PACKET ROUTING ON THE GRID

BRITTA PEIS, MARTIN SKUTELLA, AND ANDREAS WIESE

Abstract. The packet routing problem, i.e., the problem to send a given set of unit-size packets through a network on time, belongs to one of the most fundamental routing problems with important practical applications, e.g., in traffic routing, parallel computing, and the design of communication protocols. The problem involves critical routing and scheduling decisions. One has to determine a suitable (short) origin-destination path for each packet and resolve occurring conflicts between packets whose paths have an edge in common. The overall aim is to find a path for each packet and a routing schedule with minimum makespan.

A significant topology for practical applications are grid graphs. In this paper, we therefore investigate the packet routing problem under the restriction that the underlying graph is a grid. We establish approximation algorithms and complexity results for the general problem on grids, and under various constraints on the start and destination vertices or on the paths of the packets.

1. Introduction

In this paper, we study the packet routing problem on grid graphs. In an instance of this problem we are given a set of unit-size packets with specified source and destination vertices. First, we need to define a path for each packet along which we want to route it. Then we need to find a routing schedule to transfer the packets through the network. This is not trivial since each link in the network can be used by at most one packet at a time. The overall goal is to find a path assignment and routing schedule that minimizes the makespan, i.e., the time when the last packet has reached its destination. We study also the special case that the paths of the packets are already given by some other source and we need to find only the routing schedule.

The packet routing problem has several applications in practice, e.g., in parallel computing or in cell structured networks. In those settings, packets of information need to be transferred through the network. In order for the network to operate efficiently, it is needed that the packets reach their respective destinations as quickly as possible. Therefore, the paths of the packets and the routing schedule need to be computed such that the packets encounter as few delay as possible. One of the most common natural topologies of the routing problem in practical applications are grid graphs, e.g. in parallel computing. Therefore, we take this special structure into account when looking for efficient solution methods.

1.1. Packet Routing Problem.

The packet routing problem is defined as follows: Let $G = (V, E)$ be an undirected graph (in our case this will usually be a grid graph). A packet $M_i = (s_i, t_i)$ is a tuple consisting of a start vertex $s_i \in V$ and a destination vertex $t_i \in V$. Let $\mathcal{M} = \{M_1, M_2, M_3, ..., M_{|\mathcal{M}|}\}$ be a set of

packets. Then $(G, \mathcal{M})$ is an instance of the *packet routing problem with variable paths*. The problem has two parts: First, for each packet M_i we need to find a path $P_i = (s_i = v_0, v_1, ...v_{\ell-1}, v_\ell = t_i)$ from s_i to t_i such that $\{v_i, v_{i+1}\} \in E$ for all i with $0 \leq i \leq \ell - 1$. Assuming that it takes one timestep to send a packet along an edge we need to find a routing schedule for the packets such that

- each message M_i follows its path P_i from s_i to t_i and
- each edge is used by at most one packet at a time.

We assume that time is discrete and that all packets take their steps simultaneously. The objective is to minimize the makespan, i.e., the time when the last packet has reached its destination vertex. For each packet M_i we define $\bar{D}_i$ to be the length of the shortest path from s_i to t_i, assuming that all edges have unit length. Moreover, the *minimal dilation* $\bar{D}$ is defined by $\bar{D} := \max_i \bar{D}_i$. It holds that $\bar{D}$ is a lower bound for the length of an optimal schedule.

Since there are algorithms known to determine paths for routing the packets (see [31, 6, 18] or simply take shortest paths) we will also consider the *packet routing problem with fixed paths.* An instance of this problem is a tuple $(G, \mathcal{M}, \mathcal{P})$ such that G is a (grid) graph, $\mathcal{M}$ is a set of packets and $\mathcal{P}$ is a set of predefined paths, one for each packet. Since the paths of the packets are given in advance they do not need to be computed here. The aim is to find a schedule with the properties described above such that the makespan is minimized. For each packet M_i we define D_i to be the length of the path P_i, again assuming that all edges have unit length. Like above we define the *dilation* D by $D := \max_i D_i$. For each edge e we define C_e to be the number of paths that use e. Then we define the *congestion* C by $C := \max_e C_e$. It holds that C and D are lower bounds for the length of an optimal schedule.

We distinguish between grid graphs in which two packets are allowed to use an edge in opposite directions at the same time, or not. The infinite grid graph $G_\# = (V_\#, E_\#)$ is the undirected graph consisting of the vertices $V_\# = \{v_{i,j} | i, j \in \mathbb{Z}\}$ and the edges $E_\# = \{\{v_{i,j}, v_{i',j'}\} \,|\, |i - i'| + |j - j'| = 1\}$. The directed graph $\overleftrightarrow{G}_\# = \left(V_\#, \overleftrightarrow{E}_\#\right)$ is the bidirected infinite grid graph with $\overleftrightarrow{E}_\# = \{(u, v), (v, u) \,|\, \{u, v\} \in E_\#\}$. We will consider infinite grid graphs rather than finite grids because we want the borders of a finite grid not to have any impact on the problem.

Throughout the paper we will use the notation $|S|$ for the length of a schedule S. For a packet routing instance I with fixed or variable paths let $OPT(I)$ denote a schedule with minimum makespan. For an algorithm $\mathcal{A}$ for the packet routing problem denote by $\mathcal{A}(I)$ the schedule computed by $\mathcal{A}$ for the instance I. The algorithm $\mathcal{A}$ is an α-approximation algorithm if it runs in polynomial time and for all instances I it holds that $|\mathcal{A}(I)| \leq \alpha \cdot |OPT(I)|$. We call α the *approximation ratio* or *performance ratio* of $\mathcal{A}$.

1.2. **Related Work.** Packet routing and related problems are widely studied in the literature. Di Ianni show that the delay routing problem [8] is NP-hard. The proof implies that the packet routing problem on general graphs is NP-hard as well. Leung et al. [22, chapter 37] study packet routing on different graph classes. In [5] Busch et al. study the direct routing problem, that is the problem of finding a routing schedule such that a packet is never delayed once it has left its start vertex. They give complexity results and algorithms for finding direct schedules. Peis et al.

[27] present non-approximability results for the packet routing problem on several graph classes and algorithms for the problem on trees.

In [19] Leighton et al. show that there is always a routing schedule that finishes in $O(C+D)$ steps. In [20] Leighton et al. present an algorithm that finds such a schedule in polynomial time. However, this algorithm is not suitable for practical applications since the hidden constants in the schedule length are very large. There are also some local algorithms for this problem, needing $O(C) + (\log^* |\mathcal{M}|)^{O(\log^* |\mathcal{M}|)} D + poly(\log |\mathcal{M}|)^6$ [28] and $O\left(C + D + \log^{1+\epsilon} |\mathcal{M}|\right)$ [25] steps with high probability. For the case that all paths are shortest paths, Meyer auf der Heide et al. [4] present a randomized online routing protocol which needs only $O(C + D + \log |\mathcal{M}|)$ steps with high probability. Using the algorithm by Leighton et al. as a subroutine Srinivasan and Teo [31] present an algorithm that solves the packet routing problem with variable paths with a constant approximation factor. Koch et al. [18] improve this algorithm for the more general message routing problem (where each message consists of several packets).

Symnovis [32] show that every permutation on a tree with n vertices can be routed in $n-1$ routing steps. Alstrup et al. [3] give a direct routing algorithm for trees that finds such a schedule of length $n-1$ in sub-quadratic time. Mansour and Patt-Shamir [23] study greedy scheduling algorithms (algorithms that always forward a packet if they can) in the setting where the paths of all packets are shortest paths. They prove that in this setting every packet M_i reaches its destination after at most $D_i + |\mathcal{M}| - 1$ steps where D_i is the length of the path of M_i and $|\mathcal{M}|$ is the number of packets in the network. Thus, giving priority to the packets according to the lengths of their paths yields an optimal algorithm if we assume that the path-lengths are pairwise different.

Leighton, Makedon and Tollis [21] show that the permutation routing problem on an $n \times n$ grid can be solved in $2n-2$ steps using constant size queues. Rajasekaran [29] presents several randomized algorithms for packet routing on grids. They also give their bounds in terms of the grid size. For the case that each vertex of the grid is the start vertex of at most one packet Mansour and Patt-Shamir [24] present an algorithm with constant approximation factor which uses the algorithm by Leighton et al. [20] as a subroutine.

The packet routing problem is related to the multi-commodity flow over time problem [9, 10, 14, 15, 17]. In particular, Hall et al. [14] show that the latter problem is NP-hard, even in the very restricted case of series-parallel networks. It is equivalent to the packet routing problem if we additionally require unit edge capacities, unit transit times, and integral flow values. If there is only one start and one destination vertex then the packet routing problem can be solved optimally in polynomial time, e.g., using the Ford-Fulkerson algorithm for the maximum flow over time problem [7, 11, 12] together with a binary search framework. For the quickest transshipment problem with multiple sources and multiple sinks Hoppe and Tardos [17] present a polynomial time algorithm. As a consequence, the packet routing problem with a single start vertex or a single destination vertex can be solved optimally.

Finally, Adler et al. [1, 2] study the problem of scheduling as many packets as possible through a given network in a certain time frame. They give approximation algorithms and NP-hardness results.

1.3. **Our Contributions.** For the case of the bidirectional grid $\overleftrightarrow{G}_{\#}$ with the start- and the destination vertices of the packets being pairwise different we present an optimal algorithm which always computes a schedule of length D. For the undirected grid $G_{\#}$ it can be adapted to a 2-approximation algorithm (we will show later that in $G_{\#}$ this setting is NP-hard). We also show that on the grid a bad choice for the paths of the packets can yield an arbitrary high approximation factor for the entire problem. This holds even we use an optimal scheduling algorithm once the paths of the packets are fixed.

We investigate the case that there is only one start vertex but arbitrarily many destination vertices. Note here that the algorithm by Hoppe et al. [17] does not necessarily run in polynomial time since in our case the graph is not part of the input but it is given implicitly. We present an algorithm that finds a schedule with length at most $OPT + 8$. Then we improve this algorithm to a $1+\epsilon$ approximation while still guaranteeing an absolute error of eight. Denote by k the number of sink vertices and by n the length of the overall input. The runtime of our algorithm is bounded by $O\left(n \log n + f\left(1/\epsilon\right)\right)$ (for an exponential function f). For the same setting we give an optimal algorithm with running time $O\left(k^6 n\right)$. We achieve the polynomial bound on the runtime by not considering the full grid but only certain subgrids. Then we present a $(b+1)$-approximation algorithm for packet routing on the grid where we assume that the paths are fixed and have at most b bends each. For the case that the start- and destination vertices are sparsely distributed (i.e., in each row/column there is at most one start/destination vertex) we present a 9-approximation algorithm.

Finally, we study the complexity of the packet routing problem on grids. We prove that if the paths are fixed, it is NP-hard on the bidirected grid $\overleftrightarrow{G}_{\#}$, even if there is only one start vertex and all predefined paths are shortest paths. Allowing the paths to be variable, we show that the problem is still NP-hard on the undirected grid $G_{\#}$ even if no two packets share their start or destination vertex.

2. Unique Start and Destination Vertices

In Section 6 we show that the packet routing problem with variable paths on the unidirectional grid graph $G_{\#}$ is NP-hard even if no two packets have the same start or destination vertex. In this section we present an optimal algorithm for the same problem on the bidirectional grid $\overleftrightarrow{G}_{\#}$. A slight modification yields a factor 2 approximation algorithm on $G_{\#}$. Moreover, we show that a bad choice of the paths can result in arbitrarily bad schedules, even in the case of unique start and destination vertices and under the assumption that an optimal routing schedule is used.

2.1. **Optimal Algorithm for $\overleftrightarrow{G}_{\#}$.** Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of the packet routing problem with variable paths. Assume that for each pair of packets $M = (s,t)$ and $M' = (s',t')$ it holds that $s \neq s'$ and $t \neq t'$. We first need to specify the paths of the packets and then the routing schedule. The path of each packet $M = \left((s_x, s_y), (t_x, t_y)\right)$ is defined as follows: First, M moves vertically on the (unique) shortest path from (s_x, s_y) to (s_x, t_y) (call this the vertical part) and then horizontally on the (unique) shortest path from (s_x, t_y) to (t_x, t_y) (call this the horizontal part).

Since each vertex is the start vertex of at most one packet, there can be no delay in the vertical part of a packet's path. For the horizontal part we use the farthest-destination-first rule if there are two packets competing for an edge. Denote by $GRID(I)$ the obtained schedule for I and by $OPT(I)$ a schedule with minimum makespan.

Theorem 1. *Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of the packet routing problem such that no two packets share a start- or destination vertex. Then it holds that $|GRID(I)| = D = |OPT(I)|$.*

Proof. Since D is a lower bound for the length of an optimal schedule it is sufficient to show that $|GRID(I)| \leq D$. For a packet M let $makespan(M)$ be the number of steps that M needs to reach its destination in $GRID(I)$.

For packets M_i which are never delayed by another packet it clearly holds that $makespan(M_i) = D_i \leq D$. Since each vertex is the start vertex of at most one packet this holds in particular for packets which do not move horizontally.

Now consider a grid line ℓ and consider all packets that have their destination vertex in ℓ. Define the set $\mathcal{M}_{L,\ell}$ to be the set of packets $M = (v_{i,j}, v_{\ell,j'})$ with $j' < j$ (so in grid line ℓ these packets move to the left in the horizontal part of their path). Analogously, we define the set $\mathcal{M}_{R,\ell}$ to be the set of packets $M = (v_{i,j}, v_{\ell,j'})$ with $j' > j$. We order the packets in $\mathcal{M}_{L,\ell}$ ascendingly by the column of their destination vertex (in the sequel the destination-x-coordinate or short the x-coordinate). We prove by induction that for all packets $M \in \mathcal{M}_{L,\ell}$ it holds that $makespan(M) \leq D$. We start with the packet $M_1 \in \mathcal{M}_{L,\ell}$ with the lowest destination-x-coordinate (note that this packet is unique). Since we use farthest-first routing in the horizontal part and there is no delay by other packets in the vertical part of the path, M_1 will never be delayed by other packets. Thus, $makespan(M_1) \leq D_1 \leq D$. Now assume for the induction hypothesis that for each packet M among the i packets with the lowest x-coordinates it holds that $makespan(M) \leq D$.

Let M_{i+1} be the (unique) packet with the $i+1$-th lowest x-coordinate. Denote its x-coordinate by x_{i+1}. From the algorithm it follows that M_{i+1} can only be delayed by packets whose x-coordinate is strictly lower than x_{i+1}. Consider the time t when M_{i+1} encounters its last delay. Let M^* be the packet that delays M_{i+1} at time t and let x^* be the x-coordinate of M^*. Since $x^* < x_{i+1}$ and from the induction hypothesis we know that $makespan(M^*) \leq D$ it follows that $makespan(M_{i+1}) \leq D$.

A similar reasoning for all rows ℓ and all corresponding sets $\mathcal{M}_{L,\ell}$ and $\mathcal{M}_{R,\ell}$ shows that $makespan(M) \leq D$ for all packets. Note that we can consider the sets $\mathcal{M}_{L,\ell}$ and $\mathcal{M}_{R,\ell}$ independently since the paths of the two packets $M \in \mathcal{M}_{L,\ell}$ and $M' \in \mathcal{M}_{R,\ell}$ do not share any edges on their path. Since D is a lower bound for the length of an optimal routing schedule, we conclude that $|GRID(I)| = D = |OPT(I)|$. □

2.2. **2-Approximation for $G_{\#}$.** Now we look at the packet routing problem with variable paths on the grid $G_{\#}$ again with the condition that for each pair of packets $M = (s,t)$ and $M' = (s',t')$ it holds that $s \neq s'$ and $t \neq t'$. Let $I = (G_{\#}, \mathcal{M})$ be such a packet routing instance. Let $\overleftrightarrow{I} = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be a packet routing instance with the packets $\mathcal{M}$ on the bidirectional grid $\overleftrightarrow{G}_{\#}$. First we compute the schedule

$GRID\left(\overleftrightarrow{I}\right)$. Then we adjust it as follows: we split the timesteps into even ($t \in \{0, 2, 4, ...\}$) and odd ($t \in \{1, 3, 5, ...\}$) timesteps (in the analysis this will lead to an approximation factor of two). In all even timesteps we move packets whose next edge goes up or to the right. In all odd timesteps we move packets whose next edge goes down or to the left. Denote by $GRID_2(I)$ the resulting schedule.

Theorem 2. *Let $I = (G_{\#}, \mathcal{M})$ be an instance of the packet routing problem such that no two packets share a start- or destination vertex. Then, $GRID_2(I)$ is a valid schedule and it holds that $|GRID_2(I)| \leq 2 \cdot D \leq 2 \cdot |OPT(I)|$.*

Proof. From Theorem 1 we know that $\left|GRID\left(\overleftrightarrow{I}\right)\right| = D$. Our adjustment of $GRID\left(\overleftrightarrow{I}\right)$ extends it by at most factor 2. This implies that $|GRID_2(I)| \leq 2 \cdot D \leq 2 \cdot |OPT(I)|$. We know that $GRID\left(\overleftrightarrow{I}\right)$ is a valid schedule. However, it might happen that two packets use the same link in opposite directions at the same time. In contrast to $\overleftrightarrow{G_{\#}}$ in $G_{\#}$ we cannot use a link between two vertices by two packets at a time. Since we split the timesteps into even and odd timesteps in $GRID_2(I)$ an edge is used by at most one packet at a time. Thus, $GRID_2(I)$ is a valid schedule since $GRID\left(\overleftrightarrow{I}\right)$ is valid. □

2.3. **Choice of the Paths.** The choice of the paths in Theorem 1 might seem pretty simple. However, in the above setting a bad choice of paths can result in an arbitrarily high approximation factor.

Theorem 3. *For every $k \geq 1$ there is an instance $\Pi_k = \left(\overleftrightarrow{G_{\#}}, \mathcal{M}_k\right)$ of the packet routing problem with variable paths with the following property: Let OPT_k denote an optimal solution for Π_k. There is a choice for the paths of the packets $\mathcal{M}_k$ such that for the best possible makespan OPT'_k using these paths it holds that $\frac{OPT'_k}{OPT_k} \in \Omega(k)$.*

Proof. We define the start and destination vertices of the packets in $\mathcal{M}_k$ as follows: For $i \geq 0$ let $\mathcal{M}'_i = \{((-\ell, -i + \ell), (\ell, 1 + i - \ell)) \,|\, \ell \in \{0, ..., i\}\}$ (see Figure 2.1). Now we define

$$\mathcal{M}_k := \bigcup_{i=0}^{k} \mathcal{M}'_i$$

From the definition of the packets it follows that each vertex is the start and destination vertex of at most one packet. Thus, Theorem 1 implies that $OPT_k = 2k - 1$. Now we define paths for the packets as follows: Let $M = (s, t)$ be a packet, let $P_1 = \{s, v_1, v_2, ..., (0, 0)\}$ be a shortest path from s to $(0, 0)$, and let $P_2 = \{(0, 1), v'_1, v'_2, ..., t\}$ be a shortest path from $(0, 1)$ to t. We define the path for M by $P_1 \cup P_2$. From this definition it follows that all packets use the edge $e = \{(0, 0), (0, 1)\}$ (see Figure 2.2 for a possible choice of these paths). The congestion C is a lower bound for the minimim makespan and we have $k \cdot (k + 1)/2$ packets in total. Thus, for the optimal makespan OPT'_k which we get when we use the paths defined above it holds that $OPT'_k \geq k \cdot (k + 1)/2$. This implies that

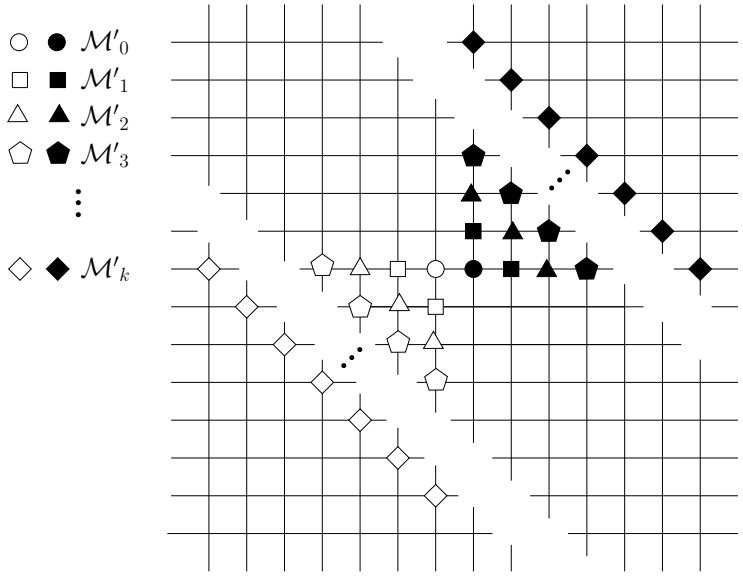

FIGURE 2.1. The packet routing instance constructed in the proof of Theorem 3. The white symbols denote the start vertices and the black symbols denote the destination vertices of the packets in the respective sets.

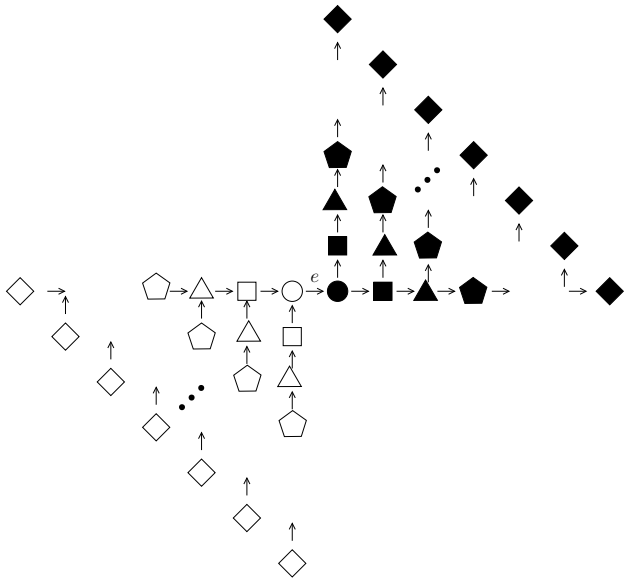

FIGURE 2.2. An inefficient choice for the paths of the packet routing instance.

$$\frac{OPT'_k}{OPT_k} \geq \frac{k \cdot (k+1)}{2} \cdot \frac{1}{2k-1} \in \Omega(k)$$

□

3. Single Start Multiple Destination Vertices

If we allow only one start vertex the packet routing problem with variable paths on arbitrary graphs can be used with the algorithm presented by Hoppe et al. [17]. However, if we are dealing with grid graphs the graph itself is not part of the input but it is given implicitly. Thus, this algorithm does not necessarily run in polynomial time.

First, we present an approximation algorithm with an absolute error of 8. Then we extend this algorithm to an $1+\epsilon$ approximation algorithm. Finally, we present an optimal algorithm with a higher bound on the runtime than the approximation algorithm stated before.

3.1. Absolute Approximation.

Let $I = (G_{\#}, \mathcal{M})$ be an instance of the packet routing problem with variable paths such that there is only one source and arbitrarily many sinks. W.l.o.g. we assume that the start vertex s is located on the grid position $s = (0, 0)$.

Our algorithm to solve this problem has two phases: in the first phase, we construct a schedule $S_0(I)$. For its length it holds that $|S_0(I)| \leq |OPT(I)| + 8$. The schedule is invalid in the sense that some edges are used by more than one packet at a time. We call such incidents a *collision*. In the second phase we modify $S_0(I)$ in order to obtain a schedule $S(I)$ without collisions. We do not change the makespan of the schedule while doing this.

3.1.1. *First Phase: Initial Schedule $S_0(I)$.*

We split the grid into eight segments, one segment for each point of the compass. We define

- $G_{NW} := \{(x, y) \,|x < 0, y < 0\}$
- $G_N := \{(x, y) \,|x = 0, y < 0\}$
- $G_{NE} := \{(x, y) \,|x > 0, y < 0\}$
- $G_E := \{(x, y) \,|x > 0, y = 0\}$
- $G_{SE} := \{(x, y) \,|x > 0, y > 0\}$
- $G_S := \{(x, y) \,|x = 0, y > 0\}$
- $G_{SE} := \{(x, y) \,|x < 0, y > 0\}$
- $G_E := \{(x, y) \,|x < 0, y = 0\}$

We call G_N, G_S, G_E, and G_W the *flat segments* and the other segments the *wide segments*. See Figure 3.1 for a sketch of the segments.

For a packet $M = (s, t) \in \mathcal{M}$ denote by $dist(M)$ be the distance between s and t. In the algorithm, we first sort the packets in $\mathcal{M}$ descendingly by $dist(M)$. Ties are broken arbitrarily. For simplicity of notation let $M_1, M_2, ..., M_n$ be an order such that $dist(M_1) \geq dist(M_2) \geq ... \geq dist(M_n)$. Now we schedule the packets exactly in this order. Since s has four adjacent edges, we schedule four packets at each timestep. To be precise, at time t we schedule the packets M_{4t}, M_{4t+1}, M_{4t+2}, and M_{4t+3} (if packets with the respective indices exist). We assign the four packets arbitrarily to the four outgoing edges of s: these edges form the first edges on the path of the packets.

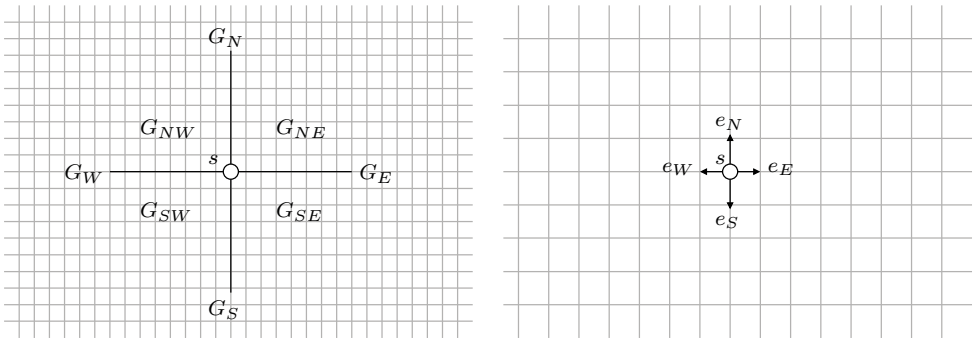

FIGURE 3.1. The segments of the grid (left) and the four outgoing edges of s (right).

Then we iterate over the packets backwards from M_n to M_1. Depending on the first edges on their paths and the segment of their destination vertices, the remainder of the paths is defined as follows:

Denote the outgoing edges of s by e_N, e_E, e_S, and e_W. See Figure 3.1 for a sketch. First we describe the path of a packet for the cases that its destination vertex lies in G_{SE} or G_E. Then the paths for destination vertices in the other segments are obtained by applying the same rules to the grid turned by 90, 180, or 270 degrees. Let M be a packet and let $(x, y) \in G_{SE}$ be its destination vertex. There are the following types of paths we can choose from:

- Type S: $\{(0,0), (0,1), ..., (0, y-1), (0, y), (1, y), ..., (x-1, y), (x, y)\}$
- Type E: $\{(0,0), (1,0), ..., (x-1, 0), (x, 0), (x, 1), ..., (x, y-1), (x, y)\}$
- Type N: $\{(0,0), (0,-1), (1,-1), ..., (x-1,-1), (x,-1),$ $(x+1,-1), (x+1, 0), (x+1, 1), ..., (x+1, y), (x, y)\}$
- Type N': $\{(0,0), (0,-1), (1,-1), ..., (x-1,-1), (x,-1), (x, 0),$ $(x, 1), ..., (x, y-1), (x, y)\}$
- Type W: $\{(0,0), (-1,0), (-1,1), ..., (-1, y-1), (-1, y), (-1, y+1),$ $(0, y+1), ..., (x, y+1), (x, y)\}$
- Type W': $\{(0,0), (-1,0), (-1,1), ..., (-1, y-1), (-1, y),$ $(0, y), ..., (x-1, y), (x, y)\}$

See Figure 3.2 for a sketch of these paths. (Note that in the figures we assume that the x- and y-indices of the vertices increase by going down and to the right, respectively.) If M leaves s through e_S then we set $type(M) := S$ and similarly if M leaves s through e_E then we set $type(M) := E$. For packets leaving s through the edge e_N there are two possible paths, N and N'. In order avoid collisions we choose the paths depending on the packet M' that leaves s through e_E two timesteps after M is scheduled. If the path of M' uses the edge $e_0 = ((x, 0), (x, 1))$ then we set $type(M) := N$, otherwise we set $type(M) := N'$. Note that this ensures that M and M' do not collide. If there is no such packet M' we simply set $type(M) := N'$. We do a similar assignment for packets leaving s through e_W.

Now let M be a packet with the destination vertex $(x, 0) \in G_E$. Depending on the edge through which M leaves s there are different paths we can choose from:

- Type E_0: $\{(0,0), (1,0), ..., (x-1, 0), (x, 0)\}$
- Type N_0: $\{(0,0), (0,-1), (1,-1), ..., (x-1,-1), (x,-1), (x, 0)\}$

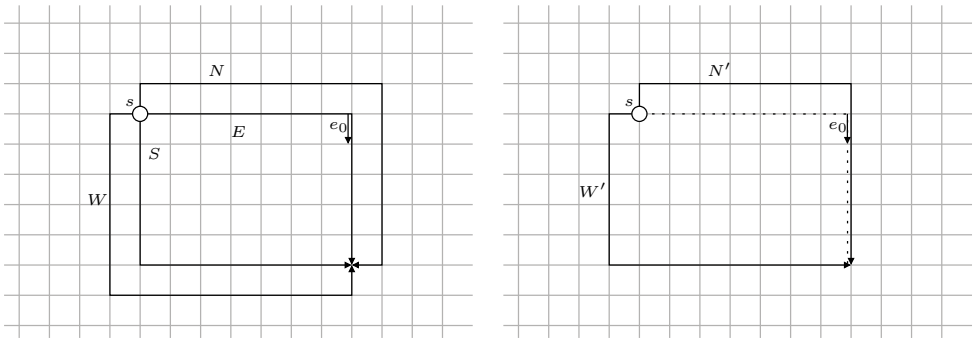

FIGURE 3.2. The possible paths of a packet with its destination vertex being in G_{SE}. The dottet line in the right figure shows the path E.

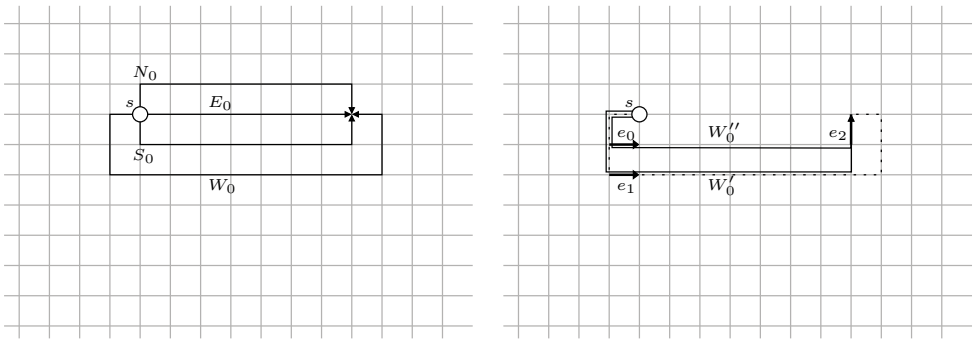

FIGURE 3.3. The possible paths of a packet with its destination vertex being in G_E. The dottet line in the right figure shows the path W_0.

- Type S_0: $\{(0,0),(0,1),(1,+1),...,(x-1,+1),(x,+1),(x,0)\}$
- Type W_0: $\{(0,0),(-1,0),(-1,1),(-1,2),(0,2),(1,2),...,(x-1,2),(x,2),(x+1,2),(x+1,1),(x+1,0),(x,0)\}$
- Type W_0': $\{(0,0),(-1,0),(-1,1),(-1,2),(0,2),(1,2),...,(x-1,2),(x,2),(x,1),(x,0)\}$
- Type W_0'': $\{(0,0),(-1,0),(-1,1),(0,1),(1,1),...,(x-1,1),(x,1),(x,0)\}$

See Figure 3.3 for a sketch. If M leaves s through e_E, e_N, or e_S we set $type(M) := E_0$, $type(M) := N_0$, or $type(M) := S_0$, respectively. If M leaves s through e_W the situation is more complicated. Let M' be the packet that leaves s through e_S two timesteps after M. If the path of M' does not use the edge $e_0 = ((0,1)(1,1))$ then we set $type(M) := W_0''$. Now assume that the path of M' uses the edge e_0. Then we know that M will not collide with M' if the path of M uses the edge $e_1 = ((0,2)(1,2))$. Now let M'' be the packet that leaves s through e_S four timesteps after M. If M'' does not use the edge $e_2 = ((x,y)(x,y+1))$ then we set $type(M) := W_0'$. Note that this ensures that M and M'' do not collide. If M'' uses the edge e_2 then we set $type(M) := W_0$. This ensures that M and M'' do not collide.

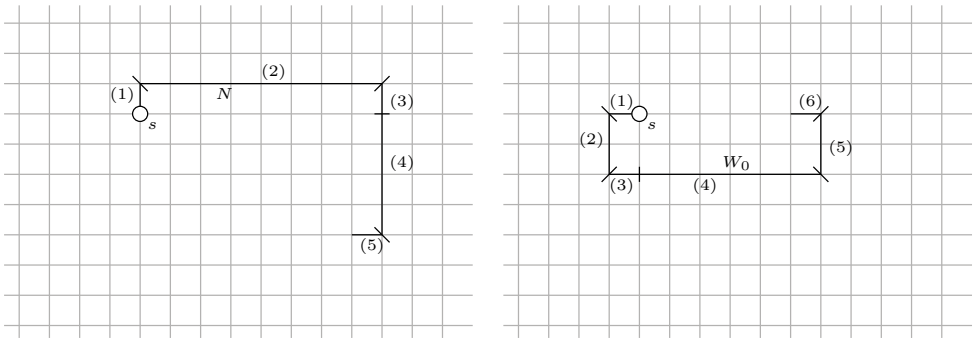

FIGURE 3.4. The segments of the paths N and W_0 as defined in the proof of Lemma 4.

If $(x, y) \notin G_{SE} \cup G_E$ we use the same rules as stated above for the grid turned by 90, 180, or 270 degrees. We route the packets along the described paths. Denote by $S_0(I)$ the resulting schedule. Unfortunately, in general $S_0(I)$ is not a valid schedule since there might be two packets M_1 and M_2 that need to use an edge $e = (x, y)$ at the same time t (a collision).

Lemma 4. *Let M_1 and M_2 be two packets that collide in edge $e = (u, v)$ at time t. Then M_1 and M_2 need to use e in opposite directions, i.e., at time t the packet M_1 is located at vertex u and M_2 is located at vertex v or vice versa.*

Proof. Let M be a packet with its destination vertex (x, y) being a wide segment. W.l.o.g. we assume that $(x, y) \in G_{SE}$. Let t be the time when M leaves s. First we assume that $type\,(M) = N$. We want to prove that M is not involved in a direct collision, i.e., a collision in which both packets need to use an edge in the same direction at the same time. In order to show this, we split the path of M into the five parts $\{(0,0),(0,-1)\}$, $\{(0,-1),...,(x+1,-1)\}$, $\{(x+1,-1),(x+1,0)\}$, $\{(x+1,0),...,(x+1,y)\}$, $\{(x+1,y),(x,y)\}$ (see Figure 3.4).

(1) Let $M' \neq M$ be a packet that uses the edge $\{(0,0),(0,-1)\}$ at time t. This implies that $type\,(M') \in \{N, N', N_0, N_0', N_0''\} =: \mathcal{N}$ and that M' leaves s at time t. But this is a contradiction since M leaves s at time t and $type\,(M) = N$. Therefore, M is not involved in a direct collision at the edge $\{(0,0),(0,-1)\}$ at time t.

(2) Let $M' \neq M$ be a packet that is involved in a direct collision with M on one of the edges on the path $(0,-1),...,(x+1,-1)$. Analyzing the possible paths of the packets (depending on their type and their destination segment) shows that $type\,(M') \in \mathcal{N}$ and M' must have left s at time t which is a contradiction.

(3) Let $M' \neq M$ be a packet that has a direct collision with M on the edge $((x+1,-1),(x+1,0))$. Analyzing all possible paths of a packet shows that M' must have left s at time t and $type\,(M') \in \{N, N'\}$.

(4) Let $M' \neq M$ be a packet that is involved in a direct collision with M on one of the edges on the path $(x+1,0),...,(x+1,y)$. Analyzing all possible paths of M' shows that either M' must have left s at time t and $type\,(M') \in \{N, N'\}$ (which is a contradiction) or $dest\,(M') \in G_{SE}$ and $type\,(M') = E$. In the latter case we know that M' left s at time $t + 2$.

However, from the assignment of the types of the paths we conclude that M' also uses the edge $e_0 = ((x,0),(x,1))$ which is a contradiction.

(5) Let $M' \neq M$ be a packet that has a direct collision with M on the edge $((x+1,y),(x,y))$. Analyzing all possible paths of a packet shows that M' must have left s at time t and $type\,(M') = N$.

For the other path types for M the claim can be proven similarly. Now let M be a packet with its destination vertex being in a flat segment. W.l.o.g. we assume that $dest(M) = (x,0) \in G_E$. Let t be the time when M leaves s. First we assume that $type\,(M) = W_0$. We split the path of M into the six segments $\{(0,0),(-1,0)\}$, $\{(-1,0),...,(-1,2)\}$,$\{(-1,2),(0,2)\}$,$\{(0,2),...,(x+1,2)\}$, $\{(x+1,2),(x+1,1),(x+1,0)\}$, $\{(x+1,0),(x,0)\}$ (see Figure 3.4).

(1) Let $M' \neq M$ be a packet that uses the edge $\{(0,0),(-1,0)\}$ at time t. This implies that M' left s at time t and that $type\,(M') \in \{W, W', W_0, W_0', W_0''\} =: \mathcal{W}$ which is a contradiction.

(2) Let $M' \neq M$ be a packet that is involved in a direct collision with M on one of the edges on the path $\{(-1,0),...,(-1,2)\}$. This implies that either M' left s at time t and that $type\,(M') \in \mathcal{W}$ (which is a contradiction) or that M' left s at time $t-2$ and that $type\,(M') = N_0''$ and $dest\,(M') \in G_S$. But this is a contradiction since then the algorithm would have defined the type of M' to be N_0 or N_0'.

(3) Let $M' \neq M$ be a packet that is involved in a direct collision with M on the edge $\{(-1,2),(0,2)\}$. This implies that either M' left s at time t and that $type\,(M') \in \mathcal{W}$ (which is a contradiction) or that M' left s at time $t-2$ and that $type\,(M') = N_0''$ and $dest\,(M') \in G_S$. But this is a contradiction since then the algorithm would have defined the type of M' to be N_0 or N_0'.

(4) Let $M' \neq M$ be a packet that is involved in a direct collision with M on the edge $\{(0,2),...,(x+1,2)\}$. This implies that either M' left s at time t and that $type\,(M') \in \mathcal{W}$ (which is a contradiction) or that $dest\,(M') \in G_{SE}$ and $type\,(M') = S$. The latter implies that M' left s at time $t+2$. However, from the assignment of the paths we conclude that M' uses the edge $e_0 = ((0,1)(1,1))$ which is a contradiction.

(5) Let $M' \neq M$ be a packet that is involved in a direct collision with M on one of the edges on the path $\{(x+1,2),(x+1,1),(x+1,0)\}$. This implies that M' left s at time $t+4$ and $type\,(M') \in \{S, S', S_0, S_0'\}$. In particular, M' uses the edge $(x+1,1),(x+1,0)$. But this is a contradiction since $type\,(M) = W_0$ implies that M' uses the edge $e_2 = ((x,0)(x,1))$.

(6) Let $M' \neq M$ be a packet that is involved in a direct collision with M on the edge $\{(x+1,0),(x,0)\}$. This implies that $type\,(M') = W_0$ and M' left s at time t.

For the other path types for M the claim can be proven similarly. □

3.1.2. *Second Phase: Fixing Collisions.* In this phase we adjust the schedule $S_0(I)$ in order to obtain a schedule $S(I)$ without collisions. Due to Lemma 4 we know that in all collisions in $S_0(I)$ the involved packets need to use the respective edge in opposite directions.

Let $e = (u,v)$ be an edge in which at time t a collisions occurs. Let M_1 and M_2 be the packets on the vertices u and v at time t, respectively. We modify the

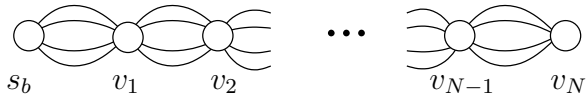

FIGURE 3.5. A sketch of the graph G_b

schedule as follows: In the schedule up to time t we exchange M_1 and M_2. Then at time t the packet M_1 is located on the vertex v and M_2 is located on the vertex u. We do not move either packet in the timestep $[t; t+1]$. Then in the remaining schedule we transfer the packets through the grid as previously defined. Now the collision at the edge e at time t is resolved. Note that no new collisions are created. We do this procedure with all collisions. Denote by $S(I)$ the resulting schedule.

Theorem 5. *For the schedule $S(I)$ it holds that*

$$|S(I)| \leq |OPT(I)| + 8$$

Moreover, the length of $S(I)$ can be computed in $O(n \log n)$.

Proof. In order to prove the approximative error we need a suitable lower bound. In order to establish this bound, we construct an instance $I_b = (G_b, \mathcal{M}_b)$ of the packet routing problem. We define $N := \max_{M \in \mathcal{M}} dist(M)$. We construct G_b as a path with $N+1$ vertices. Between two vertices of the path there are four parallel edges. Denote by s_b the leftmost vertex and by $v_1, v_2, ..., v_N$ the other vertices of G_b (see Figure 3.5 for a sketch). The vertex s_b corresponds to s in $G_{\#}$. For each packet $M \in \mathcal{M}$ we introduce one packet $M_b = \left(s_b, v_{dist(M)}\right)$ in $\mathcal{M}_b$. We claim that the length of an optimal schedule for I_b is a lower bound for $|OPT(I)|$. Let $OPT(I)$ be an optimal schedule for I. We want to construct a schedule S_b for I_b such that $|S_b| \leq |OPT(I)|$. Consider the order in which the packets leave s in $OPT(I)$. We schedule each packet $M_b \in \mathcal{M}_b$ to leave s_b at the same time as its corresponding packet $M \in \mathcal{M}$ is scheduled to leave s in $OPT(I)$. Once a packet has left s_b we move it directly to its destination without any delay. This is possible since at most four packets can leave s_b at a time and there are four edges connecting two adjacent vertices. In order to prove that $|S_b| \leq |OPT(I)|$ we show the following invariant: Let $d(M,t)$ denote the distance of a packet $M \in \mathcal{M}$ to its destination at time t in $OPT(I)$. Similarly, denote by $d_b(M_b,t)$ the distance of a packet $M_b \in \mathcal{M}_b$ to its destination at time t in S_b. We claim that $d(M,t) \geq d_b(M_b,t)$ for all t. Let t_M be the time at which M and M_b leave s and s_b, respectively. For all $t < t_M$ it holds that $d(M,t) = dist(M) = d_b(M_b,t)$. Since after time t_M the packet M_b moves one edge per timestep it holds that $d_b(M_b, t+1) = \min\{d_b(M_b,t) - 1, 0\}$. Since for $d(M,t)$ it holds that $d(M,t+1) \geq \min\{d(M,t) - 1, 0\}$ it is possible to show by induction that $d(M,t) \geq d_b(M_b,t)$ for all t. The schedule $OPT(I)$ is finished at time t if and only if $d(M,t) = 0$ for all packets M. Similarly, the schedule S_b is finished at time t if and only if $d(M_b,t) = 0$ for all packets M_b. We conclude that $|S_b| \leq |OPT(I)|$ and, therefore, $|OPT(I_b)| \leq |OPT(I)|$. Since G_b is a directed path, I_b can be solved optimally using the farthest-destination-first-algorithm[26].

First, we computed the (invalid) schedule S_0. In S_0 the packets are scheduled according to the distance to their respective destination vertices. Moreover, for each packet $M = (s,t)$ we chose a path which is at most eight edges longer than a shortest path between s and t (paths of type W_0 have the maximum detour of

8). This implies that $|S_0| \leq |I_b| + 8 \leq |OPT(I)| + 8$. Since in the construction of S from S_0 the length of the schedule does not change it follows that $|S(I)| = |S_0(I)| \leq |OPT(I)| + 8$. For the computation of $|S_0(I)|$ we first order the packets descendingly by $dist(M)$. This can be done in $O(n \log n)$. Let $M_1, M_2, ..., M_n$ be the resulting order. We partition the packets in sets $\mathcal{M}_k$ of four packets each with $\mathcal{M}_k = \{M_{4k}, M_{4k+1}, M_{4k+2}, M_{4k+3}\}$. In order to determine the type of the path of a packet M we need to check the type of at most two packets which are scheduled after M. Doing this for all packets requires $O(n)$ time. Denote by $|path(M)|$ the length of the path of a packet M. In the schedule, a packet $M \in \mathcal{M}_k$ is delayed for k timesteps before it leaves the source. Thus, it holds that

$$|S_0| = \max_k \max_{M \in \mathcal{M}_k} |path(M)| + k$$

The sorting of the packets dominates the runtime of the computation. Therefore, $|S_0|$ can be computed in $O(n \log n)$. □

Corollary 6. *We can compute an upper bound $\bar{S}(I)$ for the length of $S(I)$ with the property that*

$$|S(I)| \leq \left|\bar{S}(I)\right| \leq |OPT(I)| + 8$$

in $O(k \log k)$ time where k denotes the number of destination vertices.

Proof. In order to achieve a runtime of $O(k \log k)$ we construct the ordering $M_1, M, ..., M_n$ implicitly by ordering the destination vertices by their distance to the origin. This can be done in $O(k \log k)$. For each sink t_k let $last(t_k)$ be the time in the schedule S_0 when the last packet with destination t_k leaves s. Let $dist(t_k)$ be the distance between s and t_k. Then it holds that

$$|S(I)| = |S_0(I)| \leq \max_k dist(t_k) + last(t_k) + 8 =: \left|\bar{S}(I)\right|$$

The value of $\left|\bar{S}(I)\right|$ can be computed in $O(k)$ time when the ordering of the destinations is given. For proving the absolute error recall the lower bound S_b established in the proof of Theorem 5. It holds that $|S_b| = \max_k dist(t_k) + last(t_k) \leq OPT(I)$. This implies that $\left|\bar{S}(I)\right| \leq |OPT(I)| + 8$. Since $|S_0(I)| \leq |S_b| + 8$ we have that $|S(I)| \leq \left|\bar{S}(I)\right|$. □

3.2. **Approximation Scheme.** We can improve the algorithm described above to an algorithm which finds a schedule $S_\epsilon(I)$ such that $|S_\epsilon(I)| \leq |OPT(I)| + 8$ and $|S_\epsilon(I)| \leq (1+\epsilon)|OPT(I)|$. The idea is to solve instances which allow short schedules optimally. Instances whose makespan is provably long are being solved using the algorithm described above.

Let $\epsilon > 0$ and let n be the number of packets in I. First we compute the dilation D which is the maximum distance of a packet $M = (s, t)$ between s and t. In an instance of the packet routing problem with variable paths the congestion C is not immediately clear since it depends on the chosen paths. However, here we obtain a lower bound for C by $C_{min} := n/4$. Clearly, C_{min} and D are lower bounds for $OPT(I)$.

If $D \leq 8/\epsilon$ and $C_{min} \leq 8/\epsilon$ we compute the optimal schedule directly. Note that in this case $n + D - 1 \leq \frac{40}{\epsilon}$ is an upper bound for the length of the optimal schedule. Therefore, only grid cells whose distance to s is at most $\frac{40}{\epsilon}$ need to be considered for this computation. Also, for the number of packets n it holds that

$n \leq \frac{32}{\epsilon}$ (since $n/4 = C_{min} \leq 8/\epsilon$). If $D > 8/\epsilon$ or $C_{min} > 8/\epsilon$ we compute $S(I)$ using the algorithm described above and output it. Denote by $S_\epsilon(I)$ the resulting schedule.

Theorem 7. *For the schedule $S_\epsilon(I)$ it holds that*

$$|S_\epsilon(I)| \leq |OPT(I)| + 8$$

and

$$|S_\epsilon(I)| < (1+\epsilon) \cdot |OPT(I)|$$

Moreover, the length of $S_\epsilon(I)$ can be computed in $O\left(f\left(\frac{1}{\epsilon}\right) + n \log n\right)$ time for an exponential function f.

Proof. If $D \leq 8/\epsilon$ and $C_{min} \leq 8/\epsilon$ then the claims for the approximation guarantee are clear, since we compute an optimal schedule. For the runtime note that from the above argumentation the routing instance has only $O\left(\frac{1}{\epsilon}\right)$ packets. Also, only $O\left(\frac{1}{\epsilon^2}\right)$ grid cells need to be considered (the other grid cells are further away from the origin than our upper bound on the length of an optimal schedule). Assuming that ϵ is constant we conclude that there are only a constant number of possible distributions of the packets on the grid cells we need to consider. Thus, the optimal schedule for I can be computed in constant time using e.g. breadth-first search. If $D > 8/\epsilon$ or $C_{min} > 8/\epsilon$ then the first claim follows from Theorem 5. In order to show the second claim we conclude from

$$\frac{8}{\epsilon} \quad < \quad \max\{C_{min}, D\} \leq OPT(I)$$

that

$$\begin{aligned} |S_\epsilon(I)| &\leq |OPT(I)| + 8 \\ &< |OPT(I)| + \epsilon \cdot |OPT(I)| \\ &= (1+\epsilon) \cdot |OPT(I)| \end{aligned}$$

□

Corollary 8. *We can compute an upper bound $\bar{S}_\epsilon(I)$ for the length of $S_\epsilon(I)$ with the property that*

$$|S_\epsilon(I)| \leq \left|\bar{S}_\epsilon(I)\right| \leq |OPT(I)| + 8$$

and

$$\left|\bar{S}_\epsilon(I)\right| \leq (1+\epsilon) \cdot |OPT(I)|$$

in $O\left(f\left(\frac{1}{\epsilon}\right) + k \log k\right)$ time where k denotes the number of destination vertices.

Proof. If $D \leq 8/\epsilon$ and $C_{min} \leq 8/\epsilon$ we compute an optimal schedule in $O\left(f\left(\frac{1}{\epsilon}\right)\right)$. Otherwise we do the same computation as described in the proof of Corollary 6. □

3.3. **Optimal Algorithm.** In this section we present an optimal algorithm for solving the packet routing problem with one start and many destination vertices on the grid. It is based on the algorithm by Hoppe et al. [17] which solves the quickest transshipment problem with many sources and many sinks on arbitrary graphs in polynomial time. We will see that the flows computed by their algorithm are integral (assuming that the capacities and transit times of the given graph are all integral). Thus, the algorithm can be used to solve the packet routing problem with one start vertex optimally.

In the case of grid graphs, the number of vertices which need to be considered for solving the problem might be exponential in the input size (note that the grid is given implicitly and is not part of the input). Thus, the algorithm by Hoppe and Tardos does not yield a polynomial time algorithm for this setting. However, we show how their algorithm can be adjusted to obtain an algorithm which solves the packet routing problem on the grid with a single start vertex optimally in polynomial time.

First we give an outline of the algorithm presented in [17]. Then we show how it can be adjusted in order to obtain a strongly polynomial time algorithm for the packet routing problem on the grid with a single start vertex. The general idea is the following: As part of the computation it is necessary to compute minimum cost flows on the input graph. Instead of computing these flows on the whole grid, we compute them on subgraphs of the grid which result from considering only certain rows and columns. We will denote those graphs by *thinned out grids*. We will show that the resulting flows have the same value as the optimal flows on the entire grid.

As a general concept we introduce dynamic flows over time as defined in [17]. Let $G = (V, E)$ be a graph and let maps τ and u denote the integral transit times and the capacities on the edges, respectively. Let $T \geq 0$ be an integer. The time-expanded graph $G(T) = (V(T), E(T))$ is defined as follows: each vertex $y \in V$ has $T+1$ copies in $V(T)$, denoted by $y(0), ..., y(T)$. Each edge $yz \in E$ has $T - |\tau_{yz}| + 1$ copies in $E(T)$, each with capacity u_{yz}, denoted by $y(\theta)\, z(\theta + \tau_{yz})$ for any time θ such that both $y(\theta)$ and $z(\theta + \tau_{yz})$ are in $V(T)$. In addition, $E(T)$ contains a holdover edge $y(\theta)\, y(\theta + 1)$ with infinite capacity for each vertex y and time $\theta = 0, ..., T-1$.

Definition 9 (Flows over time)**.** A dynamic flow f with time horizon T is a static flow in $G(T)$.

3.3.1. *Test for Feasibility of Dynamic Transshipment Problem.* We sketch the algorithm presented in [17] for testing whether an instance of the dynamic transshipment problem has a solution within a timebound T. Let $G = (V, E)$ be a directed graph with non-negative capacities u_e and integral transit times τ_e for each edge $e \in E$ and a set of terminals $S \subseteq V$ (the sources and the sinks). W.l.o.g. assume that the sources have no ingoing edges and the sinks have no outgoing edges. Denote by $\mathcal{N} = (G, u, \tau, S)$ this dynamic network. Let $v : S \to \mathbb{R}$ define the supplies and demands of the terminals. We call an instance $(\mathcal{N}, v, T)$ of the dynamic transshipment problem *feasible* if there is a dynamic flow which satisfies the given supplies and demands within T timesteps. Denote by U the maximum capacity of an edge. Let $MCF(m, n)$ be the time needed to compute a static mincost-flow in a graph with $m = |E|$ edges and $n = |V|$ vertices. The algorithm presented in [17] for checking whether a dynamic transshipment problem $(\mathcal{N}, v, T)$ is feasible needs a subroutine which finds a lexicographically maximum dynamic flow within

a timebound T. Let f be a dynamic flow with time horizon T. Let $f^-(v)$ be the amount of flow entering a terminal v between $t = 0$ and $t = T$. Similarly, let $f^+(v)$ be the amount of flow leaving v in that time period. We define the flow value $f(v)$ for v by $f(v) := f^+(v) - f^-(v)$.

Definition 10 (Lexicographically Maximum Dynamic Flow Problem)**.** We are given a time horizon T and an ordering for the terminals $s_1, s_2, ..., s_k$. The objective is to find a dynamic flow f with time horizon T such that for all other dynamic flows f' with time horizon T it holds that $(f(s_1), f(s_2), ..., f(s_k))$ is not lexicographically smaller than $(f'(s_1), f'(s_2), ..., f'(s_k))$.

Theorem 11 (Hoppe et al. [17])**.** *Checking whether a dynamic transshipment problem $(\mathcal{N}, v, T)$ with k terminals is feasible can be done in $O\left(k^2 MCF(m, n) \log(nTU)\right)$.*

Proof. We give only an outline of the proof. For full details see [17] and [16]. We discuss the steps that are needed to understand how we adjust this algorithm for the graph $G_{\#}$.

We use Vaidya's algorithm [33] on the polytope $\mathcal{P} \subset \mathbb{R}$ of all transshipment supplies and demands v'_x for $x \in S$, such that $(\mathcal{N}, v', T)$ is feasible. The algorithm needs $O(kL)$ iterations, each consisting of the inversion of a $k \times k$ matrix and one optimization over $\mathcal{P}$. The latter is done by a routine which computes a lexicographically maximum dynamic flow for $\mathcal{N}$ for a given ordering of the terminals. This routine needs k min-cost flow computations (see Section 3.3.3). It can be shown that $L \in O(\log(nTU))$. This gives a total runtime of $O\left(k^2 MCF(m, n) \log(nTU)\right)$. □

3.3.2. *Construction of an Optimal Dynamic Flow.* Let T^* be the minimum value for T such that $(\mathcal{N}, v, T)$ is feasible. In the sequel we give a sketch of the algorithm presented in [17] for computing a flow for $(\mathcal{N}, v, T^*)$ which satisfies all demands. Then we discuss why the resulting flow is integral if all capacities in the network are integral.

First we define a dynamic network $\mathcal{N}'$ by $\mathcal{N}' := \mathcal{N}$. The set of sources in $\mathcal{N}'$ is constructed by adding a source x_0 for each source $x \in S$ and an edge (x_0, x) with infinite capacity and zero transit time. Likewise, we add a sink y_0 for each sink $y \in S$ and an edge (y, y_0) with infinite capacity and zero transit time. The supply map v' is defined by $v'(z_0) = v(z)$ for all sources and sinks $z \in S$. Denote by S' the set of terminals in $\mathcal{N}'$.

The algorithm consists of $k - 1$ iterations in which the network $\mathcal{N}'$ and the map v' are adjusted. In each iteration, we take two sets $R, Q \subset S'$ and pick a terminal $s_0 \in R \setminus Q$. We describe the necessary steps if s_0 is a source, sinks are treated symmetrically. Let s be the original source in S which corresponds to s_0. We assume that there are $i - 1$ other sources in S' which are adjacent to s. Under certain conditions, we add two new sources s_i and s_{i+1} to S'. Also, we add an edge (s_i, s) with zero transit time and capacity α and an edge (s_{i+1}, s) with unit capacity and transit time δ. Using binary searches, the values chosen for α and δ are integral.

Let $\mathcal{N}'$ and v' denote the resulting network with its demands. By construction, if $\mathcal{N}$ and v are integral, then $\mathcal{N}'$ and v' are integral, too. In order to obtain the desired flow for $(\mathcal{N}, v, T)$ we compute a lexicographically maximum dynamic flow in $\mathcal{N}'$ with a certain ordering of the terminals. In Theorem 13 we will show that if the given network is integral then the computed lexicographically maximum dynamic flow is also integral. Thus, our computed dynamic flow is integral.

3.3.3. *Lexicographically Maximum Dynamic Flow.* Now we sketch the algorithm presented in [17] for computing a lexicographically maximum dynamic flow in a network $\mathcal{N}$ within a timebound T. It is used as a subroutine in the algorithm presented in Sections 3.3.2 and 3.3.1. First, we adjust the network $\mathcal{N}$ by adding a super source ψ and edges (ψ, s_i) for all sources s_i with infinite capacity and zero transit time. Denote by $\mathcal{N}^{k+1}$ the resulting network and denote by g^{k+1} the zero flow in this network. In the sequel when computing minimum cost flows and circulations, we will interpret the transit times as costs.

The algorithm consists of k iterations indexed descendingly. In iteration $i \in \{k, ..., 1\}$ we consider the terminal s_i. We construct the network $\mathcal{N}^i$ and the flows f^i and g^i as follows: If s_i is a sink, we add an edge (s_i, ψ) to $\mathcal{N}^{i+1}$ with infinite capacity and transit time $-(T+1)$. We compute a minimum cost circulation f^i in the residual network of g^{i+1} in $\mathcal{N}^i$. Then we define $g^i := f^i + g^{i+1}$. We will show later that equivalently we could have computed g^i as a minimum cost circulation in $\mathcal{N}^i$ and could have defined $f^i := g^i - g^{i+1}$. If s_i is a source, then we remove (ψ, s_i) from $\mathcal{N}^{i+1}$ in order to obtain $\mathcal{N}^i$. Then we compute a maximum flow f^i from ψ to s_i with minimum cost in the residual network of g^{i+1} in $\mathcal{N}^i$. We define $g^i := f^i + g^{i+1}$. We will show later that equivalently we could have computed g^i as the minimum cost circulation in $\mathcal{N}^i$ and could have defined $f^i := g^i - g^{i+1}$. Each flow f^i is split into a standard chain decomposition Δ^i which is later used in order to compute the desired flow (see [17] for details). All these decompositions are accumulated into a chain decomposition $\Gamma := \bigcup_i \Delta_i$.

As a consequence, for the flow $f(e,t)$ on an edge e at a time t it holds that $f(e,t) = \sum_j \lambda_{j,t} h_j$ for flows h_j which result from the chain decompositions Δ^i and coeffients $\lambda_{j,t} \in \mathbb{Z}$. W.l.o.g. we assume that all computed minimum cost flows are integral if optimal integral solutions exist.

Theorem 12 (Hoppe et al. [17])**.** *A lexicographically maximum dynamic flow with k sources and sinks can be computed via k minimum cost flow computations.*

Theorem 13. *If in $\mathcal{N}$ all transit times, capacities and supplies and demands are integral, the resulting lexicographically dynamic flow is also integral.*

Proof. If $\mathcal{N}$ is integral, then in all minimum cost computations integral solutions exist. The claim then follows from the fact that $f(e,t) = \sum_j \lambda_{j,t} h_j$ for integral coefficients $\lambda_{j,t}$. □

3.3.4. *Solving the Packet Routing Problem on $G_\#$ with one Start Vertex Optimally.* The packet routing problem is closely related to the dynamic flow problem. In particular, with the techniques described above we can solve the packet routing problem on $G_\#$ with variable paths assuming that there is only a single start vertex. However, we need some adjustments since the number of vertices which need to be considered for the computation might be exponential in the input length. We will present an algorithm with runs in $O\left(k^6 n\right)$ where k denotes the number of destination vertices and n the length of the overall input.

Let $I_\# = (G_\#, \mathcal{M})$ be an instance of the packet routing problem with variable paths on $G_\#$ with a single start vertex. W.l.o.g. we assume that the source is located on the grid position $s = (0, 0)$. First, we state the following theorem which we will prove in Section 3.3.5.

Theorem 14. *Let $\mathcal{N}_\# = (G_\#, u, \tau, S)$ be a dynamic network with $u_e = 1$ and $\tau_e = 1$ for all edges e, only one source vertex s, and k sink vertices. Checking whether the instance $(\mathcal{N}_\#, v, T)$ of the dynamic transshipment problem is feasible can be done in $O\left(k^6 \log(Tk)\right)$.*

Now we can describe the algorithm for computing the optimal makespan T^* for I. First we compute a constant T with $OPT \leq T \leq OPT + 8$ (see Corollary 6). This implies that for T^* it holds that $T - 8 \leq T^* \leq T$. Let S be a set consisting of the start vertex s and the destination vertices of the packets. We define a dynamic network $\mathcal{N}_\#$ by $\mathcal{N}_\# = (G_\#, u, \tau, S)$ with $u_e = 1$ and $\tau_e = 1$ for all edges e. The map $v : S \to \mathbb{R}$ is defined by

- $v(s_i) = -|\mathcal{M}_i|$ for destination vertices s_i and $\mathcal{M}_i = \{M = (s,t) \,|t = s_i\}$
- $v(s) = |\mathcal{M}|$ for the single start vertex s

For each T' with $T - 8 \leq T' \leq T$ we test the feasibility of $(\mathcal{N}_\#, v, T')$ using the algorithm which we will present in Section 3.3.5. Denote by T^* the minimum T' such that $(\mathcal{N}_\#, v, T')$ is feasible.

Theorem 15. *It holds that T^* is the minimum time needed to solve the instance I of the packet routing problem. Moreover, T^* can be computed in $O\left(k^6 n\right)$.*

Proof. Since in $(\mathcal{N}_\#, v, T^*)$ all capacities, transit times and demands are integral the algorithm presented in Section 3.3.2 constructs an integral dynamic flow for $(\mathcal{N}_\#, v, T^*)$. Thus, this dynamic flow immediately yields a solution for the packet routing problem. This implies that T^* is the minimum time needed to solve I.

Computing T can be done in $O(k \log k)$ (see Corollary 6). This is dominated by the feasibility check for the respective values of T which takes $O\left(k^6 \log(Tk)\right)$ steps (note that we check only eight values for T'). Now let x_i and y_i be the x- and y-coordinates of a sink s_i. We define $N := \max_i \{|x_i|, |y_i|\}$. Denote by n the length of the overall input. Since all x_i and y_i are part of the input, we know that $\log N \leq n$. The length of an optimal routing schedule is bounded by $2N + |\mathcal{M}|$ since there is always a schedule of that length. This implies that $T \leq 2N + |\mathcal{M}| + 8$ and thus $O\left(k^6 \log(Tk)\right) \subseteq O\left(k^6 n\right)$.

□

3.3.5. *Test for Feasibility in $G_\#$.* Let $G_\# = (V_\#, E_\#)$ be the infinite grid graph, let $S \subset V_\#$ denote the set containing the sink vertices and the single start vertex s. We define $u_e = 1$ and $\tau_e = 1$ for all edges e. Then $\mathcal{N}_\# = (G_\#, u, \tau, S)$ is a dynamic network. Denote by $v : S \to \mathbb{R}$ the demand/supply of each terminal and by T the time horizon. We present an algorithm for testing whether $(\mathcal{N}_\#, v, T)$ is a feasible instance of the dynamic transshipment problem. As a subroutine we will need an algorithm which computes a lexicographically maximum flow in $G_\#$. We will present such an algorithm in Section 3.3.6. By definition, the graph $G_\#$ is an undirected graph. In the sequel, we will interpret it as the directed graph obtained by replacing each undirected edge by the standard gadget depicted in Figure 3.6. For simplicity we will still use the vertex and edge notation from $G_\#$.

Theorem 16. *Computing a lexicographically maximum flow in $G_\#$ with one source and k sink vertices can be done in $O\left(k^5\right)$.*

We will prove this theorem in Section 3.3.6.

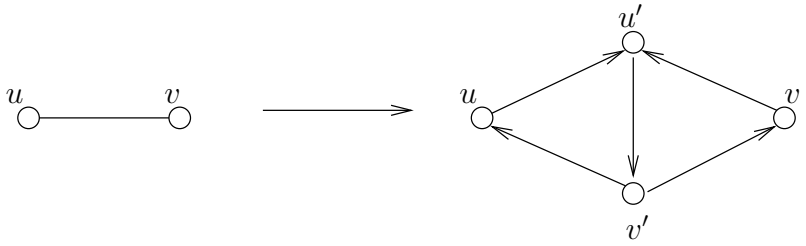

FIGURE 3.6. We replace of each undirected edge (u, v) in $G_\#$ by the gadget seen on the right. The edge (u', v') has the same capacity and transit time as (u, v). All other edges in the gadget have infinite capacity and zero transit time.

Theorem 17. *Let $\mathcal{N}_\# = (G_\#, u, \tau, S)$ be a dynamic network with $u_e = 1$ and $\tau_e = 1$ for all edges e and with only one source vertex s and k sink vertices. Checking whether the instance $(\mathcal{N}_\#, v, T)$ of the dynamic transshipment problem is feasible can be done in $O\left(k^6 \log(Tk)\right)$.*

Proof. We adjust the algorithm presented in the proof of Theorem 11. Again, we use Vaidya's algorithm [33] on the polytope $\mathcal{P} \subset \mathbb{R}$ of all transshipment supplies and demands v'_x for $x \in S$, such that $(\mathcal{N}_\#, v', T)$ is feasible. In each of the $O(kL)$ iterations, we need to invert a $k \times k$ matrix and compute a lexicographically maximum dynamic flow for $G_\#$ with timebound T and the terminals S. Note that we do not need to compute the flow itself but only its flow values. The runtime of the matrix inversion is dominated by the computation of the flow (which needs $O\left(k^5\right)$ steps). For our special case we will show later that $L \in O(\log(Tk))$. Thus, we obtain a total runtime of $O\left(k^6 \log(Tk)\right)$.

In [16] it was proven that $L \in O(\log(nUT))$ (there, U denotes the maximum capacity of an edge). Here we show how this reasoning can be adjusted to show that $L \in O(\log(Tk))$ in our special case. In Lemma 7.1.4 in [16] it was shown that $\mathcal{P} \subseteq B^Z_\infty\left(0, \sqrt{k}nU(T+2)\right)$. In our setting no terminal can receive or send more than four packets per timestep, from time $t = 0$ until time $t = T$. This implies that $\mathcal{P} \subseteq B^Z_\infty(0, 4(T+1))$ and thus $\mathcal{P} \subseteq B^Z_\infty(\hat{x}, 4(T+2))$ and $\mathcal{P} \subseteq B^Z_2\left(\hat{x}, 4\sqrt{k}(T+2)\right)$. With Lemma 7.1.13 this proves that $\mathcal{P} \supseteq B^Z_2\left(0, \frac{1}{\sqrt{k}4(T+2)}\right)$. Lemma 7.1.10 shows that $\mathcal{P}^*_{Z\hat{x}} \subseteq B^Z_2(0, 2k)$. We choose $L \in \Theta(\log(Tk))$. This ensures that $\mathcal{P}$ is contained in a ball of radius 2^L and it contains a ball of radius 2^{-L}. □

3.3.6. *Lexicographically Maximum Dynamic Flow in $G_\#$.* Now we present an algorithm for solving the lexicographically maximum dynamic flow problem in $G_\#$ with one source and k sink vertices. This problem needs to be solved in a subroutine in the proof of Theorem 17. Since the number of vertices which are necessary to consider might be exponential in the input length, the algorithm described in Section 3.3.3 does not immediately yield a polynomial time algorithm for our problem. In order to obtain polynomial runtime, we give an alternative algorithm for the minimum cost flow computations.

Since the algorithm presented in [17] (sketched in Section 3.3.3) assumes that the sources have no ingoing edges and the sinks have no outgoing edges we adjust

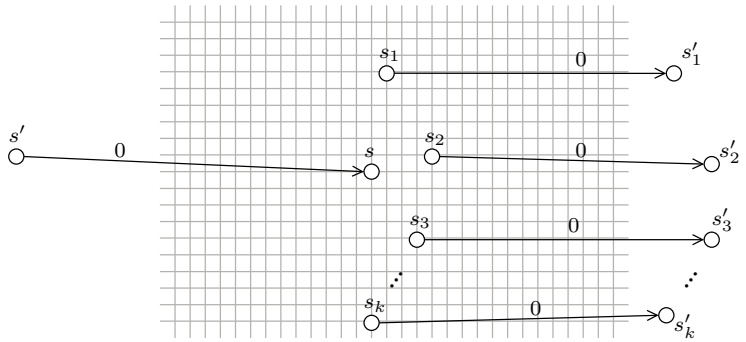

FIGURE 3.7. The network $\bar{G}$ which results from adjusting $G_{\#}$ such that the source s' has no ingoing edges and the sinks $\{s'_1, s'_2, ..., s'_k\}$ have no outgoing edges.

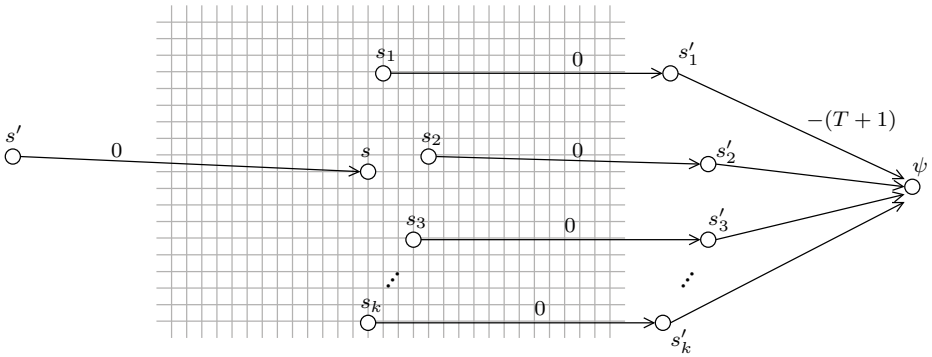

FIGURE 3.8. The network for the OSMCF-problem. The edge (s', s) has length 0 and the edges (s_i, ψ) have length $-(T+1)$. All other edges have unit length.

the graph $G_{\#}$ slightly: we added the new source vertex s' and the directed edge (s', s). Also, for each sink s_i we add a new sink s'_i and a directed edge (s_i, s'_i). All new edges have infinite capacity and zero transit time. Denote by $\bar{G}$ the resulting graph. We define s' to be the only source vertex and $S = \{s'_1, s'_2, ..., s'_k\}$ to be the set of sinks. See Figure 3.7 for a sketch. The mincost-flow instances which need to be solved are equivalent to the following problem:

Definition 18. The *One-Source-MinCostFlow-Problem (OSMCF)* is defined as follows: Let $\bar{G}$ be the adjusted grid graph as defined above. Let $s \in V_{\#}$ be a source and let $S \subset V_{\#}$ be a set of sink vertices. We introduce a super sink ψ and for each $\bar{s} \in S$ we introduce an edge $e_{\bar{s}} := (\bar{s}, \psi)$ with cost $c(e_{\bar{s}}) = -(T+1)$ and infinite capacity. Let G' be the resulting graph. The problem is to find a minimum cost flow with source s' and sink ψ in G'.

See Figure 3.8 for a sketch of the resulting network.

Lemma 19. *All mincost-flow computations in the lexicographically maximum dynamic flow algorithm (presented in Section 3.3.3) applied to $G_{\#}$ with a single source vertex can be reduced to the OSMCF-problem.*

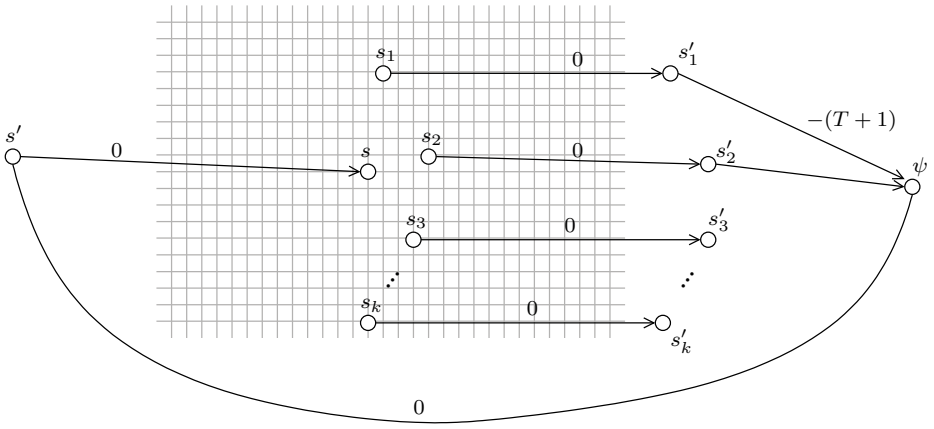

FIGURE 3.9. The network $\mathcal{N}^i$ in the setting of the grid graph with only one source vertex.

Proof. Note that before the algorithm is applied to $G_\#$ the network is adjusted to $\bar{G}$ as defined above. Consider the i-th iteration of the lexicographically maximum dynamic flow algorithm presented in Section 3.3.3. In this iteration, the algorithm takes the terminal s'_i. First we show that the minimum-cost flow computation in this iteration can be equivalently done by computing a standard minimum cost flow in the network $\mathcal{N}^i$. Then we show that this minimum cost flow problem is equivalent to an instance of the OSMCF-problem.

If s'_i is a sink, the algorithm adds an edge (s'_i, ψ) with transit time $-(T+1)$ to create $\mathcal{N}^i$. Then it computes a minimum cost circulation f^i in the residual network of g^{i+1} in $\mathcal{N}^i$, denoted by $\mathcal{N}^i_{g^{i+1}}$. We claim that this is equivalent to computing g^i as a minimum cost circulation in $\mathcal{N}^i$ and defining $f^i := g^i - g^{i+1}$. Let $\hat{g}^i$ be a minimum cost circulation in $\mathcal{N}^i$ and define $\hat{f}^i := \hat{g}^i - g^{i+1}$. We claim that $\hat{f}^i$ is also a minimum cost circulation in $\mathcal{N}^i_{g^{i+1}}$. Denote by $cost(f)$ the cost of a flow f. The flow $\hat{f}^i$ is a flow in $\mathcal{N}^i_{g^{i+1}}$ since $\hat{g}^i$ is a flow in $\mathcal{N}^i$. So now assume on the contrary that $cost\left(\hat{f}^i\right) > cost\left(f^i\right)$. We conclude that

$$\begin{aligned} cost\left(\hat{f}^i\right) &> cost\left(f^i\right) \\ \Leftrightarrow cost\left(\hat{g}^i - g^{i+1}\right) &> cost\left(f^i\right) \\ \Leftrightarrow cost\left(\hat{g}^i\right) - cost\left(g^{i+1}\right) &> cost\left(f^i\right) \\ \Leftrightarrow cost\left(\hat{g}^i\right) &> cost\left(f^i\right) + cost\left(g^{i+1}\right) \\ \Leftrightarrow cost\left(\hat{g}^i\right) &> cost\left(f^i + g^{i+1}\right) \end{aligned}$$

But $f^i + g^{i+1}$ is a flow in $\mathcal{N}^i$ which is a contradiction since $\hat{g}^i$ is a flow in $\mathcal{N}^i$ with minimum cost.

Now we want to show that the computation of a minimum cost circulation in $\mathcal{N}^i$ can be reduced to solving an instance of the OSMCF-problem. The network $\mathcal{N}^i$ for our setting with a single source vertex is sketched in Figure 3.9 (note that depending on the iteration the edge (ψ, s') might not exist). If the edge (ψ, s') does not exist, then all edges with negative costs point to ψ and ψ has no outgoing edges.

Therefore, the minimum cost circulation is simply the zero flow. Now assume that the edge (ψ, s') exists. It has infinite capacity and zero transit time. Thus, the given problem is equivalent to finding a minimum cost flow with source s' and sink ψ in the network $\mathcal{N}^i$ without the edge (ψ, s'). The latter is an instance of the OSMCF-problem.

If $s'_i = s'$, we delete the edge (ψ, s') from $\mathcal{N}^{i+1}$ to create $\mathcal{N}^i$ and compute a minimum-cost maximum flow f^i from ψ to s' in $\mathcal{N}^i_{g^{i+1}}$. We show that equivalently we could have computed g^i as the minimum cost circulation in $\mathcal{N}^i$ and could have defined $f^i := g^i - g^{i+1}$. Let $\hat{g}^i$ be a minimum cost circulation in $\mathcal{N}^i$ and define $\hat{f}^i := \hat{g}^i - g^{i+1}$. We claim that $\hat{f}^i$ is also a minimum-cost maximum flow from ψ to s' in $\mathcal{N}^i_{g^{i+1}}$. With the same arguments as above we can prove that $\hat{f}^i$ is a flow in $\mathcal{N}^i_{g^{i+1}}$ and that $cost\left(\hat{f}^i\right) \leq cost\left(f^i\right)$. Denote by $\left|\hat{f}^i\right|_{s'}$ the amount of flow entering s' in $\hat{f}^i$. Since by assumption s' has no ingoing edges in $\mathcal{N}^i$ it follows that $\hat{g}^i\left((s', s)\right) = 0$. This implies that $\left|\hat{f}^i\right|_{s'} = g^{i+1}\left((s', s)\right)$ which is the maximum amount of flow which can enter s' in $\mathcal{N}^i_{g^{i+1}}$ (again, since s' has no ingoing edges in $\mathcal{N}^i$).

If we want to compute a solution for the lexicographically maximum dynamic flow problem on $G_\#$ there is only a single source s'. Thus, if $s'_i = s'$ all edges with negative costs in $\mathcal{N}^i$ point to ψ and ψ has no outgoing edges. Therefore, the solution of the minimum cost circulation in $\mathcal{N}^i$ is simply the zero flow. □

In Section 3.3.8 we will present a method to solve the OSMCF-problem in $O\left(k^4\right)$.

Theorem 20. *The OSMCF-problem can be solved in $O\left(k^4\right)$ time where k denotes the number of sinks in the network.*

Thus, we obtain the following theorem:

Theorem 21. *Computing a lexicographically maximum dynamic flow in $G_\#$ and its flow values can be done in $O\left(k^5\right)$.*

Proof. As stated in Theorem 12 the computation of a lexicographically optimal dynamic flow needs k mincost flow computations. From Lemma 19 we know that each of these mincost flow computations can be reduced to solving an instance of the OSMCF-problem in $G_\#$. The latter can be done in $O\left(k^4\right)$ (see Theorem 27). Since our algorithm needs k iterations, for this we obtain a runtime of $O\left(k^5\right)$. As we will see in Section 3.3.8 the resulting flow will be given in a simplified grid network with only $O\left(k^2\right)$ edges. Each chain flow in Δ^i (see [17] for details) has a start and an end terminal. Since each terminal vertex has degree four in $G_\#$, we conclude that $\left|\Delta^i\right| \in O\left(k\right)$ and thus $|\Gamma| \in O\left(k^2\right)$.

When we want to compute the flow values for each terminal, we determine for each chain flow in Γ its start and end terminal and its length. Since the length of each path in Γ is bounded by $O\left(k^2\right)$ this can be done in $O\left(k^4\right)$. For a terminal x' denote by $\Gamma_{x'}$ the chain flows which use the edge (x', x) (i.e., chain flows which start in x') and by $\Gamma'_{x'}$ chain flows which use the edge (x, x') (i.e., chain flows entering x'). Then the flow value $f\left(x'\right)$ for the terminal x' is given by

$$f\left(x'\right) = \sum_{\gamma \in \Gamma_{x'}} T - \sum_{\gamma' \in \Gamma'_{x'}} \left(T - \tau\left(\gamma'\right)\right)$$

with $\tau(\gamma')$ being the length of γ'. (Note that in our setting each chain flow has unit value since all edges in $G_\#$ have unit capacity). This gives an overall runtime of $O\left(k^5\right)$ for the computation of a lexicographically maximum flow in $G_\#$ and its flow values. □

3.3.7. *Generic Algorithm for Solving the OSMCF-Problem.* Now we show how the OSMCF-problem can be solved. First, we give a generic algorithm. In the next section we present an efficient implementation which allows the runtime to be bounded by a polynomial in the number of sinks in the network.

As a special case of the general mincost-flow problem the OSMCF-problem can be solved by finding negative cycles and paths from s to ψ with negative cost in the residual network and augmenting the flow along these cycles/paths. (Note that the edge (s', s) has infinite capacity and zero cost and therefore it can be neglected.) If there are no paths or cycles with negative costs in the residual network we know that the computed flow is optimal. We want to augment the flow always along the path from s to ψ with minimum cost in the residual network. We will show that then it is not necessary to augment the flow along cycles. Moreover, we will show that we need at most four path augmentations. Each of these paths can be found using a shortest path computation in the residual network. We compute the mincost flow f as follows:

We define f_0 to be the zero flow. In iteration i (for $0 \leq i \leq 3$) we compute a shortest path P_i in G'_{f_i} from s to ψ. If $c(P_i) \geq 0$ we do nothing (P_i is not an augmenting path). If $c(P_i) < 0$ we augment the flow f_i computed so far along P_i and obtain f_{i+1}. Finally, we output $f := f_4$. Note that all flows are integral since all capacities in the network are integral.

Lemma 22. *Denote by f_i the computed flow after the i-th iteration. It holds that G'_{f_i} does not contain any negative cycles.*

Proof. We prove the lemma by induction. The flow f_0 is the zero-flow. In G_{f_0} all edges with negatives weights are directed to ψ and there are no outgoing edges from ψ. Thus, there are no negative cycles in G_{f_0}. Now assume that G_{f_i} does not contain any negative cycles. From the algorithm we know that P_{i+1} is a shortest path from s to ψ in G_{f_i}. If $c(P_{i+1}) \geq 0$ then the algorithm did not augment f_i along P_{i+1} and we have that $G_{f_{i+1}} = G_{f_i}$. Thus, the claim follows from the induction hypothesis. So now assume that $c(P_{i+1}) < 0$. Assume on the contrary that $G_{f_{i+1}}$ contains a negative cycle C.

Let us consider $C' = P_{i+1} + C$ in G_{f_i}. Since C' is the sum of a path from s to ψ and a cycle we conclude that C' can be expressed as the disjoint sum of a path P'_{i+1} from s to ϕ and a set of cycles $\mathcal{C}$. From the induction hypothesis we know that G_{f_i} does not contain any negative cycles and therefore all cycles in $\mathcal{C}$ have positive cost. This implies that $c\left(P'_{i+1}\right) \leq c(C') = c(P_{i+1} + C) < c(P_{i+1})$ and therefore, P_{i+1} was not the shortest path from s to ψ which is a contradiction. □

Lemma 23. *There is no augmenting path from s to ψ in G'_{f_4}.*

Proof. Assume on the contrary that there is an augmenting path P from s to ψ in G'_{f_4}. First we discuss the case that there is an iteration i in which no path P_i from s to ψ with $c(P_i) < 0$ could be found. This implies that in all iterations $i' \geq i$ no path $P_{i'}$ from s to ψ with $c(P_{i'}) < 0$ could be found and thus there is no such augmenting path in G'_{f_4}. Now we discuss the case that in all iterations i an

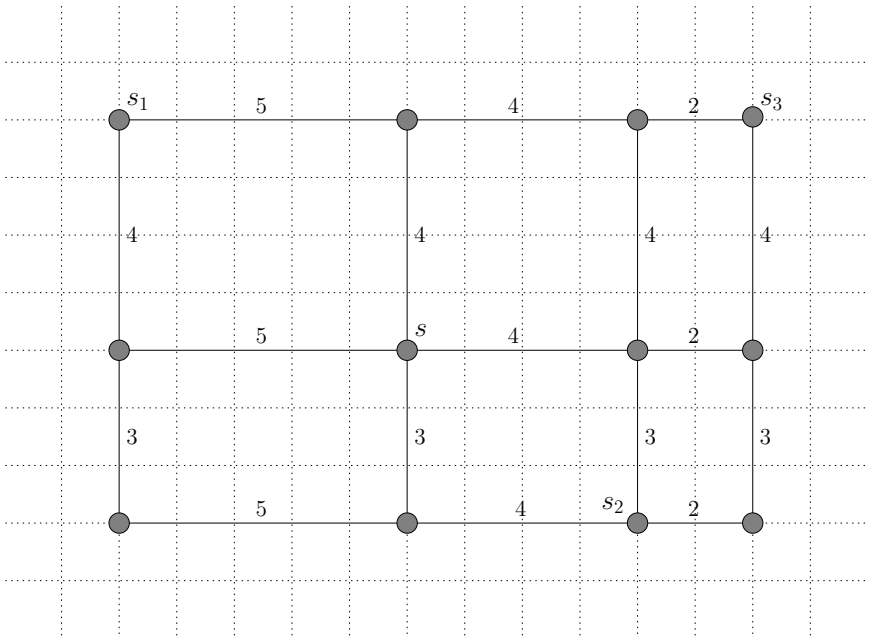

FIGURE 3.10. The thinned grid graph $G_{\#}(S)$ for the vertices $S = \{s, s_1, s_2, s_3\}$. The dottet lines denote the edges of the underlying full grid. The numbers denote the lengths of the respective edges.

augmenting path from s to ψ could be found. Since the out-degree of s is four and all flows in the computation are integral it follows that all outgoing edges of s are saturated. Thus, there can be no augmenting path from s to ψ in G'_{f_4}. □

Theorem 24. *The flow f is a solution for the OSMCF-problem.*

Proof. Follows by standard flow arguments from Lemmas 22 and 23. □

3.3.8. *Implementation for Solving the OSMCF-Problem.* Since $G_{\#} = (V_{\#}, E_{\#})$ is an infinite graph we need some adjustments in order to be able to actually run the computation above. We employ a technique which we call *thinned out grid graphs*. The intuition is the following: Given a set of vertices $S \subset V_{\#}$, we create a grid graph $G_{\#}(S)$ whose rows and columns are exactly the rows and columns of the vertices in S. The lengths of the edges are set according to the distance of their respective end-vertices in $G_{\#}$. See Figure 3.10 for an example.

Formally, let $R(S)$ and $C(S)$ be the sets of the row and column indices of the vertices in S. We define $V_{\#}(S) := \{(x, y) \,|x \in C(S) \wedge y \in R(S)\}$ and $E_{\#}(S) := \bigcup_{i \in R} E_{H,i} \cup \bigcup_{j \in C} E_{V,j}$ with $E_{H,i} := \{\{(c_j, i), (c_{j+1}, i)\} \,|1 \leq j \leq C-1\}$ and $E_{V,j} := \{\{(j, r_i), (j, r_{i+1})\} \,|1 \leq i \leq R-1\}$. We define the cost of an edge $e = \{(c_j, i), (c_{j+1}, i)\}$ by $c(e) = (c_{j+1} - c_j)$ and analogously for an edge $e' = \{(j, r_i), (j, r_{i+1})\}$ by $c(e') = r_{i+1} - r_i$. We define $G_{\#}(S) := (V_{\#}(S), E_{\#}(S))$. We call a graph G a *thinned out grid graph* if there is a set S such that $G = G_{\#}(S)$. Note that the resulting graph is again a thinned out grid graph.

Now let $G = (V, E)$ be a thinned out grid graph. We define an operation $dense(G)$ which, intuitively, makes the thinned out grid a little "denser" by adding

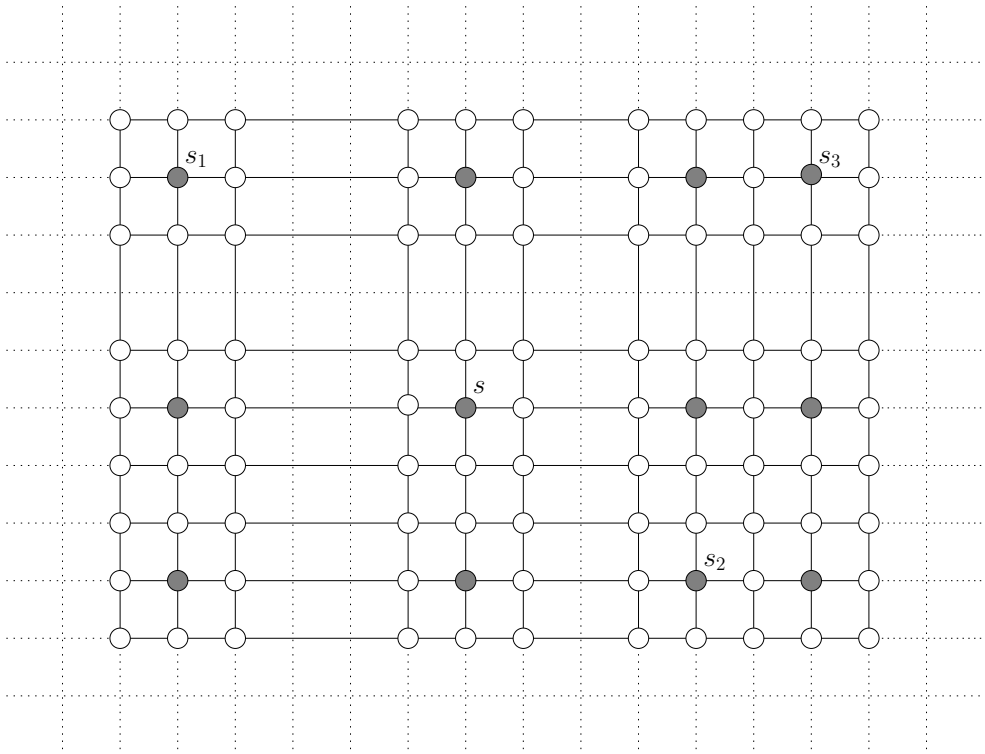

FIGURE 3.11. The graph $dense\,(G_{\#}(S))$ for $S = \{s, s_1, s_2, s_3\}$. The dottet lines denote the edges of the underlying full grid. Vertices which already existed in $G_{\#}(S)$ are marked in gray.

new rows and columns next to already existing ones. We define $V' := \{(x-1, y-1), (x, y), (x+1, y+1) \,|\, (x, y) \in V\}$ and $dense\,(G) := G_{\#}\,(V')$. See Figure 3.11 for an example.

Now we present the variant of the generic algorithm described in Section 3.3.7: We define the graphs G_0, G_1, G_2, G_3 by $G_0 := G_{\#}\,(S \cup \{s\})$ and $G_{i+1} := dense\,(G_i)$. For each graph G_i we define the graph G'_i by introducing a super sink ψ and for each $\bar{s} \in S$ we introduce an edge $e_{\bar{s}} := \{\bar{s}, \psi\}$ with cost $c\,(e_{\bar{s}}) = -(T+1)$ and infinite capacity. In the i-th iteration of the original algorithm, we were looking for an augmenting path in G'_{f_i}. Instead, now we search for an augmenting path in the residual network of f_i in G'_i (denoted by G'_{i,f_i}). In the following lemma we show that this is sufficient in order to find a path of minimum length. The rest of the algorithm remains unchanged.

For technical reasons we define the notion of *detailed thinned out grid graphs.* For a vertex v we define $r(v)$ to be its row and $c(v)$ to be its column index.

Definition 25 (Detailed Thinned Out Grids)**.** Let G be a thinned out grid graph. Then the graph $detail\,(G) = (V_{detail}, E_{detail})$ is defined as follows: Let R and C be the row and column sets of G and let $G_{\#} = (V_{\#}, E_{\#})$. We define $V_{detail} := \{v \in V_{\#} | r(v) \in R \vee c(v) \in C\}$ and $E_{detail} := \{e = (u, v) \in E_{\#} | u \in V_{detail} \wedge v \in V_{detail}\}$ (see Figure 3.12).

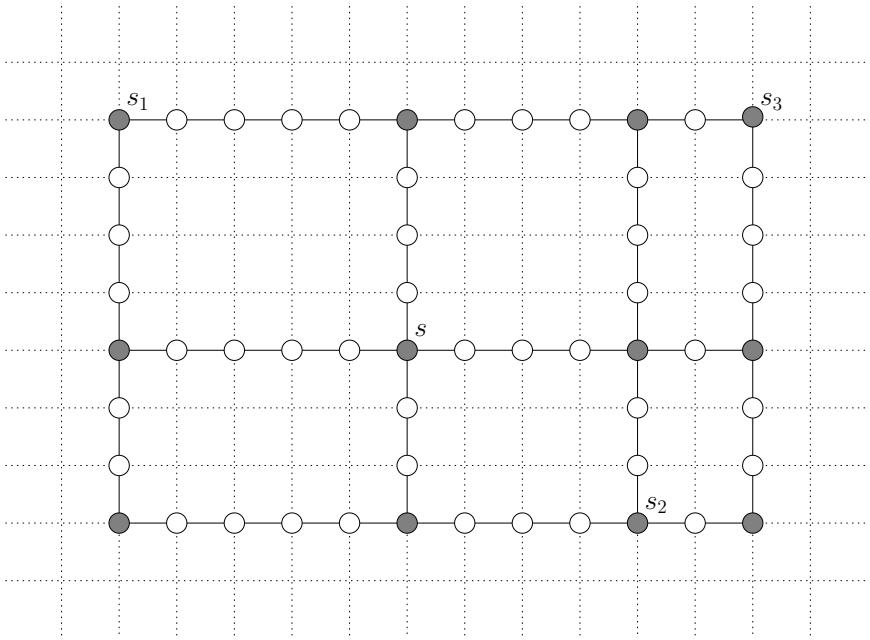

FIGURE 3.12. The graph *detail* $(G(S))$ for $S = \{s, s_1, s_2, s_3\}$. The vertices of $G(S)$ are marked in gray.

In the sequel, we will compute (static) flows f_i for the adjusted thinned out grid graphs G'_i. We will interpret them as flows in the graphs G'_{i+1} and G' in the canonical way.

Lemma 26. *Let k be an integer. There is a path P from s to ψ with $c(P) \leq k$ in G'_{i,f_i} if and only if there is a path P' from s to ψ in G'_{f_i} with $c(P') \leq k$.*

Proof. If there is a path P with $c(P) \leq k$ in G'_{i,f_i} then there is clearly a path P' with $c(P') \leq k$ in G'_{f_i} (recall that G'_{i,f_i} is a subgraph of G'_{f_i}).

For a path $\tilde{P} = (e_0, e_1, ..., e_m)$ in G'_{f_i} let $\tau\left(\tilde{P}\right)$ be the index of the first edge in $\tilde{P}$ which does not belong to *detail* (G_{i,f_i}) or $\tau\left(\tilde{P}\right) = \infty$ if such an edge does not exist. Now let $\mathcal{P}$ be the set of shortest paths from s to ψ in G'_{f_i}. Note that $\mathcal{P} \neq \emptyset$ since G'_{f_i} does not contain any negative cycles, see Lemma 22. If $i = 0$ it is clear that there is a shortest path $P \in \mathcal{P}$ from s to ψ in $G'_{i,f_i} = G'_0$. So from now on assume that $i > 0$. Let $P' \in \mathcal{P}$ be the path in $\mathcal{P}$ which maximizes $\tau(P')$. Note that such a path indeed exists since the support of f_i is finite and thus there are only finitely many shortest paths from s to ψ in G'_{f_i}. Since there are no negative cycles in G'_{f_i} we can assume that P' is simple. We claim that $\tau(P') = \infty$ and thus P' is an augmenting path in *detail* $\left(G'_{i,f_i}\right)$. Assume on the contrary that $\tau(P') = j < \infty$. We will show in the sequel that P' can be changed to a path P'' such that $\tau(P'') > \tau(P')$ with $c(P') \geq c(P'')$. This constradicts the fact that $\tau(P')$ is maximal among all paths in $\mathcal{P}$. Thus, $\tau(P') = \infty$ and we can easily transform P' into a path P in G'_{i,f_i} with $c(P) \leq k$.

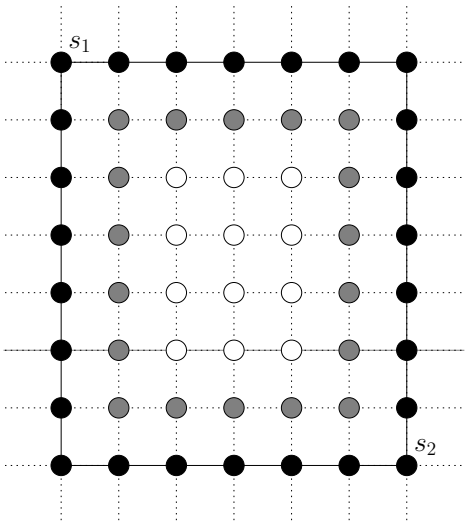

FIGURE 3.13. A face in G'_{i-1} bordered by the vertices s_1 and s_2. The outer vertices are marked in black, the middle vertices are gray, and the inner vertices are white.

First, we partition the vertices of $G_{\#}$ into *outer, middle and inner vertices* (see Figure 3.13). Let R_i, C_i, R_{i-1}, and C_{i-1} be the row and column sets of G'_i and G'_{i-1} respectively. A vertex v is an *outer vertex* if $r(v) \in R_{i-1}$ or $c(v) \in C_{i-1}$. A vertex v is a *middle vertex* if it is not an outer vertex and $r(v) \in R_i$ or $c(v) \in C_i$. A vertex v is an inner vertex if it is neither an outer vertex nor a middle vertex. Since $\tau(P') = j < \infty$ the edge $e_j = (u, v)$ is the first edge in P' which does not belong to *detail* $\left(G'_{i,f_i}\right)$. Let $e_{j-1} = (w, u)$. There are three cases depending on the alignment of e_j. For two vertices u and v in the same row or column denote by $P(u, v)$ the direct path between u and v.

(1) u is an outer vertex and v is a middle vertex. W.l.o.g. we assume that $r(u) = r(v)$, $c(u) = c(v) - 1$, and $r(w) = r(u) + 1$ (see Figure 3.14). Denote the vertices $v_1, ..., v_4$ as depicted in the figure. If $c(P(v_1, u)) < 0$ then $c(P(u, v_2)) = -c(P(v_3, v))$ and thus we can adjust P' to a path P'' by replacing the part from u to v in P' by a path $\{u, ..., v_2, v_3, ..., v\}$. Then $c(P') = c(P'')$ but $\tau(P') < \tau(P'')$. If $c(P(v_1, u)) > 0$ then we can replace the part from v_1 to v in P' by a path $\{v_1, ..., v_4, ..., v\}$ and obtain a path P''. Then P'' has the properties that $c(P') = c(P'')$ and $\tau(P') < \tau(P'')$.

(2) u is a middle vertex and v is an outer vertex. W.l.o.g. we assume that $r(u) = r(v)$, $c(u) = c(v) - 1$, and $r(w) = r(u) + 1$ (see Figure 3.14). Denote the vertices $v_1, ..., v_4$ and the parts of P' as depicted in the figure.

If the path $P'(v_4, v)$ exists in G'_{f_i} (i.e., there is no flow from v_4 to v in f_i) then we can adjust P' to a path P'' by replacing the part from v_1 to v in P' by a path $\{v_1, ..., v_4, ..., v\}$. Then $c(P') \leq c(P'')$ and $\tau(P') < \tau(P'')$. If $P(v_4, v)$ does not exist in G'_{f_i} then $c(P(v_3, v)) = -c(P(u, v_2))$ and we can adjust P' to a path P'' by replacing the part from u to v in P' by a path $\{u, ..., v_2, v_3, ..., v\}$ and obtain $c(P') = c(P'')$ but $\tau(P') < \tau(P'')$.

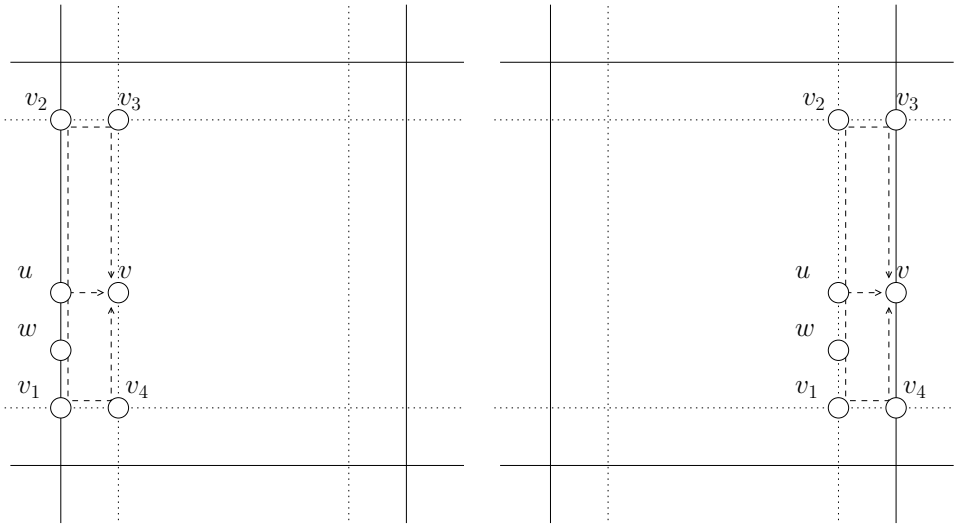

FIGURE 3.14. Cases 1 and 2 in the proof of Lemma 26.

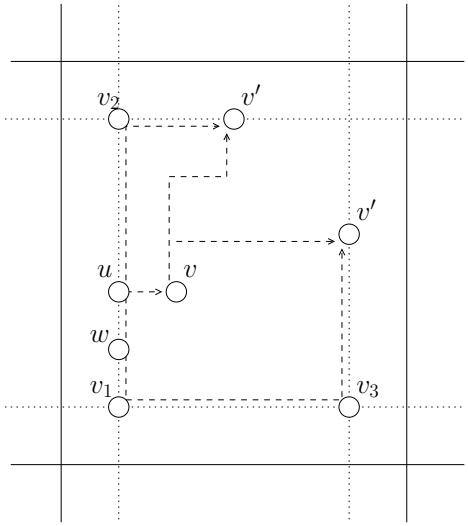

FIGURE 3.15. Cases 3 in the proof of Lemma 26.

(3) u is a middle vertex and v is an inner vertex. W.l.o.g. we assume that $r(u) = r(v)$, $c(u) = c(v) - 1$, and $r(w) = r(u) + 1$. Let v' be the next middle vertex after v in P' (see Figure 3.15). Denote the vertices $v_1, ..., v_4$ as depicted in the figure.
   - If $r(v') = r(v_2)$ then we can adjust P' to a path P'' by replacing the part from u to v' in P' by a path $\{u, ..., v_2, ..., v'\}$ and obtain $c(P') = c(P'')$ but $\tau(P') < \tau(P'')$.

- If $c(v') = c(v_3)$ then we can adjust P' to a path P'' by replacing the part from v_1 to v' in P' by a path $\{v_1, ..., v_3, ..., v'\}$ and obtain $c(P') = c(P'')$ but $\tau(P') < \tau(P'')$.
- If $c(v') = c(u)$ then we can adjust P' to a path P'' by replacing the part from u to v' in P' by a path $\{u, ..., v'\}$ and obtain $c(P') > c(P'')$.
- If $r(v') = r(v_1)$ then we can adjust P' to a path P'' by replacing the part from v_1 to v' in P' by a path $\{v_1, ..., v'\}$ and obtain $c(P') > c(P'')$.

This proves that $\tau(P') = \infty$ and thus we can easily transform P' into an augmenting path P in G'_{i,f_i} with $c(P) \leq k$. □

Theorem 27. *The flow f is an optimal solution for the OSMCF-problem in G'_4. Let f' be a minimum cost flow in from s to ψ in G'. Then $cost(f) \leq cost(f')$. Moreover, f can be computed in $O\left(k^4\right)$.*

Proof. Our algorithm is a special implementation of the algorithm presented in Section 3.3.6 in the following sense: In each iteration i we do not choose arbitrary shortest paths between s and ψ but paths which do exist in G'_{i,f_i}. Lemma 26 shows that such paths are not longer than the shortest paths from s to ϕ in G'_{f_i}. This implies that $cost(f) \leq cost(f')$. The remainder of the reasoning for the correctness works along the lines of Theorem 24.

For the runtime note that each graph G'_i contains at most $k \cdot (1 + 4 \cdot (i-1))$ rows and columns. Since $0 \leq i \leq 3$ the number of vertices in G'_i is bounded by $O\left(k^2\right)$. From the planarity of G'_i it also follows that the number of edges in G'_i is bounded by $O\left(k^2\right)$. For the computation of f we need at most four shortest path computations. Using the Bellman-Ford-Algorithm such a path can be computed in $O(|V| \cdot |E|)$ which gives a runtime of $O\left(k^4\right)$ (for all four paths). Computations needed for constructing the residual network etc. are dominated by the computation of the shortest paths. Thus, we obtain an overall runtime of $O\left(k^4\right)$. □

4. Fixed Number of Bends

We present an algorithm which finds a factor $(b+1)$ approximation for the packet routing with fixed path with the assumption that the path of each packet has at most b bends. Together with suitable routines for finding paths for the packets this yields an algorithm for the packet routing problem with variable paths.

Definition 28 (Number of bends)**.** Let $P = \{v_1, v_2, ..., v_n\}$ be a path on the grid $\overleftrightarrow{G}_{\#}$. The path P has a *bend* at the vertex $v_i = (v_{i,x}, v_{i,y})$ with $1 \neq i \neq n$ if for the vertices $v_{i-1} = (v_{i-1,x}, v_{i-1,y})$ and $v_{i+1} = (v_{i+1,x}, v_{i+1,y})$ it holds that $v_{i-1,x} \neq v_{i+1,x}$ and $v_{i-1,y} \neq v_{i+1,y}$.

Out algorithm works as follows: Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}, \mathcal{P}\right)$ be an instance of the packet routing problem with fixed paths and n packets. Let the number of bends in each path be bounded by b. We split the problem into $b+1$ subproblems. Let $M_j = (s_j, t_j)$ be a packet with the path $P_j = \{v_{j,1}, ..., v_{j,m}\}$. Assume that P_j has ℓ bends. Let $a_1, ..., a_\ell$ be indizes such that the bends of P_j are at the vertices v_{j,a_i} with $1 \leq i \leq \ell$. We define $a_0 = 1$ and $a_{\ell+1} = a_{\ell+2} = ... = a_b = a_{b+1} = m$. We split P_j into $\ell + 1$ subpaths $P_{j,i} = \left\{u_{j,a_i}, ..., v_{j,a_{i+1}}\right\}$ with $0 \leq i \leq \ell$. We do this for all packets. For all i with $0 \leq i \leq b$ we define $\mathcal{M}_i := \left\{\left(v_{j,a_i}, v_{j,a_{i+1}}\right) | 1 \leq j \leq n\right\}$, $\mathcal{P}_i :=$

$\{P_{i,j}|1 \leq j \leq n\}$ and consider the packet routing problem $I_i = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}_i, \mathcal{P}_i\right)$. Since in I_i no path has any bends, this reduces to the problem of finding optimal schedules for the packet routing problem on paths. We solve these problems using the Farthest-Destination-First-algorithm, see [22, Theorem 37.28][1]. Finally, we obtain a schedule for the instance I by successively executing the found schedules for the subproblems I_i. Denote by $BEND(I)$ the resulting schedule.

Theorem 29. *Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}, \mathcal{P}\right)$ be an instance of the packet routing problem with fixed paths. Let the number of bends in each path be bounded by b. It holds that $|BEND(I)| \leq (b+1) \cdot |OPT(I)|$ and $|BEND(I)| \leq (b+1) \cdot (C+D-1)$.*

Let C_i and D_i be the congestion and the dilation in the subproblem I_i. In order to prove the theorem, we first give a lemma that bounds the length of the schedule of a subproblems I_i in terms of C_i and D_i.

Lemma 30. *Let $I_i = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}_i, \mathcal{P}_i\right)$ be one of the subproblems defined in the algorithm. It holds that $|OPT(I_i)| \leq C_i + D_i - 1$.*

Proof. W.l.o.g. we assume that in $OPT(I_i)$ a packet is never delayed if the next edge on its path is free. Let M be a packet which reaches its destination after exactly $OPT(I_i)$ steps. Let D_M be the length of the path of M. If M is never delayed then it holds that $|OPT(I_i)| = D_M \leq C_i + D_i - 1$ (note that $C_i \geq 1$). So now assume that M is delayed at some point in the schedule $OPT(I_i)$. Let $e = (u, v)$ be the last edge on which M is delayed and assume that M arrives at the vertex v at time t (thus, M has passed e at time t). We claim that at each timestep $t' < t$ the edge e was used by a packet. Assume on the contrary that there is a timestep $t_0 < t$ in which no packet needs to use e. This implies that at time t_0 no packet is located on the vertex u. At each timestep at most one packet can arrive at u and at each timestep we can move one packet from u to v. This implies that after t_0 no packet is delayed at u. But this contradicts that M is delayed at u and M reaches v at time t. Note that this implies that $C_i \geq t$. Let P'_M be the path on which M is sent after having passed e. It holds that $|P'_M| = OPT(I_i) - t \leq D_M - 1$. This implies that $OPT(I_i) = (OPT(I_i) - t) + t \leq (D_M - 1) + C_i \leq C_i + D_i - 1$. □

Proof. (of Theorem 29): The optimal makespan $|OPT(I_i)|$ for each problem I_i is a lower bound for the optimal makespan of I. Since we have $b+1$ of these subproblems and solve each subproblem optimally it holds that $|BEND(I)| \leq (b+1) \cdot |OPT(I)|$. Moreover, since from Lemma 30 it holds that $|OPT(I_i)| \leq C_i + D_i - 1$ and $C_i \leq C$ and $D_i \leq D$ for all i we conclude that $|BEND(I)| \leq (b+1) \cdot (C+D-1)$. □

Now we give an alternative routing algorithm which guarantees better bounds in terms of C and D for the makespan. Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}, \mathcal{P}\right)$ be an instance of the packet routing problem with fixed paths. Let the number of bends in each path be bounded by b. For a packet $M_j \in \mathcal{M}$ we split its path P_j into the subpaths $P_{j,i}$ as stated above. The routing schedule is defined as follows: We always move a packet if the next edge on its path is free. A packet M_j is delayed only on the vertices

[1]Note that in our case the Farthest-Destination-First-algorithm produces the same schedule as the Smallest-Slacktime-First-algorithm since the (implicit) deadlines of the packets are all equal.

v_{j,a_i} with $0 \leq i \leq \ell$. The intuition is that if M_j is located on a vertex v_{j,a_i} (where the path of M_j has a bend or v_{j,a_i} is the start vertex of P_j) then M_j waits on this vertex until there is no other packet that needs to use the edge (v_{j,a_i}, v_{j,a_i+1}). In detail, assume that there are several packets $M_1, ..., M_k$ which are located on a vertex u at a time t and they all need to use an edge $e = (u, v)$ next. If for one packet $M \in \{M_1, M_2, ..., M_k\}$ the vertex u is neither a bend nor its start vertex then we move M along e. We call such a packet M a *running packet*. We will show later that there can be at most one running packet on a vertex u. If none of the packets $\{M_1, M_2, ..., M_k\}$ is a running packet we move an arbitrary packet among $\{M_1, M_2, ..., M_k\}$ along e. Denote by $BEND_2(I)$ the resulting schedule.

Theorem 31. *Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}, \mathcal{P}\right)$ be an instance of the packet routing problem with fixed simple paths. Let the number of bends in each path be bounded by b. Then $BEND_2(I)$ is a valid schedule and it holds that $|BEND_2(I)| \leq (b+1) \cdot (C-1) + D$.*

Proof. First we prove that the schedule is well-defined. I.e., we show that if there are several packets $M_1, M_2, ..., M_k$ being located on a vertex u and competing for an edge $e = (u, v)$ then at most one of them is a running packet. We give a proof by induction: at time $t = 0$ the claim is true since there are now running packets (all packets are located on their respective start vertex). Now suppose the claim is true at time $t = t_0$. Let $e = (u, v)$ be an edge. W.l.o.g. assume that e is a horizontal edge with $u = (x, y)$ and $v = (x + 1, y)$. Assume on the contrary that at time $t = t_0 + 1$ there are two running packets M and M' located at the vertex u which need to use e. By the induction hypothesis there has been at most one running packet $\tilde{M}$ on u at time $t = t_0$. From the algorithm it follows that is such a packet $\tilde{M}$ exists then it has been moved over e between time $t = t_0$ and $t = t_0 + 1$. This implies that $M \neq \tilde{M} \neq M'$. Since M and M' are both running packets and they were not located at u at time $t = t_0$ they were located at the vertex $w = (x - 1, y)$ at time $t = t_0$. This is a contradiction, since there can be at most one packet which is transfered over the edge $e' = (w, u)$ between $t = t_0$ and $t = t_0 + 1$.

Now we want to prove the length of the schedule. Let M be a packet. From the algorithm it follows that M waits only in its start vertex or in a vertex where its path P_M has a bend. Denote by $\mathcal{M}_e$ the set of packets whose path uses the edge e. Whenever M waits for using an edge e it is delayed at most $|\mathcal{M}_e| - 1$ times. It holds that $|\mathcal{M}_e| \leq C$. Since P_M has at most b bends (and one start vertex), M is delayed at most $(b + 1) \cdot (C - 1)$ times. Since the length of P_M is bounded by D it follows that M arrives at its destination vertex after at most $(b + 1) \cdot (C - 1) + D$ steps. This implies that $|BEND_2(I)| \leq (b + 1) \cdot (C - 1) + D$. □

Remark 32. Note that our bound $(b + 1) \cdot (C - 1) + D$ for schedule which allows the storage packets in nodes is better than the best known bound $4 \cdot (b + 1) \cdot (C - 1) + D$ for a direct routing schedule.

In the packet routing problem with variable paths we first need to find a path for each packet and then we can compute the schedule. As mentioned above, algorithms for finding paths for the packets on the grid can be combined with our algorithm. This yields an algorithm for the packet routing problem with variable paths. Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of this problem. Let C^* and D^* be the optimal values

for the congestion and the dilation, respectively. In the case of the grid, paths with congestion $C \in O(C^* \log n)$ and dilation $D \in O(D^*)$ and with $b \in O(\log n)$ bends can be found using the algorithm presented by Busch et al. [6]. Using one of the algorithms presented above for scheduling the packets (once the paths are fixed) we obtain the following theorem:

Theorem 33. *Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of the packet routing problem with variable paths. There is a randomized algorithm that finds a routing schedule of length $O(\log n \cdot (C^* \log n + D^*))$ with high probability.*

5. Sparse Start and Destination Vertices

In this section we investigate packet routing on the grid for the case that the start vertices and the destination vertices are sparsely distributed over the grid:

Definition 34. Let $\overleftrightarrow{G}_{\#} = (V_{\#}, E_{\#})$ be the infinite grid graph. A set of vertices $V' \subseteq V_{\#}$ is *sparsely row distributed* or short *row-sparse* if in each grid row there is at most one vertex of V'. Similarly, a set of vertices $V' \subseteq V_{\#}$ is *sparsely column distributed* or short *column -sparse* if in each grid column there is at most one vertex of V'.

We say a set of vertices $V' \subseteq V_{\#}$ is *sparsely distributed* if it is either *sparsely row distributed* or *sparsely column distributed.* Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of the packet routing problem on $\overleftrightarrow{G}_{\#}$ such that the start and destination vertices are sparsely distributed. We present an algorithm which computes a factor 9-approximation algorithm for such instances. Let $S \subset V_{\#}$ denote the set of start vertices and let $D \subset V_{\#}$ denote the set of destination vertices. W.l.o.g. we have one of these two cases: either S and D are both row-sparse or S is row-sparse and D is column-sparse. Note that there could be still two packets which have the same start vertex or the same destination vertex.

5.1. Start and Destination Vertices are row-sparse.

In this section we assume that in I the start and destination vertices are both row-sparse. We assume that the start vertices are ordered such that their row indices increase. W.l.o.g. let $s_1, s_2, ..., s_k$ be this (unique!) order. Now for each start vertex s_i we assign a column for each packet which starts in s_i and goes up (i.e., has its destination vertex in a row above s_i) and we assign a column for each packet which starts in s_i and goes down. First we assign the columns for packets which go up. We iterate over the start vertices from s_1 to s_k. In the ith iteration we consider the vertex $s_i = (r_i, c_i)$. If there are no packets which start in s_i and go up then we skip this iteration. We assume that in the iterations before at most one column was assigned to each start vertex. Let $\mathcal{M}_i$ denote the packets which start in s_i. For a packet M denote by $destRow(M)$ the row of its destination vertex.

Let $L'_i := |\min\{destRow(M) | M \in \mathcal{M}_i\} - r_i|$. Note that L'_i is a lower bound on the optimal makespan of I. We want to assign a column to s_i which has not been assigned to any start vertices s_j with $r_j \in [r_i - L'_i; r_i]$. We call such a column *free*. Since in the previous iterations at most one column was assigned to each start vertex, we can guarantee that in the set $C = \{c | c_i - \lfloor L'_i/2 \rfloor \leq c \leq c_i + \lfloor L'_i/2 \rfloor\}$ there is at least one free column. We assign an arbitrary column in C to s_i. Denote

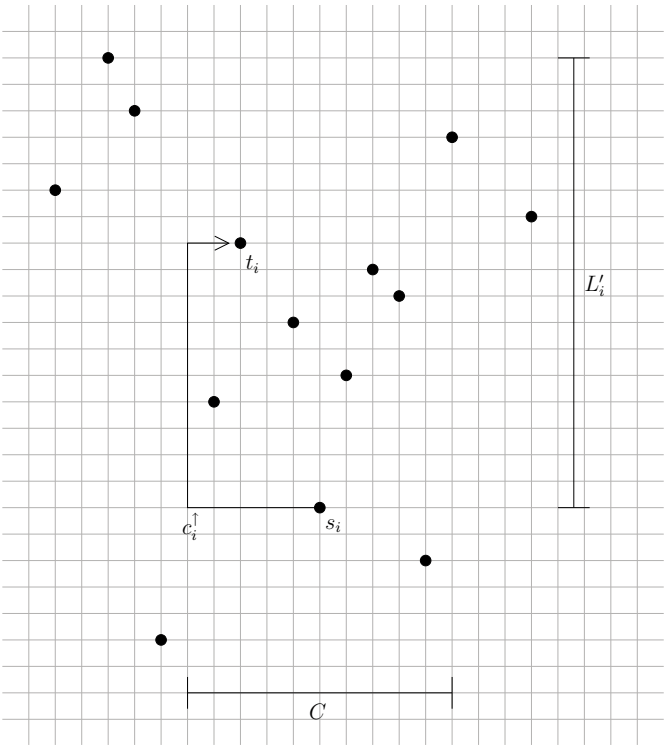

FIGURE 5.1. Sketch for used paths in $SPARSE_{rr}(I)$. For the figure we assume that s_i is the only start vertex and all other vertices which are marked by a circle are destination vertices.

by $c_i^{\uparrow}$ the assigned column. We do this procedure for all start vertices s_i. See Figure 5.1 for a sketch.

Now we assign a column to each start vertex s_i for packets which start in s_i and go down (i.e., have their destination vertex in a row below s_i). At the beginning of this procedure, we consider all columns to be free again. We iterate over the start vertices from s_k to s_1 and do the same operations as above mirroredly. Denote by $c_i^{\downarrow}$ the column assigned to s_i. We define $L_i := \max\left\{\left|c_i^{\uparrow} - c_i\right|, \left|c_i^{\downarrow} - c_i\right|\right\}$.

Now we are ready to define the paths of the packets. Let $M = (s,t)$ with $s = s_i = (s_r, s_c)$ and $t = (t_r, t_c)$ be a packet starting in s_i which goes up (i.e., $t_r < s_r$). Its path goes from $s_i = (s_r, s_c)$ via $\left(s_r, c_i^{\uparrow}\right)$ and $\left(t_r, c_i^{\uparrow}\right)$ to (t_r, t_c). For each packet $M' = (s,t')$ with $s = s_i = (s_r, s_c)$ and $t' = (t'_r, t'_c)$ which goes down (i.e., $t_r > s_r$) we define its path from $s_i = (s_r, s_c)$ via $\left(s_r, c_i^{\downarrow}\right)$ and $\left(t_r, c_i^{\downarrow}\right)$ to (t_r, t_c). For packets which start in s_i and which have their destination vertex in the same row as s_i we define their path simply to be the unique shortest path.

Now we define the routing schedule. We split the schedule into two phases. In phase I we move each packet $M = (s_i, (t_r, t_c))$ from its start vertex to $\left(t_r, c_i^{\downarrow}\right)$ or $\left(t_r, c_i^{\uparrow}\right)$, respectively, along the path described above (packets which do not change

their row are not moved at all). When there are two packets which compete for using an edge (note that such two packets must have the same start vertex) we give priority to the packet which has still the longest way to go to $\left(t_r, c_i^{\downarrow}\right)$ (or $\left(t_r, c_i^{\uparrow}\right)$, respectively). Then in phase II we move the packets along their remaining path. Conflicts are resolved by giving priority to an arbitrary packet. Denote by $SPARSE_{rr}(I)$ the resulting schedule.

Theorem 35. *Let $I = (\overleftrightarrow{G}_{\#}, \mathcal{M})$ be an instance of the packet routing problem on $\overleftrightarrow{G}_{\#}$ such that the start and destination vertices are row-sparse. Then $|SPARSE_{rr}(I)| \leq 9 \cdot |OPT(I)|$.*

In order to prove the theorem, we first establish lower bounds for the two phases of the schedule. First we consider phase I. Let $s_i = (s_r, s_c)$ be a start vertex. Like in the proof of Theorem 5 we construct an instance $I_b = (G_b, \mathcal{M}_b)$ of the packet routing problem. We define $N := \max_{M \in \mathcal{M}_i} |destRow(M) - s_r|$ and $L := \max_i L_i$. We construct G_b as a path with $N + L + 1$ vertices. Between two adjacent vertices of the path we set four parallel edges. Denote by s_b the vertex on the very left and by $v_1, v_2, ..., v_{N+L}$ the vertices on the right of s_b.

The vertex s_b corresponds to s_i in $\overleftrightarrow{G}_{\#}$. For each packet $M \in \mathcal{M}_i$ we introduce one packet $M_b = \left(s_b, v_{|destRow(M)-s_r|}\right)$ in $\mathcal{M}_b$. With similar arguments as in the proof of Theorem 5 we can show that the length of an optimal schedule for I_b is a lower bound for the optimal makespan of I.

Lemma 36. *The packets which start in s_i need at most $4 \cdot |OPT(I_b)| + \lfloor L_i/2 \rfloor$ steps to reach their destination for phase I.*

Proof. Starting with I_b for each pair of adjacent vertices we remove three edges of the four edges which connect them. Denote by I_b' the resulting instance. Since now not four but only one packet can leave s_b at a time we conclude that $|OPT(I_b')| \leq 4 \cdot |OPT(I_b)|$.

Now we distinguish two cases: $\left|c_i^{\uparrow} - c_i\right| \neq 0 \neq \left|c_i^{\downarrow} - c_i\right|$ or either $\left|c_i^{\uparrow} - c_i\right| = 0$ or $\left|c_i^{\downarrow} - c_i\right| = 0$. First we assume that $\left|c_i^{\uparrow} - c_i\right| \neq 0 \neq \left|c_i^{\downarrow} - c_i\right|$.

We change the instance again as follows: we modify each packet $M = (s_b, v_i)$ to $M = \left(s_b, v_{i+\lfloor L_i/2 \rfloor}\right)$. (Note that there is no packet which starts and ends in s_b.) Denote by I_b'' the resulting instance. For $OPT(I_b'')$ it clearly holds that $|OPT(I_b'')| \leq |OPT(I_b')| + \lfloor L_i/2 \rfloor$.

From the choice of the paths it follows that packets can be delayed only in their start vertex (packets with different start vertices do not share edges). Moreover, $|OPT(I_b'')|$ is an upper bound for the time that the packets starting in s_i need for phase I in an optimal schedule for phase I. For the routing of phase I we use farthest-destination-first-routing. This is optimal since the underlying graph structure is an out-tree [22]. This implies that in our schedule the packets starting in s_i need at most

$$\begin{aligned} |OPT(I_b'')| &\leq |OPT(I_b')| + \lfloor L_i/2 \rfloor \\ &\leq 4 \cdot |OPT(I_b)| + \lfloor L_i/2 \rfloor \end{aligned}$$

steps. If $\left|c_i^{\uparrow} - c_i\right| = 0$ or $\left|c_i^{\downarrow} - c_i\right| = 0$ the packets which start in s_i and move up do not interfere with the packets which start in s_i and move down. Thus, this case can be reduced to the first case by considering the two groups of packets separately. □

Similarly to the above we construct a lower bound instance for the routing in phase II. Let $d_i = (d_r, d_c)$ be a destination vertex. From the choice of the paths it follows that in phase II packets interfere only if they share the same end vertex. Let $\mathcal{M}_i$ denote the packets with destination vertex d_i. We distinguish between the packets which enter d_i from the left and from the right. Since their paths do not share any edges in phase II we can consider them separately. Denote by $\mathcal{M}_{i,L}$ the packets with destination vertex d_i which enter d_i from the left (for the packets which enter d_i from the right the reasoning works similarly). Again, we construct an instance $I_b = (G_b, \mathcal{M}_b)$ of the packet routing problem. We define $N := \max_{M \in \mathcal{M}_{i,L}} |startCol(M) - s_r|$ (where $startCol(M)$ denotes the column of the start vertex of M). We construct G_b as a path with $N+L+1$ vertices. Between two adjacent vertices of the path there are four parallel edges. Denote by d_b the vertex on the very left and by $v_1, v_2, ..., v_{N+L}$ the vertices on the right of d_b.

The vertex d_b corresponds to d_i in $\overleftrightarrow{G}_{\#}$. For each packet $M \in \mathcal{M}_i$ we introduce one packet $M_b = \left(v_{|origCol(M)-d_c|}, d_b\right)$ in $\mathcal{M}_b$. With similar arguments as in the proof of Theorem 5 we can show that the length of an optimal schedule for I_b is a lower bound for the optimal makespan of I.

Lemma 37. *The packets which end in d_i and which enter d_i from the left in our schedule need at most $4 \cdot |OPT(I_b)| + \lfloor L/2 \rfloor$ steps in phase II.*

Proof. This can be shown with similar ideas as used in the proof of Lemma 36: We construct an instance I'_b by removing three of the four edges between each pair of adjacent vertices. For this instance it holds that $|OPT(I'_b)| \leq 4 \cdot |OPT(I_b)|$. Then from this we construct the instance I''_b by extending the path of each packet by $\lfloor L/2 \rfloor$ edges. Then $|OPT(I''_b)| \leq |OPT(I'_b)| + \lfloor L/2 \rfloor$. We have that $|OPT(I''_b)|$ is an upper bound on the time that the packets in $\mathcal{M}_{i,L}$ need in an optimal schedule for phase II. Since all packets have the same destination vertex and we are on a path it does not matter what packet we give priority in case that two packet compete for using an edge. Thus, our schedule for phase II is optimal (assuming the given paths to be fixed). Therefore, in phase II the packets in $\mathcal{M}_{i,L}$ need at most

$$\begin{aligned} |OPT(I''_b)| &\leq |OPT(I'_b)| + \lfloor L/2 \rfloor \\ &\leq 4 \cdot |OPT(I_b)| + \lfloor L/2 \rfloor \end{aligned}$$

steps to reach their destination. □

Proof. (of Theorem 35): For each constructed lower bound I_b we have that $|OPT(I_b)| \leq |OPT(I)|$. Thus, Lemmas 36 and 37 show that for phase I and II we need at most $4 \cdot |OPT(I)| + \lfloor L/2 \rfloor$ steps each. Since L is a lower bound for $|OPT(I)|$ this gives that $|SPARSE_{rr}(I)| \leq 9 \cdot |OPT(I)|$. □

5.2. **Start Vertices are row-sparse, Destination Vertices column-sparse.** In this section we assume that in I the start vertices are row-sparse and the destination vertices are column-sparse. For a packet $M = ((s_r, s_c), (t_r, t_c))$ we define its path to go from (s_r, s_c) via (s_r, t_c) to (t_r, t_c). Again, we split the schedule into two phases: In the first phase, we move each packet $M = ((s_r, s_c), (t_r, t_c))$ from

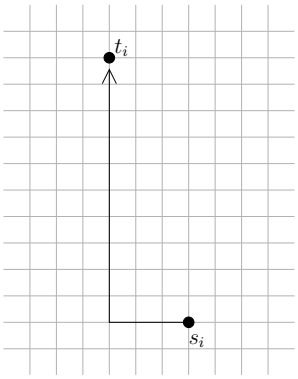

FIGURE 5.2. Sketch for used paths in $SPARSE_{rc}(I)$.

(s_r, s_c) to (s_r, t_c). In the second phase we move each packet $M = ((s_r, s_c), (t_r, t_c))$ from (s_r, t_c) to (t_r, t_c). See Figure 5.2 for a sketch. We schedule each phase according to the farthest-destination-first-rule. Denote by $SPARSE_{rc}(I)$ the resulting schedule.

Theorem 38. *Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of the packet routing problem on $\overleftrightarrow{G}_{\#}$ such that the start vertices are row-sparse and the destination vertices are column-sparse. Then $|SPARSE_{rc}(I)| \leq 8 \cdot |OPT(I)|$.*

Proof. Using similar lower bound instances I_b and I'_b as in the proof of Theorem 35 we can show that phase I and II need at most $4 \cdot |OPT(I)|$ steps each. Since we have two phases this shows that $|SPARSE_{rc}(I)| \leq 8 \cdot |OPT(I)|$. □

5.3. **Start and Destination Vertices Sparsely Distributed.** Now we assume that in I the start vertices S and the destination vertices D are sparsely distributed without any further condition whether they are row-sparse or column-sparse. If S and D are both row- or column-sparse then we compute $SPARSE_{rr}(I)$ (if they are both column-sparse then we mirror the instance in order to be able to apply the algorithm as described above). If S is row-sparse and D is column-sparse then we compute $SPARSE_{rr}(I)$ (again, mirror the instance if necessary). Denote by $SPARSE(I)$ the resulting schedule.

Theorem 39. *Let $I = \left(\overleftrightarrow{G}_{\#}, \mathcal{M}\right)$ be an instance of the packet routing problem on $\overleftrightarrow{G}_{\#}$ such that the start and destination vertices are either row-sparse or column sparse. Then $|SPARSE(I)| \leq 9 \cdot |OPT(I)|$.*

Proof. Follows from Theorems 35 and 38. □

6. Complexity Results

We present complexity results for the packet routing problem on the grid. In particular, we show that the packet routing problem with fixed paths is NP-hard on grid graphs even if there is only one source vertex and all paths are shortest paths. Using a different technique we show that the packet routing problem with variable paths is also NP-hard.

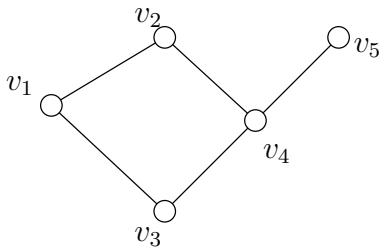

FIGURE 6.1. The example graph G_1.

6.1. **Packet Routing with Fixed Paths.** In this Section we present NP-hardness results for the packet routing problem with fixed paths.

Theorem 40. *The packet routing problem with fixed paths on grid graphs is NP-hard.*

Proof. In this proof we employ a technique which was used in [30, 5]. We reduce to the packet routing problem from the 3-COLORING problem (which is NP-hard [13]), i.e., the problem whether the vertices of a given graph can be colored with three colors such that no two adjacent vertices have the same color.

For each vertex $v_i \in V$ we introduce a packet M_i. The paths of two packets M_i and M_j share an edge if and only if v_i and v_j are adjacent in G. The paths for M_i and M_j are constructed such that if M_i and M_j are never delayed on their path they will eventually cause a collision. The main idea is that if G is 3-colorable, then there is an optimal schedule as follows: In the beginning, each packet M_i is delayed 0, 1, or 2 timesteps depending on the color of v_i. Then the packets can be transfered along their path through the network without ever being delayed again. Thus, if G is 3-colorable there is a schedule such that all vertices are delayed at most twice. Since in G the length of all paths are equal minimizing the makespan is equivalent to finding the minimum k such that each packet is delayed at most k times.

Now we define the construction in detail. Let $G = (V, E)$ be the graph from our 3-COLORING instance. For each vertex v_i we introduce a packet M_i. First we define the paths such that the path of each packet shares one vertex with the path of every other packet. The construction is rather technical to describe, in order to facilitate the readability of the proof we refer to Figure 6.2 for a sketch of the construction for a graph with n vertices.

Now we change the paths slightly. Every time a packet M_i moves up and crosses the path of another packet M_j, we insert a move one step to the right to the path of M_i. If v_i and v_j are adjacent we do this in the row where the paths of M_i and M_j cross, otherwise we do this one row further up. By this modification we ensure that the paths of M_i and M_j share an edge if and only if v_i and v_j are adjacent. Note that all paths still have the same length. Figure 6.3 shows the construction described above for the graph G_1 shown in Figure 6.2.

Denote by C this whole setup. In our final construction we have three copies of C which are glued together (see Figure 6.4 for the overall construction for the graph G_1). Note that the second copy of C has to be mirrored since, e.g., the packet which started in the upper left vertex s_1 of the first copy has to start at the bottom

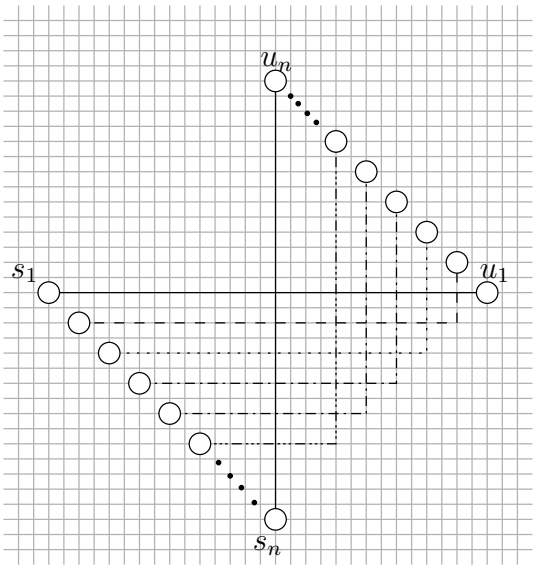

FIGURE 6.2. First path layout for the proof of Theorem 40

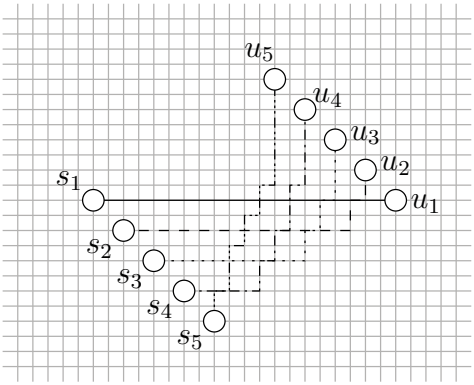

FIGURE 6.3. Basic setup for the proof of Theorem 40

right vertex $t_{1'}$ of the second copy. For each packet M_i denote by s_i its starting vertex, by u_i its last vertex in the first copy of C, by w_i its last vertex in the second copy of C and by t_i its destination vertex. We see that all predefined paths in each copy of C have the same length. Let L be the length of the overall path of each packet. (Note for the upcoming proof that L is divisible by 3.) In the proof we will see that we need the three copies of C to ensure that if two packets M_i and M_j arrive at t_i and t_j at the same time then v_i and v_j are really not adjacent.

The following observation is helpful to understand the proof: let $v_{i,j}$ be a vertex where the paths of the packets M_i and M_j intersect. Then $v_{i,j}$ has the same distance to the start vertices of M_i and M_j. Thus, M_i and M_j arrive at $v_{i,j}$ at the same time if and only if they were delayed the same number of times before reaching $v_{i,j}$.

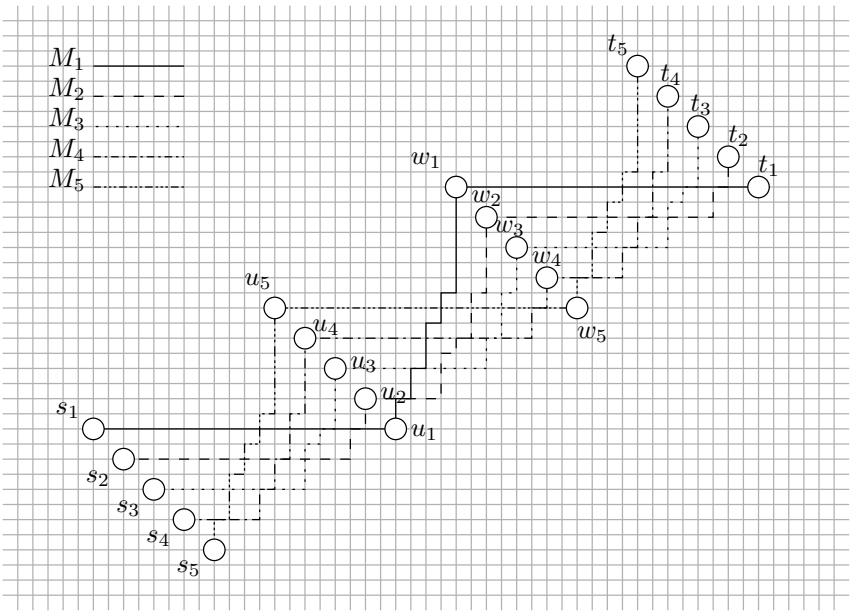

FIGURE 6.4. Basic setup for the proof of Theorem 40, repeated three times. Note that in the second copy the order of the packets when read from the top left to the bottom right is reversed and that the paths of the messages is adjusted to that.

Now we want to prove that G is 3-COLORABLE if and only if there is an optimal schedule of length at most $L+2$. First assume that G is 3-COLORABLE. Thus, there is a valid vertex coloring $c : V \to \{0, 1, 2\}$ for G which uses only three colors in total. We define our schedule as follows: In the beginning, each packet M_i is delayed $c(v_i)$ timesteps. After that each packet moves on its predefined path without being delayed any further. Now we show that no two packets need to use an edge at the same time, i.e., no packet needs to be delayed again. We see that in the first copy of our construction no packet is delayed once it has left its start vertex since this would imply that there are two packets M_i and M_j which collide with $c(v_i) = c(v_j)$ and the vertices v_i and v_j being adjacent. Since all paths of all packets in the first copy of C have the same length $L/3$, we see that a packet M_i arrives at vertex u_i after $L/3 + c(v_i)$ timesteps. With the same reasoning we can show that no packet is delayed in the second or third copy of C either. Thus, the makespan of our schedule is at most $L + 2$ (in the case that G is bipartite or contains only isolated vertices the makespan is even shorter).

Now assume that there is an optimal schedule of length at most $L+2$. We want to show that G is 3-COLORABLE. W.l.o.g. we assume that the schedule never delays a packet when it is not necessary, i.e., packets are delayed only if more than one packet needs to use the same edge at a time. Let v_i be a vertex and let $L + k$ (for some k with $0 \leq k \leq 2$) be the time when M_i reaches t_i (recall that the length of the path is L). We define our coloring $c : V \to \{0, 1, 2\}$ by $c(v_i) := k$. (So we use at most 3 colors.) We want to show that this is indeed a valid vertex coloring for G. Let V_i (for $1 \leq i \leq 3$) be all vertices corresponding to packets which reach the

last vertex of the i-th copy of C after $i \cdot n^2 + (i-1)$ timesteps. We will show that each set V_i is an independent set. This implies that all packets M_i which reach u_i after $L/3$ steps arrive at t_i after L steps. Analogously, all packets M_i which reach w_i after $2 \cdot L/3 + 1$ steps reach t_i after at most $L + 1$. Thus, the coloring defined above is valid.

Let $i = 1$. If a packet M_j arrives at u_j after $L/3$ timesteps, it has never been delayed. Thus, all vertices corresponding to these packets form an independent set. Denote by V_1 these vertices and by $\mathcal{M}_1$ the corresponding packets. Moreover, this implies that a packets $M_j \in \mathcal{M}_1$ reaches w_j after $\frac{2}{3} \cdot L$ timesteps and t_j after L timesteps.

Now let $i = 2$. We know now that a packet M_j which reaches w_j after $\frac{2}{3} \cdot L + 1$ timesteps is not in $\mathcal{M}_1$. Thus, M_j must have reached u_i after $\frac{1}{3} \cdot L + 1$ timesteps (otherwise it would be in $\mathcal{M}_1$ or could not have reached w_i after $\frac{2}{3} \cdot L + 1$ timesteps). This implies that the vertices v_i corresponding to packets M_i which reach w_i after $\frac{2}{3} \cdot L + 1$ timesteps form an independent set. Denote by $\mathcal{M}_2$ the set of the respective packets and by V_2 the set of the respective vertices. We know that each packet $M_j \in \mathcal{M}_2$ reaches t_j after $L + 1$ timesteps (since two packets in $\mathcal{M}_2$ do not delay each other and cannot be delayed by the packets $\mathcal{M}_1$).

Finally consider a packet M_j which reaches t_j after $L + 2$ timesteps. From the above reasoning we conclude that M_j reaches w_j after $\frac{2}{3} \cdot L + 2$ timesteps (otherwise it would be in $\mathcal{M}_1$ or $\mathcal{M}_2$). This implies that all vertices corresponding to packets M_j which reach t_j at time $L + 2$ form an independent set. Since there is no packet M_j which reaches t_i after more than $L + 2$ timesteps, we conclude that the set V splits into three independent set. Moreover, our coloring for G is a valid 3-coloring. □

In the construction in the proof of Theorem 40 there are several start and destination vertices. By extending the construction we can prove that packet routing is NP-hard on grid graphs even if there is only a single start vertex.

Theorem 41. *The packet routing problem with fixed paths is NP-hard on grid graphs, even if there is only one start vertex. This holds even if all predefined paths are shortest paths.*

Proof. We reduce again from 3-COLORING. Let G be a graph. We extend the construction used in the proof of Theorem 40 as follows: We add a super-start vertex v_s. We will call the packets defined so far the *vertex packets*. Then we adjust the paths of the vertex packets such that they all originate at v_s and use the same edge to leave v_s. We also add a set of packets that we call the *delay packets*. These packets delay the vertex packets to ensure that all vertex packets M_i reach their respective vertices s_i at the same time (s_i is the start vertex of M_i according to the construction in the proof of Theorem 40).

The super-start vertex v_s is placed at grid position $(3 \cdot |V| + 1, 0)$. We introduce $|V|$ additional delay packets. They all follow the same predefined path but leave v_s through a different edge than the vertex packets. The path of the delay packets shares exactly one edge with the path of each vertex packet. The length of this path is $3|V| + L + 3$. The whole construction for the graph G_1 (see Figure 6.1) is given in Figure 6.5. The intuition is the following: The given graph is 3-colorable if and only if there is a schedule of length at most $4|V| + L + 2$. In order to achieve this makespan the delay packets must not be delayed by vertex packets at any time.

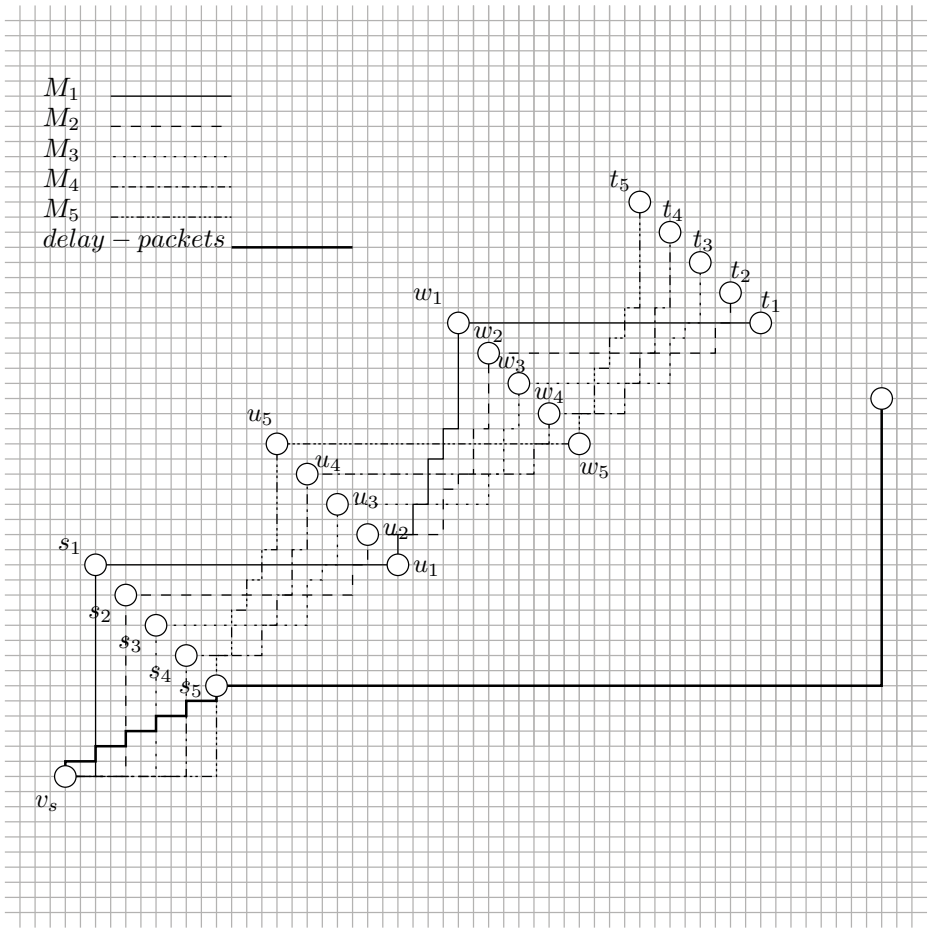

FIGURE 6.5. Construction for the proof of Theorem 41.

Each vertex packet M_i will arrive at s_i after exactly $4|V|$ timesteps. After this point, the whole construction works like in Theorem 40.

Now we prove that G is 3-colorable if and only if there is a schedule of length $4\,|V| + L + 2$. Assume that the graph is 3-colorable. Our schedule works as follows: The delay packets move on their path one after another and they are never delayed by a vertex packet. The vertex packets leave v_s in an arbitrary order. If a vertex packet and a delay packet need to use an edge at the same time then the vertex packet is delayed by the delay packet. Once each vertex packet M_i has reached its respective vertex s_i the vertex packets are scheduled like in the proof of Theorem 40.

Let M_i be a vertex packet and let (c_i, c_i') be the edge which is shared by M_i with the delay packets. From the construction it follows that the length of the path of M_i between v_s and c_i is $3i$. Also, the length of the path of M_i between c_i and s_i is $3|V| - 3i$. This implies that M_i arrives at c_i after at least $3i$ timesteps. Also, the edge (c_i, c_i') is blocked by the delay packets between timesteps $3i$ and $3i+|V|$. Thus, M_i arrives at s_i after exactly $(3i + |V|) + (3|V| - 3i) = 4|V|$ timesteps. Thus, all vertex packets M_i arrive at their respective vertices s_i after exactly $4|V|$ timesteps.

Since the length of the path for the delay packets is $3|V| + L + 3$, there are $|V|$ delay packets in total. Once a delay packet has left v_s it is not delayed any more. Thus, all delay packets reach their destinations after $4|V| + L + 2$ timesteps. Using the same reasoning as in the proof of Theorem 40, we can show that all vertex packets reach their respective destination vertices after at most $4|V| + L + 2$ as well. Thus, there is a schedule of length $4|V| + L + 2$.

Now we assume that there is a schedule with makespan $4|V| + L + 2$. We want to show that the formula is satisfiable. The length of the paths of the delay packets is $3|V| + L + 3$. Since there are $|V|$ delay packets in total and the overall makespan is $4|V| + L + 2$, we conclude that the delay packets leave the vertex v_s one after another and they are never delayed after this. With the same reasoning as above we can conclude that all vertex packets M_i reach their respective vertex s_i after at least $4|V|$ timesteps. With the same reasoning as in Theorem 40 we can show that the graph must be 3-colorable.

We observe that all paths used in the construction are shortest paths. □

Since in the construction of the proof of Theorem 41 start and destination vertices can be exchanged without changing the result we have the following corollary:

Corollary 42. *The packet routing problem with fixed paths is NP-hard on grid graphs, even if there is only one destination vertex and all predefined paths are shortest paths.*

Proof. We simply apply the construction in Theorem 41 and swap start and destination vertices. □

6.2. **Packet Routing with Variable Paths.** A natural thing to look at is the question of whether packet routing on grid graphs is still NP-hard if the paths are not fixed.

Theorem 43. *The packed routing problem with variable paths is NP-hard on the unidirectional grid graph $G_{\#}$. This holds even if no two packets have the same start and destination vertices.*

Proof. We describe a reduction from MONOTONE-NOT-ALL-EQUAL-3-SAT. This is a variant of 3-SAT in which the objective is to find a truth assignment for the variables such that in each clause there is at least one true literal and at least one false literal. Additionally, the formula is monotone, i.e., in each clause there are only positive literals. This problem can be shown to be NP-hard by using a reduction from NOT-ALL-EQUAL-3-SAT [13, p. 259] in which all negative literals $\bar{x}_i$ are replaced by new variables z_i and clauses of the type $(\bar{x}_i \vee z_i \vee z_i)$ are added to the formula.

Now we give an overview of the construction. It is set up such that the formula is satisfiable if and only if the length of an optimal makespan is at most L (for a constant L to be defined later). The paths of the packets are not fixed, however, in order to be able to arrive at their destinations after at most L timesteps they all have to move on shortest paths between their respective start and destination vertices. In particular, if a packet has its start and its destination vertex in the same row there is only one possible way for this packet.

Let $X = \{x_1, x_2, ..., x_k\}$ be the set of variables and let $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ be the set of clauses (each clause is a set of two or three literals). For each variable x_i we introduce two *variable packets* v_i and $\bar{v}_i$. The intuition is that in an optimal

schedule either v_i or $\bar{v}_i$ is delayed once at time $t = 0$ and the other packet is never delayed. Scheduling v_i first (i.e., at time $t = 0$) corresponds to setting x_i true in the variable assignment. Scheduling $\bar{v}_i$ first corresponds to setting x_i false in the variable assignment.

For each clause C_j there are two packets p_j and n_j. Intuitively, p_j ensures that if the makespan of the schedule is at most L there is a truth assignment such that there is a true literal in C_j. Analogously, n_j ensures that there is a false literal in C_j. Both packets need to move $L - 2$ steps up and one step to the right. When moving to the right they have to cross the path of a variable packet. Both of them can move to the right in the row of a variable packet v_i if and only if $x_i \in C_j$. Moreover, p_j can move to the right in the row of v_i only if v_i was scheduled at time $t = 0$ (which corresponds to setting x_i true). Similarly, n_j can move to the right in the row of v_i only if v_i was scheduled at time $t = 1$ (which corresponds to setting x_i false).

Now we describe the construction in detail. Define $L := 4k + 4m - 2$ (recall that k is the number of variables and m is the number of clauses). We place the start and destination vertices on the grid according to Table 1. We will prove later that the optimal overall makespan is at most L if and only if the formula is satisfiable. In addition to the packets already described above for each variable x_i we introduce two *checker packets* c_i and $\bar{c}_i$. The checker packets will ensure that the packet among v_i and $\bar{v}_i$ that started first is never delayed later in the schedule. We also want to block some of the edges for a certain time interval in order to force the p_j and n_j packets to cross the paths of the variable packets. In order to achieve this we introduce sets of packets T_i which we call *the train for the variable* x_i. These packets block parts of a grid row for certain time intervals. (This grid row will be the row below the row of the path of v_i.) Each set T_i contains $8k + 8m$ packets.

Independently from the trains we block certain rows completely for the entire duration of the schedule. In order to block a row r for each column c we introduce a packet with start vertex (r, c) and destination vertex $(r, c - L)$. Since the grid is infinitely large we cannot block it completely using only a finite number of packets. However, it is possible to block all the edges of a row which are important for the rest of the construction using only a finite number of packets. We block all rows r with $2k - L - 2 \leq r \leq 2$ or $2k + 1 \leq r \leq 2k + 2m$. The number of packets needed for this is bounded by a polynomial in the length of the input. At the end of the proof we will discuss what packets of this blocking procedure need to be removed in order to meet the requirement that the start and destination vertices of the packets are unique.

Additionally, we want the packets p_j and n_j only to be able to move to the right in a row of a variable x_i if $x_i \in C_j$. In order to ensure this, for each pair (x_i, C_j) such that x_i is *not* in C_j there is a *filling packet* $f_{i,j}$. Table 1 describes the start and destination vertices for the packets. Figure 6.6 shows how the filling packets work. Figure 6.7 shows a sketch of the whole construction.

Now we want to prove that there is a schedule of length at most L if and only if the formula is satisfiable. Assume that the formula is satisfiable. Then there is a variable assignment which satisfies the formula. We need to describe the paths of the packets and the routing schedule. First of all, all paths of the packets are shortest paths. Thus, each variable packet v_i never leaves row $2i$ and therefore moves on the direct way from $(2i, 2i)$ to $(2i, 2i + L - 1)$. Similarly, a variable packet $\bar{v}_i$ and

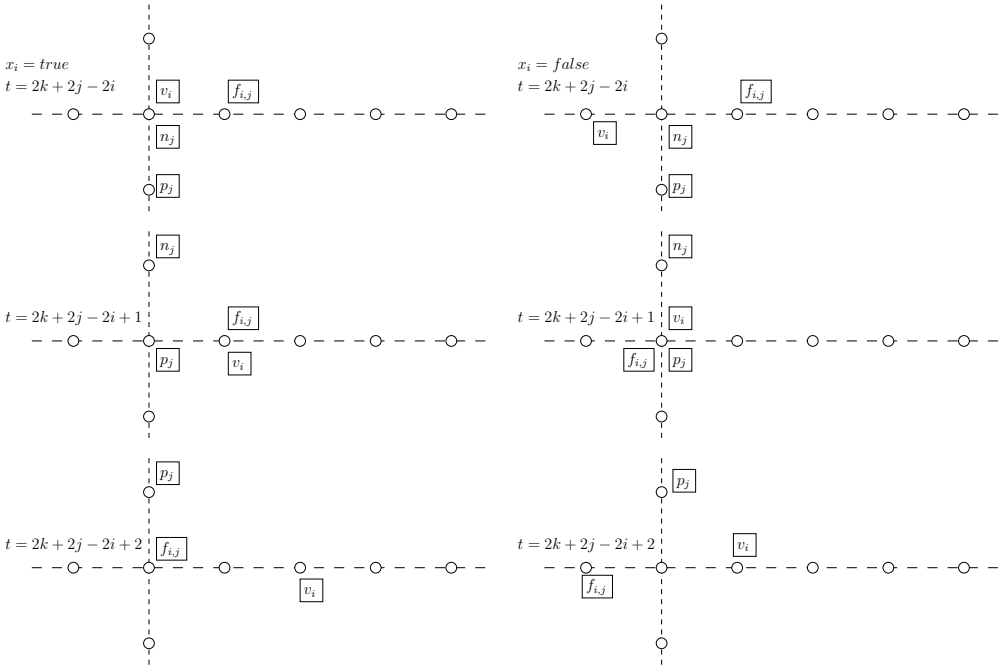

FIGURE 6.6. Depending if x_i is set true (left) or false (right) at times $t = 2k+2j-2i$, $t = 2k+2j-2i+1$, and $t = 2k+2j-2i+2$ we have the distribution of the packets sketched above (assuming that the packets p_j and n_j have not moved to the right before). In order to obtain a schedule with overall length L the packet $f_{i,j}$ can be delayed at most once and the other packets cannot be delayed at all. This forces the packets to be scheduled as sketched.

Packet	Start vertex	Destination vertex
v_i	$(2i, 2i)$	$(2i, 2i + L - 1)$
$\bar{v}_i$	$(2i, 2i + 1)$	$(2i, 2i - L + 2)$
$\bar{c}_i$	$(2i, 2i - L + 3)$	$(2i, 2i + 2)$
c_i	$(2i, 2i + L - 4)$	$(2i, 2i - 3)$
n_j	$(2k + 2j, 2k + 2j)$	$(2k + 2j - L - 2, 2k + 2j + 1)$
p_j	$(2k + 2j + 1, 2k + 2j)$	$(2k + 2j - L + 2, 2k + 2j + 1)$
$f_{i,j}$	$(2i, 4k + 4j - 2i + 1)$	$(2i + 1, 4k + 4j - 2i + 1 - L + 2)$
T_i, pth packet	$(2i + 1, 2i - L + 5 + p)$	$(2i + 1, 2i + 5 + p)$

TABLE 1. Start and destination vertices for construction in Theorem 43

checker packets c_i and $\bar{c}_i$ always stay on row $2i$. Now consider the packets p_j and n_j for a clause C_j. Both take one of the shortest paths from their start to their destination vertex, so we only need to define in which row they take their step to the right. Since the formula is satisfiable there must be at least one literal x_ℓ in C_j which is true and at least one other literal $x_{\bar{\ell}}$ which is false. We define that the

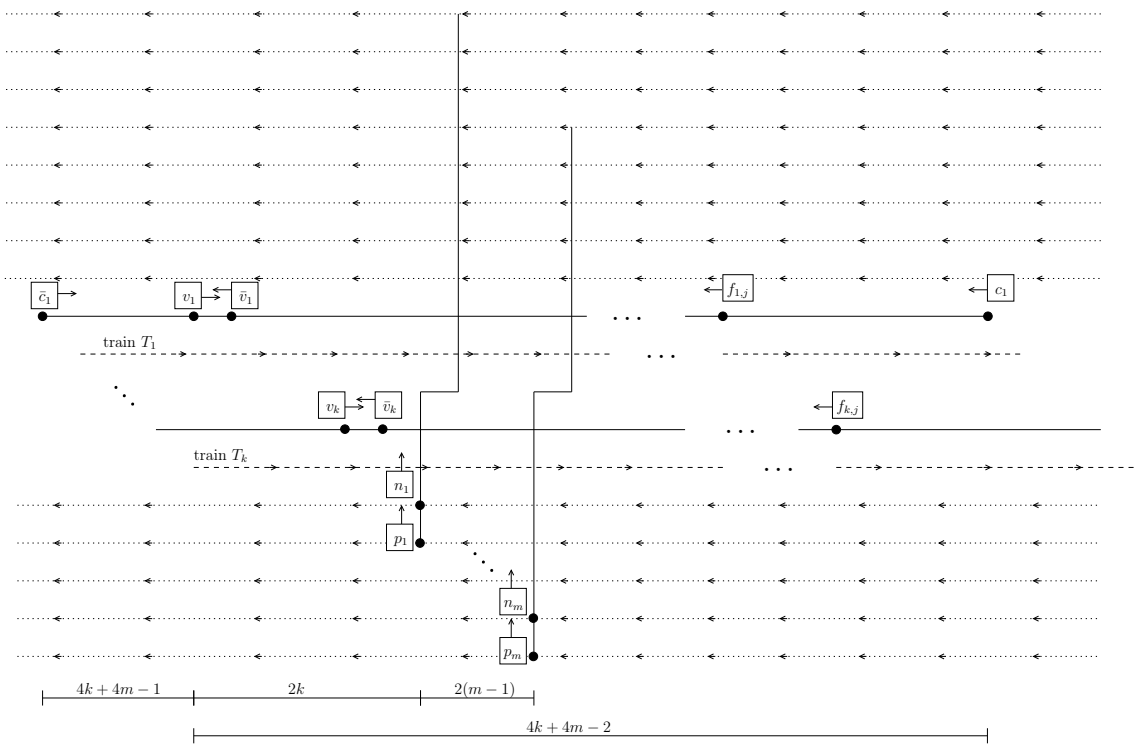

FIGURE 6.7. Sketch of the construction of Theorem 43. The dottet lines denote the blocked rows. All vertices on the dashed line are start vertices of a train packet.

packet n_j takes its step to the right in row $2\bar{\ell}$. For the packet p_j we define that it takes its step to the right in row 2ℓ. For each train T_i all packets never leave row $2i+1$ and move directly all the way to the right. Each filling packet $f_{i,j}$ also takes one of the shortest paths between its start and destination vertex (going all the way to the left and moving one step down). It moves one step down when it would otherwise cause a collision with the checker packet $\bar{c}_i$ which is moving towards it. If the latter never happens, $f_{i,j}$ moves down in its very last step.

Now we describe the schedule of the variable packets: For each variable x_i we schedule v_i at time $t = 0$ and $\bar{v}_i$ at time $t = 1$ if x_i is set true in the variable assignment. If x_i is set false we schedule v_i at time $t = 1$ and $\bar{v}_i$ at time $t = 0$. Table 2 shows the priority ranking of the packets. Whenever there are two packets which need to use an edge at the same time the packet with the higher priority moves first.

Now we show that this leads to a makespan of at most L. Consider the packets v_i and $\bar{v}_i$ for a variable x_i. Each packet can be delayed at most once and therefore will arrive at its destination after at most L steps. Consider the packets c_i and $\bar{c}_i$. If v_i was scheduled at time $t = 0$ then at time $t = 1$ the packets v_i, $\bar{v}_i$ and c_i will be on odd grid columns and $\bar{c}_i$ will be on an even grid column. From the construction of the paths it follows that $\bar{c}_i$ will be delayed once in its path (by $\bar{v}_i$) and will not be delayed by c_i. Thus, c_i and $\bar{c}_i$ will arrive at their destinations after

Priority	Packets
1	v_i/ $\bar{v}_i$ (see text)
2	c_i
3	$\bar{c}_i$
4	n_j
5	p_j
6	$f_{i,j}$
7	packets in T_i

TABLE 2. Packet priorities

at most L timesteps. If v_i was scheduled at time $t = 1$ the proof works similarly. Now consider a packet n_j for a clause C_j and let $2r$ be the row in which n_j moves to the right. From the definition of the path it follows that x_j is set false in the variable assignment. Therefore, v_r was delayed once at time $t = 0$. This implies that n_j arrives at the vertex $(2r, 2k + 2j)$ after $2k + 2j - 2r$ timesteps which is one timestep earlier than v_i. After $2k + 2j - 2r$ timesteps the packets c_r and $\bar{c}_r$ are located at the vertices $(2r, 4r + 2k + 4m - 2j - 2)$ and $(2r, 2j - 2k - 4m + 2)$, respectively, and therefore do not interfere with n_j, since $1 \leq j \leq m$ and $1 \leq r \leq k$. Therefore, the packet n_j reaches its destination after $L - 1$ timesteps.

Now consider a packet p_j for a clause C_j and let $2r'$ be the row in which p_j moves right. From the definition of the paths it holds that $x_{r'}$ was set true in the variable assignment. Therefore, p_j arrives at the vertex $(2r', 2k + 2j)$ after $2k+2j-2r'+1$ timesteps, one timestep after v_j. Similarly as above, after $2k+2j-2r'+1$ timesteps the packets $c_{r'}$ and $\bar{c}_{r'}$ are located at the vertices $(2r', 4r + 2k + 4m - 2j - 3)$ and $(2r', 2j - 2k - 4m + 3)$, respectively, and therefore do not interfere with p_j, since $1 \leq j \leq m$ and $1 \leq r \leq k$. Thus, p_j is never delayed and arrives at its destination after L timesteps.

Now consider a packet $f_{i,j}$. We will show that $f_{i,j}$ can be delayed at most once. In particular, this proves that $f_{i,j}$ cannot be delayed by other filling packets $f_{i',j'}$. The existence of $f_{i,j}$ implies that the clause C_j does not contain the variable x_i. Since the packet $\bar{v}_i$ is delayed at most once (and moves to the left), it cannot delay $f_{i,j}$. The packet v_i can delay $f_{i,j}$ at most once. All packets $v_{i'}$ with $i' \neq i$ cannot interfere with $f_{i,j}$ by construction of the paths. The choice of the path of $f_{i,j}$ implies that all packets $c_{i'}$, $\bar{c}_{i'}$ never delay $f_{i,j}$. Now let $n_{j'}$ be a packet which moves to the right in row $2i$. It moves to the right when it is located at the vertex $(2i, 2k + 2j')$. This happens after $2k + 2j' - 2i$ timesteps. The packet $f_{i,j}$ reaches the vertex $(2i, 2k + 2j')$ after $2k + 4j - 2i - 2j' + 1$ or $2k + 4j - 2i - 2j' + 2$ timesteps (since it might be delayed once). Since C_j does not contain the variable x_i this implies that $j \neq j'$ and therefore $f_{i,j}$ is not being delayed by $n_{j'}$. Similarly, the packet $p_{j'}$ reaches the vertex $(2i, 2k + 2j')$ after $2k + 2j' - 2i + 1$ timesteps and cannot delay $f_{i,j}$. Thus, $f_{i,j}$ arrives at its destination after at most L timesteps. See also Figure 6.6 for a sketch of filling packets passing by other packets.

Now consider a train T_i. The only packets that could cause a collision with the packets in T_i are the filling packets. Assume on the contrary that there is a filling packet $f_{i',j'}$ that needs to use an edge at the same time as a packet $p \in T_i$ and assume that this is the first time in the schedule that a packet of a train is delayed. Assume this happens at time $t = t_0$. Since p is delayed by $f_{i',j'}$ it must hold that

$i = i'$ so the delay must happen when $f_{i',j'}$ is located in row $2i + 1$. Assume that p is located in the vertex $(2i + 1, \bar{j})$ when it is delayed. Since by construction of the paths $f_{i',j'}$ moved down when it otherwise would have been delayed by $\bar{c}_{i'}$ this implies that at time t_0 the package $\bar{c}_{i'}$ must be in a grid column greater or equal $\bar{j}$. But this is a contradiction since this is the first time that a package of T_i was delayed and all packages of T_i started in grid columns greater than the starting grid column of $\bar{c}_i$. Therefore, all train packages reach their destinations after L timesteps.

Now we want to show that the formula is satisfiable if there is a schedule of length at most L. So assume that there is a routing schedule of length at most L. First of all we observe that for each packet the length of its path from start to destination is either $L - 1$ or L. Since the graph is a grid graph this implies that each packet travels on a shortest path from its start vertex to its destination and can be delayed at most once. Thus, for each variable x_i at time $t = 0$ either v_i or $\bar{v}_i$ is delayed by one timestep. We set x_i true if and only if v_i is scheduled at time $t = 0$ and $\bar{v}_i$ is scheduled at time $t = 1$. Now we want to show that this assignment satisfies the formula. Let C_j be a clause and consider the two packets n_j and p_j. From the blocking of the rows we conclude that n_j moves to the right in a row $r = 2\ell$ with $1 \leq \ell \leq k$. This happens at time $t = 2k + 2j - 2\ell$.

We claim that x_ℓ was set false in our assignment and that x_ℓ is a literal in C_j. Assume on the contrary that x_ℓ was set true in our assignment. (Note that this implies that $\bar{v}_\ell$ has to delay the packet $\bar{c}_i$ once.) Then the packets n_j and v_ℓ arrive at the vertex $(2\ell, 2k + 2j)$ at the same time. Thus, either n_j or v_ℓ must have been delayed. If n_j was delayed then it will arrive at the vertex $(2k + 2j - L - 1, 2k + 2j + 1)$ at the same time as p_j and thus one of them will arrive at the destination after at least $L + 1$ timesteps. If v_ℓ was delayed then v_ℓ has to delay c_ℓ later in order to be on time at its destination. Due to the parities of c_ℓ and $\bar{c}_\ell$ one of them has to be delayed twice in total and therefore will not reach its destination before $t = L + 1$. Thus, x_ℓ was set false in our assignment. It remains to show that x_ℓ is a literal in C_j. Assume on the contrary that x_ℓ is not a literal in C_j. Then there exists a filling packet $f_{\ell,j}$ (see Figure 6.6). Thus, after $2k + 2j - 2\ell$ timesteps v_ℓ is located at the vertex $(2\ell, 2k + 2j - 1)$, n_j is at the vertex $(2\ell, 2k + 2j)$ and $f_{\ell,j}$ is at the vertex $(2\ell, 2k + 2j + 1)$. This implies that either v_ℓ or n_j are delayed once or $f_{\ell,j}$ is delayed twice. Each case leads to a contradiction.

Now let r be the row in which the packet p_j moves to the right. This happens at time $t = 2k + 2j - r + 1$. Similarly to the above we can show that r is even, and thus, $r = 2\ell'$ and that $x_{\ell'}$ was set true in our assignment and that $x_{\ell'}$ is a literal in C_j. This reasoning implies that in each clause C_j there is at least one literal which is set true and one literal which is set false. Therefore, the formula is satisfiable.

In the current construction there are packets with the same start and destination vertices: for each packet p_j there is one row blocking packet r_j which has the same start vertex as p_j and one row blocking packet r'_j which has the same destination vertex as p_j. We remove r_j and r'_j from the construction. Since the row blocking packets move to the left the packet p_j does not gain any flexibility from that. We do the same procedure with each packet n_j. Now the start and destination vertices of all packets are unique. □

7. Conclusion

We investigated the complexity of the packet routing problem on the grid. We showed that if the paths are fixed the problem is NP-hard. This holds even if there is only one start vertex and all paths are shortest paths. Note that in the case that the paths are variable, the packet routing problem with one source vertex can be solved in polynomial time as we showed in Section 3.3. It remains open to investigate whether the problem with fixed paths is still NP-hard on the grid if all packets share the same start and destination vertex. We showed that the packet routing problem with variable paths on the unidirectional grid $G_{\#}$ is NP-hard. This holds even if we require the start and destination vertices of the packets to be pairwise different. On general planar graphs there can be no $1+\epsilon$ approximation algorithm for the packet routing problem (with fixed or variable paths) unless $P = NP$ [26]. The special structure of the grid prevented us from ruling out the existence of an approximation scheme in this setting. It remains open to construct a PTAS or to show that it cannot exist unless $P = NP$.

For the setting of unique start and destination vertices we presented an optimal algorithm for the bidirectional grid $\overleftrightarrow{G}_{\#}$. It implies a 2-approximation for the same problem on $G_{\#}$ (which we proved to be NP-hard). The paths chosen by this algorithm are one-bend paths. This might look very simple at first sight, but we showed that choosing the paths in a disadvantageous way can lead to arbitrarily large approximation factors, even if the computed routing schedule which is based on the computed paths is optimal.

For the special case of only one start vertex we gave a $1+\epsilon$ approximation algorithm which simultaneously guarantees an absolute error of eight. For arbitrary given graphs, the packet routing problem with variable paths with a single start vertex can be solved in polynomial time. But in the case of the grid the graph is not part of the input but it is given implicitly. Therefore, the number of vertices in the relevant part of the grid can be exponential in the input length. Thus, the usual methods for solving the problem do not lead to polynomial time algorithms. Even if the grid is part of the input the runtime of our $1+\epsilon$ approximation of $O(f(\epsilon) + k \log k)$ (with k being the number of destination vertices in the instance) is significantly faster. We also gave an optimal algorithm for this setting which runs in $O\left(k^6 \cdot n\right)$ which is strongly polynomial. Note that we can still guarantee polynomial runtime if the number of packets which need to be delivered to each destination vertex s_i is binary coded. It remains open to improve the runtime of this algorithm. Moreover, it uses Vaidya's algorithm [33] as a subroutine and thus it would be interesting to find a purely combinatorial algorithm.

For the packet routing problem with fixed paths on the grid we presented an algorithm with approximation factor $b+1$ where b denotes the maximum number of bends in each path. We gave another algorithm for the same setting which constructs a schedule of length $(b+1)\cdot(C-1)+D$. In comparison, the best known result for a direct schedule guarantees a schedule of length $4 \cdot (b+1)(C-1) + D$ [5]. Thus, it is desirable to find a path selection routine which finds paths whose congestion, dilation and number of bends in each path is bounded by a constant times the length of an optimal schedule. This would yield a constant approximation for the problem. Srinivasan and Teo [31] already gave a constant factor approximation for the general packet routing problem with variable paths. However, they use the algorithm by Leighton et al. [20] for finding the routing

schedule. Therefore, their approximation constant is very large (and not explicitly given in the paper). It would be interesting to find a constant approximation with a smaller constant which can be explicitly calculated.

Acknowledgements. We would like to thank Joachim Reichel, Ronald Koch, and Daniel Plümpe for helpful discussions and numerous suggestions for improving the writeup.

References

[1] M. Adler, S. Khanna, R. Rajaraman, and A. Rosén. Time-constrained scheduling of weighted packets on trees and meshes. *Algorithmica*, 36:123–152, 2003.

[2] M. Adler, R. Sitaraman, A. Rosenberg, and W. Unger. Scheduling time-constrained communication in linear networks. In *Proceedings of the 10th annual ACM symposium on Parallel algorithms and architectures*, pages 269–278, 1998.

[3] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Direct routing on trees. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1998.

[4] F. Meyer auf der Heide and B. Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31, 1999.

[5] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. Spirakis. Direct routing: Algorithms and complexity. *Algorithmica*, 45:45–68, 2006.

[6] C. Busch, M. Magdon-Ismail, and J. Xi. Optimal oblivious path selection on the mesh. *IEEE Transactions on Computers*, 57:660–671, 2008.

[7] W. J. Cook, L. Lovász, and J. Vygen, editors. *Research Trends in Combinatorial Optimization*. Springer, 2009.

[8] M. di Ianni. Efficient delay routing. *Theoretical Computer Science*, 196:131–151, 1998.

[9] L. Fleischer and M. Skutella. Minimum cost flows over time without intermediate storage. In *Proceedings of the 14th Annual Symposium on Discrete Algorithms*, 2003.

[10] L. Fleischer and M. Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36:1600–1630, 2007.

[11] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[12] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[13] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. Freeman NY, 1979.

[14] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 2719:397–409, 2003.

[15] A. Hall, K. Langkau, and M. Skutella. An FPTAS for quickest multicommodity flows with inflow-dependent transit times. *Algorithmica*, 47:299–321, 2007.

[16] B. Hoppe. *Efficient Dynamic Network Flow Algorithms*. PhD thesis, Cornell University, 1995.

[17] B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25:36–62, 2000.

[18] R. Koch, B. Peis, M. Skutella, and A. Wiese. Real-time message routing and scheduling. In S. Naor, editor, *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 5687 of *LNCS*, pages 217–230, 2009.

[19] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-scheduling in $O(congestion + dilation)$ steps. *Combinatorica*, 14:167–186, 1994.

[20] F. T. Leighton, B. M. Maggs, and A. W. Richa. Fast algorithms for finding $O(congestion + dilation)$ packet routing schedules. *Combinatorica*, 19:375–401, 1999.

[21] F. T. Leighton, F. Makedon, and I. G. Tollis. A $2n-2$ step algorithm for routing in an $n \times n$ array with constant size queues. In *Proceedings of the 1st Annual Symposium on Parallel Algorithms and Architectures*, pages 328–335, 1989.

[22] J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. 2004.

[23] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14, 1993.

[24] Y. Mansour and B. Patt-Shamir. Many-to-one packet routing on grids. In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 258–267, 1995.

[25] R. Ostrovsky and Y. Rabani. Universal $O(congestion + dilation + \log^{1+\epsilon} N)$ local control packet switching algorithms. In *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pages 644–653, 1997.

[26] B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. Technical Report 003-2009, Technische Universität Berlin, February 2009.

[27] B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. In E. Bampis and K. Jansen, editors, *Proceedings of the 7th Workshop on Approximation and Online Algorithms*, LNCS. Springer, Berlin, 2010. To appear.

[28] Y. Rabani and É. Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th annual ACM Symposium on Theory of Computing*, pages 366–375. ACM, 1996.

[29] S. Rajasekaran. Randomized algorithms for packet routing on the mesh. Technical Report MS-CIS-91-92, Dept. of Computer and Information Sciences, Univ. of Pennsylvania, Philadelphia, PA, 1991.

[30] I. Spenke. *Complexity and approximation of static k-splittable flows and dynamic grid flows*. PhD thesis, Technische Universität Berlin, 2006.

[31] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30, 2001.

[32] A. Symvonis. Routing on trees. *Information Processing Letters*, 57(4):215 – 223, 1996.

[33] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 338–343, 1989.