_{opt}. Since v^r_{Gr} is on the left of v^r_{opt}, we have that $\varphi_{Gr}(a)x$ is either vertical or diagonal backwards. In this case it is not possible for the vertex embedded on v^ℓ_{opt} to be embedded on the left of $\varphi_{opt}(a)$ or on the vertical copy of $\varphi_{opt}(a)$. From this follows that we can build an embedding ψ using $\varphi_{opt}(S^r)$ and $\varphi_{Gr}(S^\ell)$.

So, we have ψ which is reached by `FindCrossedLadderEmbedding` until at least a. By iterating from left to right through P and use of the exchange argument we produce a solution reachable by the algorithm. Therefore, if there is a solution with the defined endpoints there is also one, where the subtrees of each vertex on P are embedded as far to the left as possible.

Now we have to show that the algorithm indeed finds a solution which satisfies the claim, i.e. it finds a solution which is embedded as far to the left as possible. With Lemma 5.1 we have that by embedding a path on the top layer a solution can be found if there is one. Since the algorithm iterates over all pairs of vertices and chooses them as leftmost and rightmost vertex `FindCrossedLadderEmbedding` finds an embedding of T if there is one. Assume there is an embedding of T. By the algorithm we have that there is a leftmost and a rightmost vertex that are embedded on the top layer. Each other vertex is placed between these vertices either on the top or on the bottom layer. So we have a solution to embed T in the L_2. ■

After we prove that the algorithm decides the problem in polynomial time, we give an modification of the algorithm, that has only linear running time.

Theorem 5.3: *Let T be a tree. Finding an embedding φ of T in L_2, if one exists, is possible in polynomial time.*

Proof. We use FindCrossedLadderEmbedding. With Lemma 5.2 we have that this algorithm finds an embedding of T if T is embeddable in L_2. If T is not embeddable, then `FindCrossedLadderEmbedding` does not find an embedding.

It remains to show that `FindCrossedLadderEmbedding` terminates in polynomial time. First, we show that `FindCrossedLadderEmbedding` terminates. The outer loop iterates through every pair of vertices of T. Since $|V(T)| < \infty$, there are only finite iterations because we do not change T within the loop. In the loop we only look at each vertex once so we only need finite time within the loop body. Hence, `FindCrossedLadderEmbedding` terminates.

Now we show that `FindCrossedLadderEmbedding` needs polynomial time. First, checking whether the maximum degree is less or equal to 5 is possible in $O(n^2)$. The loop makes at most n^2 iterations. Within the loop we check the length of the current ℓr-path P. This is possible in $O(n)$. Then, the line-sweep algorithm starts. One step in the line sweep algorithm checks the size and the number of the subtrees $\mathcal{S}$ starting in the current vertex and not containing P. Each tree of $\mathcal{S}$ has at most length $\frac{n}{2}$, because otherwise there is not enough place to embed $\mathcal{S}$ between ℓ and r. This is possible in $O(n)$. Comparing the size of each $S \in \mathcal{S}$ with the saved number and determine which subgraph has to embedded where is in $O(1)$. Thus one step in the line sweep algorithm is in $O(n)$. The line sweep algorithm has at most n iterations so the whole line sweep algorithm needs $O(n) \cdot O(n) = O(n^2)$ time. One iteration of the outer loop needs $O(n^2) + O(n)$ time. For the whole loop we need $O(n^2) \cdot n^2 = O(n^4)$. So, in total the algorithm only needs $O(n^4) + O(n^2) = O(n^4)$ time. From this follows that `FindCrossedLadderEmbedding` finds an embedding of T in L_2 in polynomial time. ■

5.1.1 Linear Running Time for Embedding a Tree in a Crossed Ladder

In the following we give a modification of `FindCrossedLadderEmbedding`, which finds an embedding of a tree T in linear time. To do that, we construct ℓr-paths instead of trying each possible path. We also have a closer look at the inner loop and use a data structure when saving the tree, such that determine the degree of the vertices is easier.

In the following we give the construction of the ℓr-paths P and in Lemma 5.4 we give an explanation why this construction provides a path that can be used instead of trying all possible paths in `FindCrossedLadderEmbedding`.

Now, we construct the paths $\mathcal{P}$ we use in the algorithm instead of trying every possible path. To find $\mathcal{P}$ we construct the longest possible paths, that contains every vertex of degree at least 4 and sufficient vertices of degree 3.

First, we determine the degree of each vertex of the given tree T. Next, we find a path P' in T that contains every vertex with degree at least 4. Let v be an endpoint of P'. Then, we look at the subtrees $\mathcal{S}$ of T which are adjacent v but do not contain P' and determine the longest and most useful subpath to extend P' to P. Note that we do the following for both end points of P'.

We distinguish two cases for $\mathcal{S}$. First, assume that every subtree S in $\mathcal{S}$ maximal degree 2. i.e. every S is a path. Then, we take the longest path S_L of $\mathcal{S}$ and embed them in the top layer. Then we have that if there is an embedding of T every other subpath than S_L in $\mathcal{S}$ is now embeddable in the bottom layer, since if there is an embedding with a shorter path S' in the top layer one can change S' and S_L. Now, assume that at there is a tree $S \in \mathcal{S}$ such that S contains a vertex of degree 3. To find a path P_S to embed in the top layer, we first select a subtree S, in which we find P_S. If the end point v has degree 5 then there is exactly one subtree S_v containing a vertex of degree 3. Then we take S_v as S. Now assume that v is of degree 4. Then we have that at most two subgraphs S_1 and S_2 may contain a vertex of degree 3 each. If only S_1 contains a vertex of degree 3, then we take S_1 as S. In the following assume that both S_1 and S_2 both contain at least one vertex of degree 3. First we look at the neighbor v_1, v_2 of v in both S_1 and S_2 respectively. If both are of degree 3, T is not embeddable. If one of them is of degree 3, say v_1, then we select S_2 as S. If both v_1 and v_2 are not of degree 3 then T is not embeddable, since in this case there is a vertex w of degree 3, that is embedded on the bottom layer, without having an edge to the top layer. This is not possible.

In the following we determine the path P_S in S to embed in the top layer. We We want to that P_S contains every vertex of degree 3, which is not adjacent to another vertex of degree 3 and is as long as possible. To do this, we build an auxiliary graph G where every path in S between to vertices of degree other than 2 is replaced by an edge. Then we find a longest path P_ℓ in G, starting in v. Then we look at the second and third to last vertex v_1, v_2 of P_ℓ. Let v_3 be the neighbor of v_2, which is not on P_ℓ. If v_3 is of degree 1, then we choose the vv_1-path in S. Every subpath starting in v_1 is a path, by construction of G. We take the longest subpath P_{v_1} starting in v_1 and join the vv_1-path and P_{v_1} to P_S. If v_3 is of degree 3, there are two cases. Assume that $v_1v_2 \in E(T)$ and $v_2v_3 \notin E(T)$. Then we select the vv_3-path P_{v_3} and join P_{v_3} with the longest subpath starting in v_3 to P_S. If $v_1v_2 \notin E(T)$ and $v_2v_3 \in E(T)$, we select the vv_1-path P_{v_1} and join P_{v_1} with the longest subpath starting in v_1 to P_S.

If both of the edges $v_1v_2, v_2v_3 \in E(T)$, then we save two paths. The first path P_S^1 contains the vv_1-path in T and the longest subpath starting in v_1 and the second path is P_S^3 containing the vv_3-path in T and the longest subpath starting in v_3. To construct P, we join P_S and P'. Note that we have at most four different paths P, since for each end vertex v we get at most 2 different P_S.

Lemma 5.4: *Let T be a tree. Then we find an embedding φ of T in L_2 with a path P as described above on the top layer if and only if there exists an embedding of T in L_2.*

Proof. First, note that if there is an embedding with P on the top layer, then we have an embedding of T in L_2.

Now assume that we have an embedding $\tilde{\varphi}$ of T and let $\tilde{\varphi}(\ell), \tilde{\varphi}(r)$ the leftmost and the rightmost vertex in the embedding respectively.

By Lemma 5.1 we find an embedding φ' of T with the ℓr-path $\tilde{P}$ embedded on the top layer. First, we show that every vertex of T with degree at least 4 is in $\tilde{P}$. Then we give a transformation from $\tilde{P}$ to P.

Note that every vertex in L_2 has degree 5. Every vertex in L_2 is adjacent to two vertices in the same layer. Since every vertex v of T embedded in the bottom layer is adjacent to at most one vertex in the top layer, we have that v has at most degree 3. Since we φ' embeds T with $\tilde{P}$ in the top layer, $\tilde{P}$ contains every vertex of T with degree at least 4. Note that the path P' between the leftmost and the rightmost vertex of degree 4 respectively is unique. Thus, P' is a subpath of both $\tilde{P}$ and P.

In the following we show that transforming $\tilde{P}$, where $\tilde{P}$ is not P' preserves that we still find an embedding φ of T. Let v be an endpoint of $\tilde{P}$. Then, every subtree $\mathcal{S}$ staring in v, but not containing $\tilde{P}$, only contains vertices of degree at most 3. Let P_{algo} be the vr-path or the $v\ell$-path respectively selected by `FindCrossedLadderEmbedding` and let $P_{\varphi'}$ be the vr-path or the $v\ell$-path respectively given by φ'. If P_{algo} is the same path as $P_{\varphi'}$, then nothing has to change. Assume that $P_{\varphi'}$ and P_{algo} are in different subtrees starting in v. Let T_e be the subtree containing P_{algo} and T_a be the subtree containing $P_{\varphi'}$. In the following we show that there is a transformation form T_e to T_a. First, we build an auxiliary graph of T as described above. Note that T_e and T_a are different, if they are different in G.

We have a closer look at vertices of degree 3 within an arbitrary embedding ψ of T in L_2. If a vertex v of degree 3 is embedded on the bottom layer, both of the subgraphs S_1, S_2 starting in v on the bottom layer are paths. Otherwise there is edge going to the top layer. Thus in G we find a path P_ℓ such that for every vertex v of degree 3 on P_ℓ there is a subpath starting S_v in v of length at most 2. Note that S_v goes to the bottom layer and contains at most one vertex of degree 3.

Assume that the paths P_a, P_e found in G for T_a and T_e respectively in T differ at a vertex v which is at least the fourth to last vertex on P_ℓ. Then we have that at least two vertices u, w of degree 3 are in the bottom layer in T_e. By construction we have that u and w are in the same path, but this is not possible. So, we know that P_a and P_e differ in the last two vertices. Let v the last vertex which is chosen on both paths P_a and P_e. Let v_a the selected neighbor of v by the algorithm and v_e the selected vertex by φ'. From thus follows that $vv_a \in E(G)$ and $vv_e \in E(G)$. If one of vv_e and vv_a does not exist in T, then the respective vertex v_a, v_e is in the P_ℓ. Otherwise there is no embedding, because in this case there is a vertex of degree 3, that is embedded in the bottom layer without an edge to the top layer. This is not possible. If both edges exists, then we have that both v_a, v_e start paths, that are usable in the algorithm, so the only difference is in selecting the last vertex. Since the algorithm only selects the longest path ending, it is possible to change the selected path.

Thus we have a transformation from T_e to T_a. ∎

To use this construction of paths in the algorithm, we need that the construction of the paths is possible in linear time. To do that, we have a closer look at the used algorithms and data structures.

Lemma 5.5: *Constructing a path P of a tree T as described above is possible in linear time.*

Proof. We assume that T is given as adjacency list AL_T. Then we iterate over AL_T and determine the degree of every vertex. While doing that we save a reference to every vertex with degree at least 4. This is possible in $O(n)$, since determining the degree of each vertex in an adjacency list is possible in $O(n + m)$, but for a tree it holds that $m = n - 1$.

Next, we have to find a path, which contains every vertex with degree at least 4. Note that every two vertices in a tree are connected through a unique path. So, we select one of the vertices of degree at least four. Starting there, we do a modified depth first search (mDFS). The mDFS finds a path P which contains every vertex with degree at least 4, by skipping every vertex with degree 2 during its backtracking step. First, we mark every vertex of degree at least 4, we say *big vertex*, as not found. We also initialize a *found path*, as an empty path. During the mDFS we add already confirmed paths to the found path, to construct P. We start the mDFS at a big vertex v and mark v as found. When exploring a new vertex w we add vw to the *possible path*. If w is a big vertex we mark w as found. Then we join the possible path to the found path. During the backtracking step starting from x, we distinguish if the current edge $e = xy$ is on the possible path or in the found path. If e is on the possible path, we backtrack as known from a DFS and delete e from the possible path. When backtracking on the found path y is not a big vertex, we immediately backtrack further, even if y still has unexplored edges left. If y is a big vertex we do the backtracking known from DFS. When the algorithm terminates and all big vertices are found, then we have the found path contains all big vertices and we take P as the found path.

Before we go further, we show that the mDFS finds the path $\tilde{P}$ containing all big vertices if it exists. We show that by induction. If at most one big vertex exist, then the algorithm finds a path $\tilde{P}$ containing all big vertices, since the mDFS starts at a big vertex v. So assume that there are at least two big vertices. Since T is connected, the mDFS finds a next big vertex w, since until this point we only use the normal DFS. When backtracking, the algorithm does not look for further paths that start at vertices which are already on $\tilde{P}$. We have that if there is a vertex u that connects three different subgraphs containing at least one big vertex each, there is no path that contains all of these big vertices. Thus, the algorithm ignores all of these possible paths starting between two already found big vertices. Because $\tilde{P}$ is unique, it does not matter at which vertex the algorithm starts. Now we show that the alternation of the algorithm do not change the running time of the DFS. Saving or deleting an edge when exploring or backtracking is possible in $O(1)$. Adding the possible path to the found path is also possible in $O(1)$, since we only can add those paths to the start or the end of the already found path, since we use lists. Marking big vertex as found and checking whether a vertex is a big vertex by using an array are both also possible in $O(n)$. Thus this algorithm still needs $O(n + m) = O(n)$, for trees.

Next, we search for subgraphs $\mathcal{S}$ of T which start at the end points of $\tilde{P}$ and not containing any vertices of $\tilde{P}$. For any trees in $\mathcal{S}$, we determine whether they contain any vertices of degree 3. We do that, by using a Depth first search (DFS) starting at the respective endpoint v and save if we find a vertex of degree 3. When backtracking to v, we save for the subtree S whether S contains a vertex of degree 3. Note that the running time is $O(n)$. The selection of the needed subtree S based on the degree of v and the degree of the neighbors of v is possible in $O(n)$. We need to do the DFS at most eight times, but since eight is a constant, it remains in $O(n)$.

The auxiliary graph G is needed, since we only have vertices of degree 3 on the bottom layer, if they are adjacent to a vertex of degree at least 3 on the top layer. For building G, we use a DFS. We modify the DFS and call it $G - DFS$, such that we build the G during exploring T. During the $G - DFS$ if we explore a new vertex of degree 3 we save it as *current vertex*. Every time we explore a new vertex w of degree 3 or 1, we add w to G by adding an edge between w and the current vertex. Then we set w as the current vertex. If we backtrack to a vertex x of degree 3, we set x as current vertex. This modification provides G, because $G - DFS$ skips vertices of degree 2 when building G. Since we already saved the degrees of the vertices one can add a vertex and an edge in $O(1)$. So, $G - DFS$ needs $O(n)$ and is used at most once for each side of $\tilde{P}$.

Next, we want to find the longest path P_G in the auxiliary graph starting in v. Since G is a tree, we use a breadth-first search (BFS) and save the distances for each vertex and the parent vertex, so find P_G. The running time for BFS is in $O(n + m) = O(n)$ and the modification are in $O(1)$, so the total running time for the BFS is still in $O(n)$.

We look the last three vertices and their neighbors on P_G and compare their degrees. These comparisons are in $O(1)$ Since we already know that the degree is at most three, we have that choosing the vertex is in $O(1)$. At last, we again search for the longest path starting in the respective vertex. Since all remaining vertices are at most of degree 2, we just follow the paths and count the vertices, so we have that this is also in $O(n)$.

We showed, that each operation for constructing P is in $O(n)$. Since every of these operations happens sequentially and only for a bounded number of times, we have that whole construction is in $O(n)$. ∎

With this construction of a path, we replace the loop `FindCrossedLadderEmbedding` by the given construction of P.

Theorem 5.6: *`FindCrossedLadderEmbedding` modified with the construction of paths P as described above instead of creating all possible paths needs linear running time and also provides an embedding of a tree T in L_2 if one exists.*

Proof. Note that the modified algorithm still provides an embedding of a tree T in L_2 if one exists by Lemma 5.4.

Now, we look at the running time of the modified algorithm. First we determine the degree of each vertex. Since the construction of P also determines the degrees of each vertex, we just combine that and terminate the algorithm if there is a vertex with degree greater than 5. So the determination of the maximum and constructing the paths only need $O(n)$ time by Lemma 5.5. Note that we may have up to four different versions of P. So we have to do the line sweep algorithm at most four times. Since this is a constant number of times, we only need to show that the line sweep algorithm itself is in $O(n)$.

We do not change the line sweep algorithm but we have a closer look on the line sweep part. We determine how many times each vertex is used in the line sweep part. We have that every vertex on P is only used when the line sweep algorithm iterates to it. For every vertex w of T and not on P we find exactly one vertex v of P that is the root of a subtree S containing w. From this follows that w is only investigated in the line sweep step of v. Now we determine how many times w is used during the line sweep step of v. First, one determines the size of S. To do that, w is used once. Then we embed w in the crossed ladder, where w is used a second time. Thus we have that each vertex not in P is used two times during the line sweep algorithm and each vertex in P only once. From this we have that the whole line sweep part is already in $O(n)$.

With that, we have that finding an embedding of a tree in the crossed ladder is indeed in $O(n)$. ∎

In this subsection we showed that finding an embedding of a tree in the crossed ladder is possible in linear time, when using the correct data structures.

5.2 Embedding a Graph in the k-Wide Crossed Ladder

In the following we show that for a finite graph G and a constant k one can decide in polynomial time if there exists an embedding of G in a k-wide crossed ladder L_k.

To solve that problem, we look at the embedding φ of a graph G and find piecewise embeddings $\psi_1, \dots, \psi_j$ by looking at columns of L_k. We shall see, that $\psi_1, \dots, \psi_j$ can be combined again to φ. With this observation we show that there is an embedding of G by using piecewise embeddings $\psi_1, \dots, \psi_j$ in the algorithm that decides whether G is embeddable in L_k.

In Figure 5.3 we give an example, where G is embedded and we find parts $H_1, \dots, H_j$ of $\varphi(G)$ of size $P_3 \boxtimes P_k$, such that each column of $\varphi(G)$ corresponds to the center column of one of $H_1, \dots, H_j$. We can retrieve φ of G by putting $H_1, \dots, H_j$ together, such that H_i and H_{i+1} overlap by two columns.

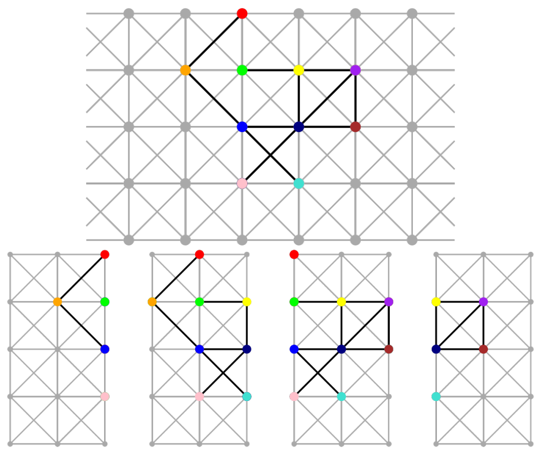


Figure 5.3: The graph G in the upper image is embedded in a k-wide crossed ladder. In the lower images a sequence of configurations is shown, which embed G. The first one is a start configuration, the last one is an end configuration. Each configuration between them, including the end configuration, is a following configuration. The vertices are color coded in the graph and the configurations.

Let $|V(G)| = n$. We have at most n^{3k} different embeddings of induced subgraphs $G_1, \dots, G_j$ of G, because each vertex of G can only be embedded on $3k$ different positions in $P_3 \boxtimes P_k$. The edges within each G_i are given by G since G_i is an induced subgraph. We use $G_1, \dots, G_j$, to define an algorithm, which generates such $H_1, \dots, H_j$ and fits them together. Since the algorithm only generates each time only a number of configurations polynomial in n, we show that this algorithm decides the problem whether G is embeddable in L_k in polynomial time.

In the following we call $H_1, \dots, H_j$ configurations, which are defined as follows: Let $V(P_3) = \{1, 2, 3\}$ be the vertex set of P_3 and $K = P_3 \boxtimes P_k$ a graph. Then, we have three columns of K, which we enumerate such that the i^{th} column K_i contains the vertex component i of P_3.

Let $X \subseteq V(G)$, which we will specify later and let $c : X \longrightarrow V(K)$ be a map. We define X_i as a set such that $c(X_i) \subseteq V(K_i)$. We also define the restriction $c^i : X_i \longrightarrow [k]$ with $c^i(v) = u$ for $c(v) = \langle i, u \rangle$, that maps vertices of G to their vertex component of P_k if c maps them to the i^{th} column of K. Let G be a graph. We say c is a *configuration*, if the following two criteria are fulfilled. First, the map c is an embedding of G' for G' being the induced subgraph of G on the vertex set $X \neq \emptyset$. Second, the whole 1-hop-neighborhood of X_2 in G is embedded by c i.e. $N_1^G(X_2) \subseteq X$.

For combining two configurations c_A, c_B, the last two columns of c_A have to match the first two columns of c_B. More formally, let $c_A : A \longrightarrow V(K)$, $c_B : B \longrightarrow V(K)$ be two configurations. Then we say c_B is a *following configuration* of c_A if $A_2 = B_1$ and for all$x \in A_2$ we have $c_A^2(x) = c_B^1(x)$ and $A_3 = B_2$ and for all $x \in A_3$ it holds that $c_A^3(x) = c_B^2(x)$.

A configuration is called *start configuration* if no vertices are embedded in the first column and a configuration is called *end configuration* if no vertices embedded in the third column.

For the following algorithm, we consider sequences of configurations. Let $(c_i)_{i \in [r]}$, $r \in \mathbb{N}$ be a sequence of configurations, where c_{i+1} is a following configuration of c_i for all $i \in [r-1]$. We say a sequence is *closed*, if the first configuration is a start configuration and the last configuration c_r is an end configuration.

Lemma 5.7: *Let G be a graph. Let φ an embedding of G in L_k. Then there is a closed sequence of configurations $(c_i)_{i\in[r]}$, $r \in \mathbb{N}$, where each configuration has one column of $\varphi(G)$ as middle column.*

Proof. To see this, we enumerate the columns from left to right in $\varphi(G)$. We construct a start configuration c_1 by restricting φ to the first column containing the leftmost vertex of $\varphi(G)$, the column on its left and the second column. For constructing the i^{th} configuration c_i, we restrict φ to its $i-1^{th}, i^{th}, i+1^{th}$ column. We have an end configuration, since the $(r+1)^{th}$ column does not contain any vertices but is the third column of c_r. ∎

Lemma 5.7 shows, that we can construct a closed sequence of configurations from an embedding of a graph G. We also want to construct an embedding of a graph G from a sequence of configurations. Then finding an embedding of G is equivalent to finding a closed sequence of configurations for G. In general it is not possible to retrieve an embedding from a (closed) sequence of configurations, because it could be the case that same vertices of G, are in multiple, non consecutive configurations. We now want to construct a map and a subgraph of L_k that fulfills this idea of an embedding. As we identify P_∞ with $\mathbb{Z}$, we say that L_k^i is the column of L_k, where every vertex contains the vertex component $i \in V(P_\infty)$. Having a closed sequence of configurations $(c_i)_{i\in[r]}, r \in \mathbb{N}$ we identify L_k^i with the middle column of the i^{th} configuration by a function t with $t(c_i) = L_k^i$. For a closed sequence of configurations $(c_i)_{i\in[r]}$, $r \in \mathbb{N}$ and a graph G, let

$$X = \{v \mid v \in V(L_k), \exists w \in V(G), \exists i \in [r] : t(c_i(w)) = v\}$$

be the set of vertices in L_k where a vertex of G is mapped to by any configuration in the configuration sequence. We define the *induced mapping* of a closed sequence $(c_i)_{i\in[r]}$ by $\rho : X \longrightarrow V(G)$ with $\rho(v) = w$. We have that w is the vertex in G that is mapped to v by any configuration. We define the *induced graph* $I_\rho = (X, E) \subseteq L_k$ where $ab \in E$ if and only if there is a configuration in the configuration sequence containing $\rho(a)\rho(b) \in E(G)$.

Let G be a graph. Then we state that we there is a closed sequence of configurations if and only if G is embeddable in L_k. To prove that we first have a closer look at closed sequences and therefore I_ρ. We show that there is a connected component C in I_ρ such every vertex of G is mapped to exactly one vertex of C.

We start by showing that every vertex of G is mapped to at least one vertex of I_ρ.

Lemma 5.8: *Let G be a connected graph and $k \in \mathbb{N}$. Let $(c_i)_{i\in[t]}$, $t \in \mathbb{N}$ be a closed sequence of configurations with an induced mapping ρ. Then there is a connected component C in I_ρ such that ρ is surjective on C. That means that for every vertex v of G there is at least one vertex w in C such that $\rho(w) = v$.*

Proof. Let $(c_i)_{i\in[t]}, t \in \mathbb{N}$ be a closed sequence of configurations. Let ρ the induced mapping and I_ρ the induced graph. Since every configuration embeds at least one vertex on the center column, we have that I_ρ contains at least one connected component. Chose the connected component C, such that there is a vertex $r \in V(I_\rho)$ with r being in the end configuration c_t. Let $r_G = \rho(r)$. By definition of the configuration we have that the whole 1-hop neighborhood in G of any vertex $v \in V(G)$ is also embedded in a configuration. By definition of the induced mapping we also have every vertex of G is mapped to L_k and that for every two vertices x, y that are adjacent in G their mappings $\rho(x), \rho(y)$ are adjacent. Since G is connected, we find a $r_G v$-path for every $v \in V(G)$, if v is embedded in a center column. Since we already know that $r_G = \rho(r)$, we find an rv'-path in C, such that $v = \rho(v')$.

Assume for the sake of contradiction that there is a vertex $w \in V(G)$ such that there is no vertex in $V(C)$ that is mapped to w by ρ. We still have an $r_G w$-path P in G. We find a vertex w'_G on P, such that there is still a vertex w' with $\rho(w') = w'_G$ but there is no mapping for the following vertex of w'_G on P. This is the case if w'_G is in the first or third column of a configuration. If w'_G is on the third column of a configuration, then there is a following configuration, where w' is on the center column. By definition of a configuration the whole 1-hop-neighborhood in G of w' is embedded in L_k, so we have a contradiction. If w'_G is on the first column of a configuration c_M, then we find a configuration, for which c_M is a following configuration. Thus w' is again in the center column of a configuration. This is again a contradiction.

We have that for every vertex v of G there is a vertex w of L_k for which $\rho(w) = v$. Thus ρ is surjective on $V(C)$. ∎

It remains to show that ρ is injective when restricting I_ρ to one connected component. Before proving that, we show a more general statement. That statement claims that if we have a cycle C to embed into L_k, the induced graph I_ρ does not embed C multiple times in a connected component. That means that if we have a cycle C and the connected component Z, if we follow C and the respective vertices in Z, induced by ρ we run multiple times through C i.e. We find multiple sequences of vertices of Z, such that ρ maps them to the sequence of vertices of C.

Lemma 5.9: *Let $C = v_0 v_1 \dots v_r v_0$ be a cycle of length $r+1 \in \mathbb{N}$. Then there is no closed sequence of configurations with induced mapping ρ such that I_ρ contains a connected component Z where C is multiple times embedded i.e. there are multiple sequences $s_1, \dots, s_j$ of vertices of Z of length $r+1$ such that $\rho(s_1) = C$. The sequences $s_1, \dots, s_j$ may only overlap in their first and last vertex respectively.*

Proof. Note that a cycle of arbitrary length is embeddable in L_k, for $k \geq 2$. We denote the vertices of C by $v_0 v_1 \dots v_r v_0$ and the vertices of Z by u.

Assume that there is a shortest closed sequence of configurations $(c_i)_{i \in [r]}, r \in \mathbb{N}$ such that the induced graph I_ρ has a connected component Z where C is embedded multiple times as described above, i.e. $Z = z_0, \dots, z_\ell, z_0$ with $\rho(z_j) = v_{(j \mod k)}$ for all $j \in [\ell]$, $\ell \in \mathbb{N}$.

First, we show that Z is again a cycle. Assume without loss of generality, that v_0 is embedded by c_1. By definition of configuration we find an $u_0 u_r$-path P in Z such that $\rho(u_i) = v_i$ for all $u_i \in V(P) \subseteq V(Z)$ and $i \in [0, r]$. We have that u_r is adjacent to vertex $\tilde{u}_0$, with $\rho(\tilde{u}_0) = v_0$. Then we have $u_0 \neq \tilde{u}_0$ since we assume that Z contains multiple copies of C. Note that for every vertex except u_0 and $\tilde{u}_0$ the whole 1-hop-neighborhood in Z is already covered, thus by considering further vertices, the vertices are connected to u_0 or the second to last considered vertex. Since the sequence of configurations is closed by assumption, we have that for every vertex in C the whole neighborhood in Z is covered. From this follows that u_0 also is connected to a vertex $\tilde{u}_r$ with $\rho(\tilde{u}_r) = v_r$. Thus, after some finite number of copies of C, there is a vertex $\tilde{u}_r$ with $\rho(\tilde{u}_r) = v_r$ that is adjacent u_0 and thus closes the cycle.

In the following we show that there is no closed configuration sequence that contains one vertex of C twice. Assume for the sake of contradiction that there is such a closed sequence of configurations. Let ℓ be the leftmost vertex and r be the rightmost vertex of Z respectively. Since Z is a cycle, we have a ℓr-paths P_a and a $r\ell$-path P_b. Assume that that P_a is at most as long as P_b.

First assume that P_a does not contain a copy of C i.e. the image of $V(P_a)$ under ρ is a real subset of $V(C)$. Since we have that the path P_aP_b contains at least two copies of C, there is a subpath P_S of P_b such that $\rho(u^a) = \rho(u^S)$ for $u^a \in V(P_a)$ and $u^S \in V(P_S)$. In the following we use the notation of $u^a \in V(P_a)$ and $u^S \in V(P_S)$. Let j be the length of P_a. We have that u_0^a is on the left side of u_0^S. Otherwise either $u_0^a = \ell$ is not the leftmost vertex of C or there are two vertices in the same column in Z that ρ maps to the same vertex of C. We also have that $u_j^a = r$ is on the right side of u_j^S because either r is not the rightmost vertex or there are two vertices in a column that ρ maps to the same vertex of C. When iterating through P_1 and P_S, starting in ℓ or u_a^S respectively, we find vertices v_i^a, v_i^S such that $\rho(v_i^a) = \rho(v_i^S)$ and v_i^a is on the left side of v_i^S but v_{i+1}^a is not on the left side of v_{i+1}^S. Thus v_i^a and v_i^S are in the same column or in consecutive columns of Z. From this follows that there is a configuration that contains v_i^a and v_i^S, which is not possible by definition of a configuration.

In the other case assume that both P_a and P_b contain at least one full copy of C.

So let $x_0 \dots x_r$ be a copy of C in P_a and $w_0 \dots w_r$ be a copy of C in P_b. i.e. $\rho(x_i) = \rho(w_i) = v_i$ for all $i \in [0, r]$. By assumption we have that x_0 is the leftmost vertex in Z. Then we have that w_0 is at least two columns to the right of x_0, otherwise either the configuration containing v_0 on the middle column is not a valid configuration or v_0 is not embedded as leftmost as possible. We also have, that w_r has to be in the same column as x_0 or on the column one to the right, because we have by construction that the edge x_0w_r exists. From this follows that x_r is not in the first two columns. So we have that x_r is on the right of w_r and x_0 is on the left of w_0. By iterating through P_a and P_b, we find an index i such that x_i and w_i are in consecutive columns or the same column of Z. This is the case for the first i for which w_i is not anymore on the right side of x_i. So we find two embeddings of the same vertex v_i within a configuration. This is a contradiction.

Thus, we see that there are not possible to embed multiple copies of a C within one connected component. ∎

With this we show that ρ is injective one a connected component of I_ρ.

Lemma 5.10: *Let G be a graph, $k \in \mathbb{N}$. Let $(c_i)_{i\in[r]}$ $r \in \mathbb{N}$ a closed sequence of configurations with the induced mapping ρ. Then there is a connected component Z in I_ρ such that ρ is injective on Z, i.e. for every vertex v of G there is at most one vertex w in Z such that $\rho(w) = v$.*

Proof. Let $(c_i)_{i\in[r]}$ and $r \in \mathbb{N}$ be a shortest closed sequence of configurations. Let ρ be an induced mapping and I_ρ the induced graph. We take the connected component Z that contains a vertex, that is embedded by c_r.

Now, we show that ρ is injective on Z. Assume that for the sake of contradiction, ρ is not injective. So there are two vertices $v, u \in V(Z)$ such that $\rho(u) = \rho(v) = z \in V(G)$. Then there is a uv-path P in Z. Let w a vertex on P then we have an uw-path $P_u = w = u_1u_2u_2 \dots u_j = u$ and the wv-path $P_v = w = v_1v_2 \dots v_\ell = v$.

Assume for contradiction that $\rho(w)$ is not in a cycle in G. Then we have that $\rho(u_2) = \rho(v_2)$, since a path in a graph without a cycle is unique. It is not possible that both u_2 and v_2 are mapped to the same vertex of Z because then there is a configuration where a vertex of G is embedded twice.

So, assume that $\rho(w)$ is on a cycle C_w and that $\rho(u_2) \neq \rho(v_2)$. Let z' be first vertex in $V(G)$ such that z' is on the same cycle as $\rho(w)$ and on every $z\rho(w)$-path and on P. We have that z' always exists, because either it equals z or we find a vertex on C_w that connects the remaining vertices with a bridge. We have that there are two vertices u_i and v_j in P_u and P_v respectively such that $\rho(u_i) = \rho(v_i) = z'$. By construction we have that there is a cycle in C_w containing two vertices that map to z'. With Lemma 5.9 this is not possible.

Thus a vertex does not appear multiple times in a connect component. ∎

With these statement we show that it is sufficient to find a closed sequence of configuration for embedding a graph in L_k.

Theorem 5.11: *Let G be a connected graph and $k \in \mathbb{N}$. Then G is embeddable in L_k if and only if there is a closed sequence of configurations.*

Proof. First, consider an embedding φ of G in L_k. Without loss of generality we assume that the leftmost vertex is embedded in L_k^1 and the rightmost vertex in L_k^r for some $r \in \mathbb{N}$ (otherwise we just translate φ). Then, we interpret $\varphi(G)$ as induced graph I_ρ for $\rho = \varphi^{-1}$. By Lemma 5.7 we have a closed sequence of configurations $(c_i)_{i\in[t]}$, $t \in \mathbb{N}$.

Now consider a shortest sequence of configurations $(c_i)_{i\in[r]}$. Let C be a connected component of I_ρ. Since we have a shortest sequence of configurations we also have that there is a vertex of C in both the first and the last column of I_ρ. Otherwise we find a shorter sequence of configurations that only contains C. So we assume that I_ρ is connected and equals C. With Lemma 5.8 and Lemma 5.10 we have that I_ρ contains every vertex of G exactly once. By definition of ρ we have that each column used in I_ρ can be identified with a center column of a configuration in $(c_i)_{i\in[r]}$. By definition of a configuration we have that exactly those edges that are in $E(G)$ are mapped to I_ρ. Thus we have I_ρ is a copy of G and since I_ρ is embedded in L_k, G has an embedding in L_k. ∎

With this theorem we build an algorithm `EmbeddingInKWideCrossedLadder` which checks whether there is a closed sequence of configurations for a given graph G. After we present the algorithm, we show that it has polynomial running time.

The algorithm starts by generating all possible start configurations and stores them in a set S. For each configuration in S, `EmbeddingInKWideCrossedLadder` finds all possible following configurations and saves them in a set S' if the same configuration is not already in S'. If one of the following configurations is an end configuration the algorithm terminates and states that there exists an embedding of G in L_k. When every following configuration of every configuration of S is found, without `EmbeddingInKWideCrossedLadder` terminating, we have S' as new set S. The algorithm starts again finding following configurations for every configuration of S. If `EmbeddingInKWideCrossedLadder` does not find an end configuration within $n = V(G)$ iterations, `EmbeddingInKWideCrossedLadder` terminates and claims that no embedding of G in L_k exists.

In the following lemma we show the correctness of `EmbeddingInKWideCrossedLadder`.

Lemma 5.12: *The algorithm `EmbeddingInKWideCrossedLadder` described above decides whether a connected graph G is embeddable in the k-wide crossed ladder.*

Proof. The algorithm only states that G is embeddable in L_k if it finds an end configuration c_E. Since the algorithm begins by creating start configurations and after that only following configurations, we have that there is a closed sequence of configurations, ending in c_E. With Theorem 5.11 we have that G is embeddable in L_k.

There are two cases when `EmbeddingInKWideCrossedLadder` states that G is not embeddable in L_k. First, assume that the algorithm cannot create another following configuration. Assume for the sake of contradiction that G is embeddable in L_k. Then, by Theorem 5.11 it exists a closed sequence of configurations $(c_i)_{i\in[r]}, r \in \mathbb{N}$. Since the algorithm starts with all possible start configurations, c_1 is created. Since the algorithm creates all possible following configurations all $(c_i)_{i\in[r]}$ are created. texttt EmbeddingInKWideCrossedLadder terminates before creating c_r the closed sequence does exist. This contradicts the assumption that G is embeddable.

The second case claims, that if after $n = |V(G)|$ iterations there is no end configuration, there will be no end configuration following. Note that an embedding of G contains at most n vertices. Since G is connected, each configuration of a closed configuration sequence contains at least one vertex of G. So there is a closed sequence of configurations that contains at most n configurations. Since the algorithm creates all possible start configurations and in each step all possible following configurations it finds an end configuration after at most n steps if there is one. Otherwise there is no end configuration and thus no closed sequence. By Theorem 5.11 there is no embedding of G. ■

At last we show that the algorithm terminates and it decides in polynomial time whether a graph is embeddable in L_k. Note that the algorithm may have a better running time than estimated in the proof but a tighter upper bound is not necessary to show polynomial running time.

Theorem 5.13: *Let G be a graph and $k \in \mathbb{N}$. Deciding whether $G \subseteq P_\infty \boxtimes P_k$ is possible in polynomial time.*

Proof. To decide whether a graph G is embeddable in the k-wide crossed ladder, we use the algorithm described above. With Lemma 5.12 we have that the algorithm decides the statement.

Now we show that `EmbeddingInKWideCrossedLadder` always terminates. Since we only look at finite graphs we have $|V(G)| < \infty$. Since we do not change the graph within the loop of the algorithm, it only makes finite iterations. The loop body also terminates in finite time, since a configuration is a combination of vertices of G. The algorithm is deterministic and thus it always terminates.

Next, we show that the algorithm has polynomial running time. To estimate the number of possible configurations we determine the total number of possibilities $\mathcal{N}$ to place pairwise different vertices of G in $P_3 \boxtimes P_k$. We do not have a closer look at the edges, since they are given by the vertices. Note that $\mathcal{N}$ is bigger than the number of configurations, since we also count "configurations", where vertices on the center column lack neighbors from G. So an upper bound for the number of configurations is $\mathcal{N} = n^{3k}$.

First, we create each possible configuration. To do this, we pick an arbitrary set of vertices, embed them in the center column and take the 1-hop-neighborhood to embed them around it. At least we again pick arbitrary vertices to embed them in the first and last column. This is possible in $O(n)$, because we assume that finding a neighbor of a vertex is possible in $O(1)$. Thus generating all configurations is possible in $O(\mathcal{N}n) = O(n^{3k+1})$. We find all possible start configurations in $O(\mathcal{N}k) = O(\mathcal{N})$, since we only look at each configuration and determine whether it is a start configuration.

We have that there are at most n iterations. In each iteration we find all following configurations for each found configuration. Finding all possible following configurations for one configuration is possible in $O(\mathcal{N}2k)$, since we look at each possible configurations and compare the overlapping columns. For adding a found configuration we need at most $O(\mathcal{N})$, since we have to make sure that each configuration appears at most once in the set. Thus, finding all following configurations and save them is possible in $O(\mathcal{N}^2 2k)$. Since we have to find all following configurations for all configurations, we have a running time for one iteration of $O(\mathcal{N}^3 2k)$. From this follows that we have a total running time of $O(\mathcal{N}) + O(\mathcal{N}n) + nO(\mathcal{N}^3 2k) = O(n\mathcal{N} + n\mathcal{N}^3) = O(n(n^{3k}) + n(n^{3k})^3) = O(n^{9k+1})$ Therefore the algorithm has polynomial running time, since k is given. ■

It is possible to also check whether graphs that are not connected are embeddable in L_k by checking if every connected component is embeddable separately.

6 Conclusion

In this thesis we saw that the complexity of deciding whether a tree or a graph is embeddable into a grid is closely related to definition of the graph to embed and the graph to embed into.

We showed that deciding whether a tree is embeddable into the ω-rectangle grid and the ω-full grid is NP-complete. This suggests, that it is not relevant if we use the cartesian or the strong product to for creating the grid. We also see that the multiple layers do not enlarge the product graph enough, such that the decision problems becomes easier. For further research it might be interesting to investigate if one can find a smallest clique size depending on the tree, such that the tree is embeddable. We assume that one may find an approximation to this clique size and that this size depends somehow on the amount and constellation of inner vertices of the tree, where multiple branches start with length at least 2. If we have none of such vertices, we have a caterpillar. We showed that deciding whether a caterpillar is embeddable into both the ω-rectangle grid and the ω-full grid is possible in linear time.

As another case in this work, we restricted the product graph to the k-wide crossed ladder. We investigated that deciding whether a graph is embeddable into the k-wide crossed ladder is possible in polynomial time. In this case one can optimize the algorithm but we assume that the problem will still be in $O(n^{O(k)})$, because if the running time does not depend on k anymore that would contradict the fact that the decision problem of deciding whether a tree is embeddable in F_1 is NP-complete. We also showed that finding an embedding of a graph in the 2-wide crossed ladder if one exists is possible in linear time. So for further research it might be interesting if there is an constructive algorithm that also provides embeddings for $k > 2$.

Bibliography

[AKS91] Alok Aggarwal, Maria Klawe, and Peter Shor. "Multilayer Grid Embeddings for VLSI". In: *Algorithmica* Volume 6 (June 1991), pp. 129–151. DOI: *10.1007/bf01759038*.

[BEU23] Therese Biedl, David Eppstein, and Torsten Ueckerdt. *On the complexity of embedding in graph products*. 2023. arXiv: *2303.17028*.

[BLSS04] WOLFGANG W. BEIN, LAWRENCE L. LARMORE, CHARLES O. SHIELDS, and I. HAL SUDBOROUGH. "EMBEDDING A COMPLETE BINARY TREE INTO A THREE-DIMENSIONAL GRID". In: *Journal of Interconnection Networks* Volume 05 (2004), pp. 111–130. eprint: *https://doi.org/10.1142/S0219265904001052*.

[CGL17] Yijia Chen, Martin Grohe, and Bingkai Lin. *The Hardness of Embedding Grids and Walls*. 2017. arXiv: *1703.06423*.

[Dvo+21] Zdeněk Dvořák, Tony Huynh, Gwenael Joret, Chun-Hung Liu, and David R. Wood. "Notes on Graph Product Structure Theory". In: *2019-20 MATRIX Annals*. Springer International Publishing, 2021, pp. 513–533. ISBN: 9783030624972. DOI: *10.1007/978-3-030-62497-2_32*.

[FW91] Michael Formann and Frank Wagner. "The VLSI layout problem in various embedding models". In: *Graph-Theoretic Concepts in Computer Science*. Edited by Rolf H. Möhring. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 130–139. ISBN: 978-3-540-46310-8.

[Gre89] Angelo Gregori. "Unit-length embedding of binary trees on a square grid". In: *Information Processing Letters* Volume 31 (1989), pp. 167–173. ISSN: 0020-0190. DOI: *https://doi.org/10.1016/0020-0190(89)90118-X*.

[HW21] Robert Hickingbotham and David R. Wood. *Structural Properties of Graph Products*. 2021. arXiv: *2110.00721*.

[HW25] Kevin Hendrey and David R. Wood. *Short Paths in the Planar Graph Product Structure Theorem*. 2025. arXiv: *2502.01927*.

[Iva16] D. Ivanova. "On the Distortion of Embedding Perfect Binary Trees into Low-dimensional Euclidean Spaces". In: 2016.

[KV84] MR Kramer and J Van Leeuwen. "THE COMPLEXITY (OR WIRE-ROUTING AND RINDING MINIMUM AREA LAYOUTS FOR ARBITRARY VLSI CIRCUITS". In: (1984).

[MT92] Jiři Matoušek and Robin Thomas. "On the complexity of finding iso- and other morphisms for partial k-trees". In: *Discrete Mathematics* Volume 108 (1992), pp. 343–364. ISSN: 0012-365X. DOI: *https://doi.org/10.1016/0012-365X(92)90687-B*.

[SV10] Benny Sudakov and Jan Vondrák. "A randomized embedding algorithm for trees". In: *Combinatorica* Volume 30 (July 2010), pp. 445–470. DOI: *10.1007/s00493-010-2422-5*.

[Tam87] Roberto Tamassia. “On Embedding a Graph in the Grid with the Minimum Number of Bends”. In: *SIAM Journal on Computing* Volume 16 (1987), pp. 421–444. eprint: *https://doi.org/10.1137/0216030*.

[Tik16] Mikhail Tikhomirov. “On computational complexity of length embeddability of graphs”. In: *Discrete Mathematics* Volume 339 (2016), pp. 2605–2612. ISSN: 0012-365X. DOI: *https://doi.org/10.1016/j.disc.2016.05.011*.

[TT24] Guilherme Simon Torres and Vilmar Trevisan. “Brouwer’s Conjecture for the Cartesian product of graphs”. In: *Linear Algebra and its Applications* Volume 685 (2024), pp. 66–76. ISSN: 0024-3795. DOI: *https://doi.org/10.1016/j.laa.2023.12.019*.

[UWY22] Torsten Ueckerdt, David Wood, and Wendy Yi. “An improved planar graph product structure theorem”. In: *The Electronic Journal of Combinatorics* Volume 29 (June 2022). DOI: *10.37236/10614*.