THROUGHPUT MAXIMIZATION FOR PERIODIC PACKET SCHEDULING

BRITTA PEIS AND ANDREAS WIESE

Abstract. In the periodic packet routing problem a number of tasks periodically create packets which have to be transported through a network. Due to capacity constraints on the edges, it might not be possible to find a schedule which delivers all packets of all tasks in a feasible way. In this case one aims to find a feasible schedule for as many tasks as possible, or, if weights on the tasks are given, for a subset of tasks of maximal weight. In this paper we investigate this problem on trees and grids with row-column paths. We distinguish between direct schedules (i.e., schedules in which each packet is delayed only in its start vertex) and not necessarily direct schedules. For these settings we present constant factor approximation algorithms, separately for the weighted and the cardinality case.

Our results combine discrete optimization with real-time scheduling. We use new techniques which are specially designed for our problem as well as novel adaptions of existing methods.

1. Introduction

In the *periodic packet routing problem* (formally defined below) an infinite number of packets (created periodically by a set of tasks) has to be sent through a network without violating given capacities on the edges. This problem is well studied (see e.g. [2, 9]) and has numerous applications in theory and practice (e.g., on communication networks with bounded bandwidth).

Networks occurring in practical applications are mostly of rather simple topology such as paths, trees, or grids. Also, in communication networks the transmission links can be used in both directions without interfering with each other. This motivates us to investigate periodic packet routing primarily on bidirected trees and bidirected grids with row-column paths.

In this paper we study the corresponding optimization problem of finding a subset of tasks with maximum weight (w.r.t. given weights on the tasks) such that a feasible schedule for these tasks exists. This problem, which we call the Max-Task-problem, combines discrete optimization with real-time scheduling. For this reason our algorithms partly are based on techniques from both of these two research directions. On the other hand, some of our techniques designed for the Max-Task-problem might as well be extendable for related discrete optimization and scheduling problems.

Beside practical applications, there is also a second reason why we restrict to trees and grids when searching for constant-factor approximations: even on

A *row-column* path moves along the row of its start vertex s_i to the column of its destination vertex t_i, and then along the column of t_i to t_i itself. Such paths have also been studied in [1, 5].

the simple class of chain graphs the MAX-TASK-problem contains the MAXIMUM-INDEPENDENT-SET-problem (see Theorem 27) which is known to be NP-hard to approximate within a factor of $|T|^{1-\epsilon}$ for all $\epsilon > 0$ [13].

1.1. **Periodic Packet Routing.** The periodic packet routing problem is defined as follows: Let $G = (V, E)$ be a graph and p be an integer. Let $T = \{\tau_0, \tau_1, ..., \tau_{|T|-1}\}$ be a set of tasks. Each task $\tau_i = (s_i, t_i, w_i)$ creates a new packet $M_{i,j}$ at timesteps $t = j \cdot p$ for all $j \in \mathbb{N}$. These packets start in the vertex s_i and have to be delivered to t_i along a predefined path P_i. We assume that all packets move simultaneously, it takes one timestep to transfer a packet over an edge and each edge can be used by at most one packet at a time in each direction. In order to stress that the edges can be used in both direction independently we will use later speak of *bidirected* graphs.

We denote by D_i the length of P_i and by $\mathcal{P}$ the set of all paths. The *weight* of a task τ_i is given by a value w_i. The tuple $(G, T, \mathcal{P}, p)$ forms an instance of the *periodic packet routing problem*. We assume that all predefined paths are simple paths. A *schedule* for $(G, T, \mathcal{P}, p)$ is given by a map $task : E \times \{0, 1, ..., p-1\} \to T$: The packets created by a task τ_i are allowed to use an edge e at time t only if $task\,(e, t \bmod p) = \tau_i$. Schedules of this kind were defined as *template schedules* in [2]. A schedule is *feasible* if every packet which is ever created reaches its destination vertex eventually. Note that if in a feasible schedule two packets $M_{i,j}, M_{i',j'}$ are located on the same vertex at the same time this implies that $i \neq i'$ (i.e., the packets were created by different tasks). A schedule is *direct* if each packet which is ever created is delayed only in its start vertex. For ease of notation we say a schedule is *indirect* if it is not necessarily direct.

In this paper we focus on template schedules. Whenever we say that a feasible schedule for a set of tasks exists, we mean that a template schedule exists. Note that in the setting of indirect schedules a feasible (template) schedule for a set of tasks T' exists if and only if no arc is used by more than p tasks in T' [9].

1.2. **Maximum Task Problem.** The MAXIMUM-TASK-problem (or short: MAX-TASK-problem) is given by a tuple $(G, T, \mathcal{P}, p)$. The aim is to find a set $T' \subseteq T$ of maximum weight such that there exists a feasible schedule for $(G, T', \mathcal{P}, p)$. (If the paths of the tasks are given implicitly, e.g., on trees, we will omit the set $\mathcal{P}$ in the definition of the instances.) We distinguish between instances for which we want to find a task selection which allows a direct schedule and instances where we are interested in task selections which allow indirect schedules. W.l.o.g. we assume that $|T| > p$ since if $|T| \leq p$ then we can definitely schedule all tasks. In particular, if $|T| > p$ this implies that p is bounded by a polynomial in the input size. For an instance I we denote by $OPT_{indir}(I)$ a subset of the tasks of I of maximum weight such that there exists an indirect schedule for these tasks. Likewise, $OPT_{dir}(I)$ denotes a subset of tasks with maximum weight which allows a direct schedule. For any set of tasks T we denote by $w(T) := \sum_{\tau_i \in T} w_i$ their total weight.

For certain families of graphs $\mathcal{G}$ we study the *price of directness*. Denote by $\mathcal{I}(\mathcal{G})$ the set of all MAX-TASK-instances on a graph in $\mathcal{G}$. Intuitively, the price of directness measures how much we lose at most on graphs in $\mathcal{G}$ when we require a direct schedule rather than an indirect schedule. Formally, it is defined by $\max_{I \in \mathcal{I}(\mathcal{G})} \frac{w(OPT_{indir}(I))}{w(OPT_{dir}(I))}$.

Graph class	indirect schedules		direct schedules		Complexity (all cases)
	cardinality	weighted	cardinality	weighted	
Bidirected tree	2	$\max\left\{2, 3-\frac{2}{p}\right\}$	$\max\left\{2, 3-\frac{2}{p}\right\}$	3	$MAXSNP$-hard [4]
Bidirected grid	4	$\max\left\{4, 6-\frac{4}{p}\right\}$	$\max\left\{4, 6-\frac{4}{p}\right\}$	6	$MAXSNP$-hard [5]

TABLE 1. Overview of approximation factors for the respective versions of the MAX-TASK-problem.

1.3. **Our Contribution.** We study the MAX-TASK-problem which to our knowledge has not been investigated before. First, we show that on directed trees the problem can be solved optimally in polynomial time. Then we prove that the price of directness on directed trees is 1. On bidirected trees the unweighted case is of the MAX-TASK-problem is already $MAXSNP$-hard (due to the contained EDGE-DISJOINT-PATH-problem [4]). We give a 2-approximation for indirect schedules (based on the algorithm by Garg et al. [6] for the integral multi-commodity-flow-problem on undirected trees) and a $\max\{3-\frac{2}{p}, 2\}$-approximation for direct schedules. Interestingly, the latter is also a $\max\{3-\frac{2}{p}, 2\}$-approximation in comparison to the optimal set of tasks allowing an indirect schedule. In the weighted case we show how to compute a set of tasks T' which allows a direct schedule such that $w(T') \geq w(OPT_{indir})/3$ (and hence $w(T') \geq w(OPT_{dir})/3$). This means that the algorithm which computes T' is a 3-approximation for the weighted MAX-TASK-problem in the setting of direct *and* indirect schedules (even though it always outputs tasks which allow a direct schedule). The algorithm is based on techniques by Erlebach et al. [5]. Then we give a $\max\left\{2, 3-\frac{2}{p}\right\}$-approximation for the setting of indirect schedules, extending the techniques of [5] to a more general setting than independent set problems. Finally, we show that the price of directness on bidirected trees is at least 6/5 and at most 2.

Then we study the problem on bidirected grid graphs. Like in [1, 5] we assume that all paths are row-column-paths. In the unweighted case we obtain a 4-approximation algorithm (setting of indirect schedules) and a $\max\{4, 6-\frac{4}{p}\}$-approximation algorithm (direct schedules). In the weighted setting we obtain an algorithm which is a 6-approximation for the setting of indirect schedules as well as for the setting of direct schedules (like in the weighted case for bidirected trees). Also, for the weighted setting with indirect schedules we obtain a $\max\left\{4, 6-\frac{4}{p}\right\}$-approximation. We show that the price of directness on the grid is between 6/5 and 4. The used techniques for the grid are similar to the ones for the bidirected tree.

Table 1 shows a summary of the approximation factors of our algorithms for the respective settings.

1.4. **Related Work.** The MAX-TASK-problem is equivalent to the maximum EDGE-DISJOINT-PATH-problem if $p = 1$. In [4, 5] Erlebach et al. present algorithms for the latter on bidirected trees and grid graphs with row-column-paths (i.e., each path P_i moves along the row of s_i to the column of t_i and then along the column of t_i to t_i itself). They also show that the problem is $MAXSNP$-hard in these settings. This implies that there can be no PTAS unless $P = NP$. Thus, the MAX-TASK-problem

is $MAXSNP$-hard as well. It is not hard to see that in general graphs (and even in grid graphs with arbitrary paths) the edge-disjoint-path-problem contains the maximum independent set problem and therefore, it is NP-hard to approximate with a factor of $|T|^{1-\epsilon}$ for all $\epsilon > 0$ [13].

In [1] Adler et al. consider the problem of scheduling a maximum number of packets with given release times and deadlines through a network. They consider trees and and grid graphs in which the paths are row-column-paths. They study the cardinality case as well as the weighted case. In [5] Erlebach et al. give improved approximation algorithms for the weighted case such that their respective approximation factors match the factors for the unweighted case given in [1] (factor 3 for trees and factor 6 for grid graphs). All these schedules are direct.

If we are interested in indirect schedules, our problem is a special case of the integral maximum multicommodity flow problem. For undirected trees there is a 2-approximation algorithm for the cardinality case [6] and a 4-approximation for the weighted case [3]. In our case, all edges have the same capacity. If the capacity of each edge is at least 2 there is even a 3-approximation for the weighted case [7]. However, this algorithm cannot be adjusted easily to the setting of bidirected trees.

For the periodic packet routing problem on general graphs, Andrews et al. [2] prove the existence of a schedule which delivers each packet within $O(d_i + 1/r_i)$ steps where d_i denotes the length of its path and r_i denotes the insertion rate of its task ($1/r_i = p$ in our notation). For trees and equal period lengths, Peis et al. [9] give improved bounds on the delivery time of each packet. Also, they show that it is NP-hard to decide whether for a set of tasks a direct schedule exists.

2. Greedy Algorithm

First we study a greedy algorithm for the Max-Task-problem. The algorithm considers the tasks in an arbitrary order and adds a task to the output set if it "fits in". In general this algorithm can perform arbitrarily bad in terms of its approximation factor. However, in the sequel we will use it as a subroutine with a special ordering of the tasks. (Note that we will use it only in the unweighted cases.)

First we discuss the greedy algorithm for direct schedules. Let $I = (G, T, \mathcal{P}, p)$ denote an instance of the unweighted Max-Task-problem. We iterate over the tasks. While we iterate, we define the map $task : E \times \{0, 1, ..., p-1\} \to T \cup \{none\}$. We initialize $task(e, k) = none$ for all e and all k and $GREEDY_{dir}(I) := \emptyset$. In the ith iteration we consider the task τ_i. Denote by e_j the jth edge on P_i. We determine whether there is an offset $o_i \in \{0, 1, ..., p-1\}$ for τ_i such that $task(e_j, (o_i + j) \bmod p) = none$ for all $j = 0, 1, ..., |P_i| - 1$. If there is such an offset o_i then we define $task(e_j, (o_i + j) \bmod p) := \tau_i$ for all $j = 0, 1, ..., |P_i| - 1$ and add τ_i to $GREEDY_{dir}(I)$. Finally, we output $GREEDY_{dir}(I)$.

For indirect schedules, in the ith iteration we check whether for each edge e_j on the path of P_i there is a value $o_{i,j}$ such that $task(e_j, o_{i,j}) = none$. If this is the case then we define $task(e_j, o_{i,j}) := \tau_i$ for all edges e_j on P_i and add τ_i to $GREEDY_{indir}(I)$. We denote the resulting set by $GREEDY_{indir}(I)$.

In general, the greedy algorithm can perform arbitrarily bad as the following proposition shows. Nevertheless, in the sequel we will use the greedy algorithm with a special ordering of the tasks which will allow us to bound its approximation ratio.

Proposition 1. *For any $k > 0$ there is an instance I_k of the* Max-Task*-problem such that $|OPT(I_k)_{dir}| = |OPT(I_k)_{undir}|$ and $|GREEDY(I_k)_{dir}| = |GREEDY(I_k)_{undir}|$ and $|OPT(I_k)_{dir}| / |GREEDY_{dir}(I_k)| \geq k$, where $OPT(I_k)$ denotes an optimal task selection for I_k.*

Proof. Consider an instance on a path with vertices $v_0, v_1, ..., v_k$. We have one task τ_{long} with start vertex v_0 and destination vertex v_k. Also, we have k tasks with start vertex v_i and destination vertex v_{i+1} for $0 \leq i \leq k-1$. We assume that $p = 1$. When we run the greedy algorithm with the task order $\tau_{long}, \tau_1, ..., \tau_k$ then $GREEDY_{dir}(I) = GREEDY_{indir}(I) = \{\tau_{long}\}$. However, the optimal solution consists of the tasks $\tau_1, ..., \tau_k$. Therefore, $|OPT(I_k)| / |GREEDY_{dir}(I_k)| \geq k$. □

3. Trees

We study the Max-Task-problem on trees. First, we show that on directed trees the problem can be solved in polynomial time and we show that the price of directness is 1. Then we study the case of bidirected trees. There, the problem is $MAXSNP$-hard, already in the cardinality case and no matter whether we want to compute a direct or indirect schedule. For the cardinality case we show that with a special ordering of the tasks the greedy algorithm is a 2-approximation for the case of indirect schedules and a $\max\{3 - \frac{2}{p}, 2\}$-approximation for the case of direct schedules. Interestingly, the resulting set of tasks in the direct case is also by at most a factor of $\max\{3 - \frac{2}{p}, 2\}$ smaller than $w(OPT_{indir})$ (i.e., the optimal set of tasks which allows an *indirect* schedule). For the weighted case we show how to compute a set of tasks which allows a direct schedule and which forms a 3-approximation, again not only in comparison to $w(OPT_{dir})$ but also in comparison to $w(OPT_{indir})$. Then we give a $\max\left\{2, 3 - \frac{2}{p}\right\}$-approximation algorithm for the setting of indirect schedules, based on a linear program. Finally, we show that the price of directness in bidirected trees is upper-bounded by 2. A lower bound instance shows that it is at least 5/6.

Throughout this section we make the following assumptions: For the trees of the instances which we solve we define an arbitrary vertex v_r to be the root vertex. For each task τ_i we denote by v_i the vertex on P_i which is closest to v_r. We define the *height* of a task τ_i to be the distance between v_r and v_i. When executing the greedy algorithm we assume that the tasks are ordered descendingly by their height. Since the paths of the tasks are unique we will assume that they are given implicitly.

3.1. Directed Trees.

Before discussing the setting of bidirected trees, we first study the Max-Task-problem on directed trees, i.e., on instances on trees in which each edge has an orientation such that it is used only in the direction given by the orientation. First, we show that whenever for a set of tasks there is an indirect schedule there is also always a direct schedule, i.e., the price of directness on directed trees is 1. Therefore, in the sequel we discuss only the problem of finding a set of tasks which allows an indirect schedule.

We show how to reduce the problem to the minimum-cost-flow-problem. Thus, we can solve it optimally in polynomial time. This can also be seen in the wider context of linear programs on network matrices. Network matrices are uni-modular and linear programs on them can be reduced to the minimum-cost-flow-problem [11].

Now, we study the price of directness for directed trees.

Theorem 2. *Let $I = (G, T, p)$ be an instance of the* Max-Task*-problem on a directed tree and let $T' \subseteq T$ denote a set of tasks for which there is an indirect schedule. Then there also exists a direct schedule for the tasks T'.*

Proof. We use similar techniques as introduced in [8]. Assume that the indirect schedule for T' is given by a map $task_{undir}$. We have that each arc is used by at most p tasks. In [8] it was shown that then there is a path coloring $f : T' \to \{0, 1, ..., p-1\}$ (i.e., with p colors) such that two paths which share an arc are colored with different colors. We define a time-dependent edge-coloring $c : E \times \{0, 1, ..., p-1\} \to \{0, 1, ..., p-1\}$: For an arbitrary edge e^* we define $c(e^*, k) := k$ for $k \in \{0, 1, ..., p-1\}$. The other values of c are obtained from the following two properties:

- For two consecutive arcs $e = (u, v)$ and $e' = (v, w)$ we require that $c(e, i) = c(e', (i+1) \bmod p)$ for $0 \leq i < p$.
- For two adjacent arcs $e = (u, v)$ and $e' = (u, v')$ (or $e = (u, v)$ and $e' = (u', v)$) we require that $c(e, i) = c(e', i)$ for $0 \leq i < p$.

We now define the map $task_{dir} : E \times \{0, 1, ..., p-1\} \to \{0, 1, ..., p-1\}$ as follows: If there is a task $\tau_i \in T_e$ such that $f(\tau_i) = c(e, k)$ we define $task_{dir}(e, k) := \tau_i$. Since f is a valid path coloring there can be at most one such task. If $g(e, k) \notin f(T_e)$ we define $task_{dir}(e, k) := none$. From the first property of c we conclude that the schedule obtained by $task_{dir}$ is a direct schedule. □

Theorem 2 implies the desired bound on the price of directness.

Corollary 3. *The price of directness is on directed trees is 1.*

Now we study how to solve the Max-Task-problem optimally on directed trees. Since the price of directness is 1 in this setting we present an algorithm which ensures only that for the computed set of tasks there is an indirect schedule. Due to Theorem 2 we know that there exists also a direct schedule.

Let $I = (G, T, p)$ be an instance of the weighted Max-Task-problem on a directed tree G. We show how to reduce it to the min-cost-flow-problem. We define a min-cost-flow instance I' as follows. We start with G as the underlying graph and define for each arc e the capacity $c(e) := p$ and the cost $a(e) := 0$. For each task τ_i we introduce an arc $e_i := (t_i, s_i)$ with capacity $c(e_i) := 1$ and cost $a(e_i) := -w_i$. All vertices have zero supply/demand. From the definition of I' it is immediate that a task selection $T' \subseteq T$ with total weight α corresponds to an integral flow in I' with total cost $-\alpha$. To show that both problems are in fact equivalent it remains to prove the following lemma.

Lemma 4. *Let f be an integral flow in I' with total cost $-\alpha$. Then there is a task selection $T' \subseteq T$ in I with total weight α.*

Proof. If $\alpha = 0$ it suffices to define $T' = \emptyset$. So now assume that $\alpha > 0$. Hence, there must be an arc e_i with negative cost such that $f(e_i) = 1$. Since we assumed G to be a directed tree, flow conservation implies that for all edges e on the path between s_i to t_i it has to hold that $f(e) \geq 1$. We add the task τ_i to T' and reduce f by one unit on e_i and on all edges between s_i and t_i. The resulting flow is still integral and has cost $-\alpha + w_i$. To show the claim it remains to find a task selection with total weight $\alpha - w_i$. The above procedure reduces the flow by at least one unit in one edge. Hence, applying it iteratively will eventually result in the zero-flow.

Each task τ_i will be picked at most once (when the edge e_i is considered). Thus, the claim then follows from induction. □

Knowing that I and I' are equivalent yields the following theorem.

Theorem 5. *Let I be an instance of the* Max-Task*-instance on a directed tree. There is a polynomial time algorithm which computes a task selection $T' \subseteq T$ which allows a direct schedule such that $w(T') = w(OPT_{dir}(I)) = w(OPT_{indir}(I))$.*

Proof. In the instance I' of min-cost-flow obtained by the reduction above all capacities and weights are integral. Hence, there are polynomial time algorithms which compute an optimal integral flow for I' (e.g., see [12]). Then the claim follows from Theorem 2 and Lemma 4. □

3.2. **Unweighted Tasks on Bidirected Trees.** We now study the Max-Task-problem on bidirected trees with unweighted tasks. In contrast to the problem on directed trees, this is already $MAXSNP$-hard [4]. For the problem of finding tasks of maximum weight which allow an indirect schedule we show that the greedy algorithm is a 2-approximation. Note that this problem is a special case of the maximum multicommodity integral flow problem. If a direct schedule is desired we show that the greedy algorithm is a $\eta(p)$-approximation in comparison to OPT_{dir} *and* OPT_{indir}, where $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$.

Theorem 6. *Let I be a* Max-Task*-instance on a bidirected tree. Then $|OPT_{indir}(I)| \leq 2 \cdot |GREEDY_{indir}(I)|$.*

Proof. The claim can be shown using the primal-dual scheme as used in [6] for showing a 2-approximation for the maximum integral multicommodity flow problem (IMCF) on undirected trees. In fact, our problem is a special case of the IMCF-problem: each task corresponds to one commodity with source s_i and sink t_i and each arc is given capacity p. To model that each task can be assigned at most once we add an arc with capacity 1 to each s_i-t_i-path.

As for the IMCF-problem on undirected trees, the dual problem is the minimum multicut problem: We are looking for a set of edges S such that each s_i-t_i-path uses at least one of the edges in S. The construction of such a set can be done as described in [6]. Also, it is possible to show that each s_i-t_i-path uses at most two arcs from the constructed set. With the primal-dual scheme this proves an approximation factor of two. □

Now we analyze the greedy algorithm in the setting of direct schedules.

Theorem 7. *Let I be an unweighted* Max-Task*-instance on a bidirected tree. It holds that*

$$|GREEDY_{dir}(I)| \geq \frac{1}{\eta(p)} |OPT_{indir}(T)| \geq \frac{1}{\eta(p)} |OPT_{dir}(T)|$$

with $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$.

Proof. We consider the set $DIFF := OPT_{indir}(T) \backslash GREEDY_{dir}(I)$. For each task $\tau_j \in GREEDY_{dir}(I)$ we introduce a variable β_j. Now let $\tau_i \in DIFF$ be a task. For each possible delay $d \in \{0, 1, ..., p-1\}$ there must be an edge $e_j \in P_i$ and a task $\tau_{i(d)} \in GREEDY_{dir}(I)$ with $i(d) < i$ such that $task(e_j, (d+j) \bmod p) = \tau_{i(d)}$. For each task $\tau_{i(d)}$ with $d \in \{0, 1, ..., p-1\}$ we increase $\beta_{i(d)}$ by $\frac{1}{p}$. We say that

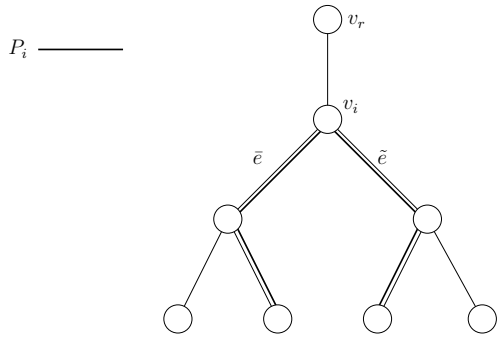

FIGURE 3.1. The path P_i with the edges $\bar{e}$ and $\tilde{e}$ as defined in the proof of Theorem 7.

the tasks $\tau_{i(d)}$ *pays* for the task τ_i. We do this procedure for all tasks $\tau_i \in DIFF$. Note that after this $\sum_i \beta_i = DIFF$.

We define $T_1' := GREEDY_{dir}(I) \cap OPT_{indir}(T)$ and $T_2' := GREEDY_{dir}(I) \setminus OPT_{indir}(T)$. We claim that $\beta_i \leq 2 \cdot \frac{p-1}{p}$ for all tasks $\tau_i \in T_1'$ and $\beta_i \leq 2$ for all tasks $\tau_i \in T_2'$. Let $\tau_i \in GREEDY_{dir}(I)$ and denote by P_i the path of τ_i and by $v_i \in P_i$ the vertex on P_i which is closest to v_r. Denote by $\bar{e}$ and $\tilde{e}$ the edges on P_i which are incident to v_i, see Figure 3.1. Let $\tilde{\tau}$ be a task which τ_i pays for. Let $h(\tau_i)$ denote the height of v_i. Since $h(\tilde{\tau}) \leq h(\tau_i)$ we conclude that $\tilde{\tau}$ either uses $\bar{e}$ or $\tilde{e}$. Since in the $OPT_{indir}(T)$ there can be at most $2p$ such tasks we conclude that $\beta_i \leq 2$. Moreover, if $\tau_i \in OPT_{indir}(T)$ then there can be at most $2(p-1)$ tasks in $DIFF$ which use $\bar{e}$ or $\tilde{e}$. Thus, if $\tau_i \in OPT_{indir}(T)$ then $\beta_i \leq 2 \cdot \frac{p-1}{p}$. We complete the proof by calculating that

$$\begin{aligned}
|OPT_{dir}(T)| &= |DIFF| + |T_1'| \\
&\leq \sum_i \beta_i + |T_1'| \\
&= \sum_{i:\tau_i \in T'_1} \beta_i + |T_1'| + \sum_{i:\tau_i \in T'_2} \beta_i \\
&\leq 2 \cdot \frac{p-1}{p} \cdot |T_1'| + |T_1'| + 2 \cdot |T_2'| \\
&\leq \max\left\{2, 3 - \frac{2}{p}\right\} |GREEDY_{dir}(I)| \\
&= \eta(p) \cdot |GREEDY_{dir}(I)|
\end{aligned}$$

This shows that $|GREEDY_{dir}(I)| \geq \frac{1}{\eta(p)} |OPT_{indir}(T)|$. The fact that $|OPT_{indir}(T)| \geq |OPT_{dir}(T)|$ completes the proof. □

3.3. **Weighted Tasks on Bidirected Tree.** Now we study the weighted MAX-TASK-problem on bidirected trees. Straight-forward examples show that in the weighted setting the greedy algorithm can perform arbitrarily bad. Hence, we need to find more sophisticated methods for solving the problem. We use techniques based on [5]. The key idea is to formulate the problem as an integer program and solve the LP-relaxation. From the obtained (fractional) solution we derive an

integral solution whose weight is at most by a constant factor smaller than the weight of the fractional solution.

First, we present a 3-approximation algorithm for the setting of direct schedules. Then we show that the total weight of the resulting set is also by at most a factor of 3 smaller than $w(OPT_{indir})$ and hence it also works as a 3-approximation in the setting of indirect schedules. Finally, we show a $\max\left\{2, 3-\frac{2}{p}\right\}$-approximation algorithm for the setting of indirect schedules. We also prove that the respective LP has an integrality gap of at most $\max\left\{2, 3-\frac{2}{p}\right\}$.

Let $I = (G, T, p)$ be an instance of the weighted Max-Task-problem on a bidirected tree G in the setting of direct schedules. We reduce the problem to an instance of the weighted maximum independent set problem. We define the graph $G_{MIS} = (V_{MIS}, E_{MIS})$ as follows: for each task τ_i we introduce p vertices $\langle \tau_i, k\rangle$ with $0 \leq k \leq p-1$. A vertex $\langle \tau_i, k\rangle$ corresponds to scheduling the task τ_i such that it uses the first edge on its path at times t with $t \bmod p = k$. We call such a value k the *offset* of τ_i. We connect two vertices $\langle \tau_i, k\rangle$, $\langle \tau_j, \ell\rangle$ by an edge if and only if either

- $\tau_i = \tau_j$ or
- P_i and P_j use an edge in the same direction and if τ_i and τ_j had the offsets k and ℓ their packets would collide.

We assign each vertex $\langle \tau_i, k\rangle$ the weight w_i. Then any solution for weighted maximum independent set on G_{MIS} corresponds to a solution for I with the same weight and vice versa.

Note that the size of G_{MIS} is bounded by a polynomial in the size of I. In order to apply the framework by Erlebach et al. [5] we define the following linear programs LP_I and LP_w. First, we consider the LP-relaxation LP_I of the weighted maximum independent set problem on G_{MIS}. Denote by $\mathcal{C}$ the set of all maximal cliques $\{\langle \tau_{i_1}, k_1\rangle, \langle \tau_{i_2}, k_2\rangle, ..., \langle \tau_{i_m}, k_m\rangle\}$ in G_{MIS} which arise

- because $\tau_{i_1} = \tau_{i_2} = \tau_{i_3} = ... = \tau_{i_m}$ or
- because there is an edge e which is used by each of the tasks $\tau_{i_1}, ..., \tau_{i_m}$ at the same time if they are assigned the offsets $k_1, ..., k_m$

Note that the number of cliques in $\mathcal{C}$ is bounded by a polynomial in the size of G_{MIS}. We define LP_I by

$$
\begin{aligned}
(LP_I)\ \max \quad & \sum_{\langle \tau_i, k\rangle \in V_{MIS}} w_i \cdot x_{i,k} & \\
\text{s.t.} \quad & \sum_{\langle \tau_i, k\rangle \in C} x_{i,k} \leq 1 & \forall C \in \mathcal{C} \\
& 0 \leq x_{i,k} \leq 1 & \forall \langle \tau_i, k\rangle \in V_{MIS}
\end{aligned}
$$

Since the size of LP_I is bounded by a polynomial we can solve it optimally in polynomial time. Let x^* be an optimal solution of LP_I. We now interpret each value $x^*_{i,k}$ as a cost value and define $c_{i,k} := x^*_{i,k}$. We define a linear program LP_w which computes a fractional coloring for the vertices in V_{MIS}. Each vertex $\langle \tau_i, k\rangle \in V_{MIS}$ has to be colored with colors whose total weight is at least $c_{i,k}$. This can be seen as assigning each vertex a set of disjoint intervals of total length $c_{i,k}$ such that the intervals of two adjacent vertices do not intersect. For the definition

of LP_w we denote by $\mathcal{J}$ the set of all independent sets in G_{MIS} (note that a coloring can be understood as a partition into independent sets).

$$\begin{aligned}(LP_w)\ \min & \sum_{J\in\mathcal{J}} y_J \\ \text{s.t.} & \sum_{J\in\mathcal{J}\mid\langle\tau_i,k\rangle\in V_{MIS}} y_J \geq c_{i,k} && \forall\,\langle\tau_i,k\rangle\in V_{MIS} \\ & 0\leq y_J\leq 1 && \forall J\in\mathcal{J}\end{aligned}$$

In ordinary graph coloring, the clique number $\omega(G)$ of a graph G is a lower bound on the number of needed colors. Likewise, we define a fractional clique number $\omega_{\mathcal{C}}\left(G_{MIS},c\right) := \max_{C\in\mathcal{C}(G)}\sum_{\langle\tau_i,k\rangle\in C} c_{i,k}$ which is also a lower bound on the total weight of the needed colors in our setting. After having computed the optimal solution to LP_I, we compute an approximative solution for LP_w as described in the following lemma.

Lemma 8. *There is a polynomial time algorithm which computes a solution y for LP_w with $\sum_{J\in\mathcal{J}} y_J \leq 3\cdot\omega_{\mathcal{C}}\left(G_{MIS},c\right)$.*

Proof. We order the vertices $\langle\tau_i,k\rangle$ ascendingly by the height of their peak vertex v_i. Then we assign each vertex $\langle\tau_i,k\rangle$ (greedily) color intervals of total length $c_{i,k}$ such that the intervals of two adjacent vertices to not intersect. From this we can extract a value y_J for each independent set J. When considering a vertex $\langle\tau_i,k\rangle$ we observe that colors of weight at most $2\cdot\omega_{\mathcal{C}}\left(G_{MIS},c\right)-2c_{i,k}$ cannot be used due to vertices $\langle\tau_j,\ell\rangle$ with $\tau_j\neq\tau_i$ such that τ_j uses one of the edges on P_i adjacent to v_i. Also, at most $\omega_{\mathcal{C}}\left(G_{MIS},c\right)-c_{i,k}$ cannot be used due to vertices $\langle\tau_i,\ell\rangle$ with $\ell\neq k$. All other colors are available. Therefore, colors of total weight $3\cdot\omega_{\mathcal{C}}\left(G_{MIS},c\right)$ suffice for the greedy algorithm. □

Having computed the solution y for LP_w, we output the independent set $J\in\mathcal{J}$ of maximum weight for which $y_J>0$. Note that since y was computed in polynomial time there can be only a polynomial number of such independent sets. Denote by $BT_{dir}(I)$ the set of tasks corresponding to the vertices in J.

Theorem 9. *Let I be a weighted* Max-Task*-instance on a bidirected tree. It holds that $w\left(BT_{dir}(I)\right)\geq\frac{1}{3}w\left(OPT_{dir}(I)\right)$.*

Proof. Let $w\left(x^*\right)$ denote the objective value of the optimal solution x^* of LP_I. The key idea for the proof is that there is an independent set J with $y_J>0$ such that $w(J)\geq w\left(x^*\right)/3$. So now assume on the contrary that all for all $J\in\mathcal{J}$ with $y_J>0$ we have that $w(J)<w\left(x^*\right)/3$. Note that any solution y for LP_w satisfies $\sum_{J\in\mathcal{J}} w(J)\cdot y_J\geq w\left(x^*\right)$. We calculate that

$$w\left(x^*\right)\leq\sum_{J\in\mathcal{J}} w(J)\cdot y_J<\frac{w\left(x^*\right)}{3}\sum_{J\in\mathcal{J}} y_J\leq w\left(x^*\right)\cdot\omega_{\mathcal{C}}\left(G_{MIS},c\right)\leq w\left(x^*\right)$$

which is a contradiction. □

Now we show that $BT_{dir}(I)$ is also a 3-approximation in comparison with the set of tasks with optimal weight which allows an *indirect* schedule. In the proof of Theorem 9 we showed that $w\left(BT_{dir}(I)\right)\geq w\left(x^*\right)/3$ where x^* is the optimal

solution for LP_I. Now we prove that $w(x^*)$ is also an upper bound on the optimal weight obtained by a task selection which allows an indirect schedule. The latter problem can be formulated by an integer program with the following LP-relaxation:

$$\begin{aligned}
(LP'_I)\ \max & \sum_{\tau_i \in T} w_i \cdot x_i & \\
\text{s.t.} & \sum_{\tau_i \in C} x_i \leq p & \forall C \in \mathcal{C} \\
& 0 \leq x_i \leq 1 & \forall \tau_i \in T
\end{aligned}$$

Here, we derive the set of cliques $\mathcal{C}$ by considering every arc e and taking the clique consisting of all tasks which use e. (The original integer program is obtained by additionally requiring $x_i \in \{0,1\}$ for all x_i.) Let x' denote an optimal solution for LP'_I with value $w(x')$. Since LP'_I is a relaxation of the original problem, $w(x')$ is an upper bound on the total weight of tasks which allow an indirect schedule. Now we show how to transform x' to a solution x for LP_I with the same weight. For each variable $x_{i,k}$ with $\tau_i \in T$ and $k \in \{0,1,...,p-1\}$ we define $x_{i,k} := x'_i/p$. Hence, for the optimal value $w(x^*)$ of LP_I we have that $w(x^*) \geq w(x')$. We conclude with the following theorem.

Theorem 10. *Let I be a weighted* Max-Task*-instance on a bidirected tree. It holds that $w(BT_{dir}(I)) \geq \frac{1}{3} w(OPT_{indir}(I))$.*

Proof. Follows from $w(BT_{dir}(I)) \geq w(x^*)/3 \geq w(x')/3 \geq w(OPT_{indir}(I))/3$. □

Above, we introduced the linear program LP'_I which is the LP-relaxation of an integer program which solves the weighted Max-Task-problem on bidirected trees in the setting of indirect schedules. Based on LP'_I, we now present a $\eta(p)$-approximation algorithm for that problem (with $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$). We give a procedure (running in polynomial time) which takes an optimal (fractional) solution x' for LP'_I with weight $w(x')$ and turns it into an integral solution whose weight is at least $\frac{1}{\eta(p)} w(x')$. Hence, this procedure yields a $\eta(p)$-approximation algorithm.

Let I be an instance of the weighted Max-Task-problem on bidirected trees in the setting of indirect schedules. Note that the size of LP'_I is bounded by a polynomial in the length of I. Hence, we can solve LP'_I optimally in polynomial time. Let x' be an optimal solution for LP'_I with weight $w(x')$. If x' is already integral then we have found an optimal solution for our problem which we denote by $BT_{indir}(I)$. Now assume that x' contains at least one fractional entry. Recall that above we used the linear program LP_w to cover the vertices in G_{MIS} (fractionally) with independent sets where the overall aim was to compute an independent set with maximum weight in G_{MIS}. The set $\mathcal{J}$ denoted all independent sets (and hence all valid solutions). Eventually, we picked the independent set with maximum weight which is used in the solution for LP_w.

In the current problem, the overall aim is no longer to compute an independent set but to compute a set of tasks such that each arc is used by at most p tasks. Our strategy is now to cover the tasks (fractionally) with valid solutions. Denote by $\mathcal{J}'$ the set of all valid solutions to our problem. We interpret each value x'_i as a

cost value and define $c_i := x'_i$. Consider the following linear program LP'_w (which takes the role of LP_w in the algorithm for computing $BT_{dir}(I)$).

$$\begin{aligned}(LP'_w)\ \min & \sum_{J\in\mathcal{J}'} y'_J \\ \text{s.t.} & \sum_{J\in\mathcal{J}'|\tau_i\in J} y'_J \geq c_i && \forall \tau_i \in T \\ & 0 \leq y'_J \leq 1 && \forall J \in \mathcal{J}'\end{aligned}$$

In the following lemma we show how to compute a solution y' for LP'_w such that $\sum_{J\in\mathcal{J}'} y'_J \leq \eta(p)$.

Lemma 11. *Assume that the values c_i in LP'_w are constructed from a valid solution x' for LP'_I. Then there is a polynomial time algorithm which computes a solution y' for LP'_w such that $\sum_{J\in\mathcal{J}'} y'_J \leq \max\left\{2, 3-\frac{2}{p}\right\} = \eta(p)$.*

Proof. We interpret the problem as assigning each task a set of disjoint subintervals of $[0,\eta(p))$ with total length c_i. We do this such that for each arc e we have that for each $t \in [0,\eta(p))$ there are at most p task in using e which are assigned intervals containing t. We call this property the *packing property*.

We sort the tasks ascendingly by the height of their peak vertices. We consider the tasks in this order. The first task τ_1 is assigned an arbitrary subinterval of $[0,\eta(p))$ with length $c_1 \leq 1 < \eta(p)$. For induction, consider the ith iteration in which we consider the task τ_i which we want to assign intervals of total length c_i (obeying the packing property). Since we ordered the tasks by the height of their peak vertices, it suffices to ensure the height property in the two arcs $\bar{e}$ and $\tilde{e}$ adjacent to v_i. Denote by $\bar{q}$ and $\tilde{q}$ the total length of the subintervals of $[0,\eta(p))$ which are already used by p tasks from $T_{\bar{e}}$ or $T_{\tilde{e}}$, respectively. Since the values c_i are constructed from a valid solution of LP'_I we know that $\sum_{\tau_j\in T_{\bar{e}}} c_j \leq p$ and $\sum_{\tau_j\in T_{\tilde{e}}} c_j \leq p$, and hence $\bar{q} \leq 1-\frac{c_i}{p}$ and $\tilde{q} \leq 1-\frac{c_i}{p}$. We calculate that $\bar{q}+\tilde{q}+c_i \leq 2+c_i\left(1-\frac{2}{p}\right) \leq \max\left\{2, 3-\frac{2}{p}\right\} = \eta(p)$. Hence, there are always disjoint subintervals of $[0,\eta(p))$ with total length c_i for τ_i such that the packing property is fulfilled. □

Let y' denote the solution obtained for LP'_w by the procedure described in the proof of Lemma 11. Our integral solution for LP'_I is obtained by taking the solution $J \in \mathcal{J}'$ with maximum weight such that $y'_J > 0$. Denote by $BT_{indir}(I)$ the resulting tasks.

Theorem 12. *Let I be an instance of the weighted* Max-Task*-problem on a bidirected tree in the setting of indirect schedules. For the solution $BT_{indir}(I)$ it holds that $w\left(BT_{indir}(I)\right) \geq \frac{1}{\eta(p)} w\left(OPT_{indir}(I)\right)$ with $\eta(p) = \max\left\{2, 3-\frac{2}{p}\right\}$. Moreover, the integrality gap of LP'_I is bounded by $\eta(p)$.*

Proof. If our computed solution x' with weight $w(x')$ for LP'_I is already integral then there is nothing to prove. So now assume that x' contains at least one fractional entry. We show both claims by proving that $w\left(BT_{indir}(I)\right) \geq \frac{1}{\eta(p)} w\left(x'\right)$. Assume on the contrary that $w\left(BT_{indir}(I)\right) < \frac{1}{\eta(p)} w\left(x'\right)$ and consider the solution y' for

LP'_w. We conclude that for all $J \in \mathcal{J}'$ with $y'_J > 0$ we have that $w(J) < \frac{1}{\eta(p)} w(x')$. Note that since y' is a solution for LP'_w we have that $\sum_{J \in \mathcal{J}'} w(J) \cdot y'_J \geq w(x')$. However, this implies that

$$w(x') \leq \sum_{J \in \mathcal{J}'} w(J) \cdot y'_J < \frac{w(x')}{\eta(p)} \sum_{J \in \mathcal{J}'} y'_J \leq w(x')$$

which is a contradiction. □

3.4. **Price of Directness in Bidirected Trees.** Now we analyze the price of directness in the setting of bidirected trees. We show an upper bound of 2 and a lower bound of 6/5.

Theorem 13. *The price of directness for the* Max-Task*-problem on bidirected trees is upper-bounded by* 2.

Proof. Let $I = (G, T, p)$ be a Max-Task-instance on a bidirected tree. We present a procedure which splits the set $OPT_{indir}(I) \subseteq T$ into two sets T^1 and T^2 such that there are direct schedules for T^1 and T^2. Since $w(T^1) + w(T^2) = w(OPT_{indir}(I))$ we have that either $w(T^1) \geq \frac{1}{2} w(OPT_{indir}(I))$ or $w(T^2) \geq \frac{1}{2} w(OPT_{indir}(I))$. This shows that the price of directness on bidirected trees is bounded by 2.

Now we present the mentioned procedure. For both sets T^k we maintain a map $task_k : E \times \{0, 1, ..., p-1\} \to T$. We order the tasks in $OPT_{indir}(I)$ *ascendingly* by the height of their peak vertex. We consider the tasks one by one. In the ith iteration we consider the task $\tau_i \in OPT_{indir}(I)$. Let $e_0, ..., e_{|P_i|-1}$ be the edges on P_i. We try to assign τ_i a start offset d with which it fits into one of the maps $task_k$. We say a value d is *blocked* in a map $task_k$ if there is an edge e_j such that $task_k(e_j, d + j \bmod p) \neq none$. By the order of the tasks we observe that it suffices to check whether a value d is blocked in one of the edges $\bar{e}$ and $\tilde{e}$ incident to the peak vertex v_i of τ_i. There can be in total at most $2p - 2$ task different from τ_i using $\bar{e}$ and $\tilde{e}$. Each such task can block at most one value d in one of the maps $task_k$. We conclude that there is always one value d which is *not* blocked in $task_1$ or $task_2$. Let $task_k$ be the map in which a value d is not blocked. We then define $task_k(e_j, d + j \bmod p) := \tau_i$ for all edges $e_j \in P_i$ and we assign τ_i to the set T^k. This procedure defines the sets T^1 and T^2. □

Now we show a lower bound on the price of directness on bidirected trees of 6/5.

Proposition 14. *There is an instance $\bar{I}$ of the* Max-Task*-problem on a bidirected tree such that $OPT_{indir}(\bar{I})/OPT_{dir}(\bar{I}) = 6/5$.*

Proof. First we describe the instance $\bar{I}$. The underlying (undirected) tree has the vertices $v_0, v_1, v_2, v_3, v_4, v_5$ and the edges $\{v_0, v_2\}, \{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}$. We have six tasks of unit weight: $\tau_1 = (v_5, v_4)$, $\tau_2 = \tau_3 = (v_5, v_1)$, $\tau_4 = (v_0, v_1)$, and $\tau_5 = \tau_6 = (v_0, v_4)$. See Figure 3.2 for a sketch. We define the period length p by $p := 3$. We observe that each arc is used by at most three tasks and hence $OPT_{indir}(\bar{I}) = 6$.

Now we want to show that $OPT_{dir}(\bar{I}) \leq 5$. Assume on the contrary that $OPT_{dir}(\bar{I}) = 6$. This implies that there is a direct schedule for the six tasks given by a map $task_{dir} : E \times \{0, 1, 2\} \to T$. W.l.o.g. we can assume that $task_{dir}((v_5, v_3), 0) = \tau_1$. This implies that $\{task_{dir}((v_5, v_3), 1), task_{dir}((v_5, v_3), 2)\} = \{\tau_2, \tau_3\}$ and hence $\{task_{dir}((v_2, v_1), 0), task_{dir}((v_2, v_1), 1)\} = \{\tau_2, \tau_3\}$. Thus, we have that

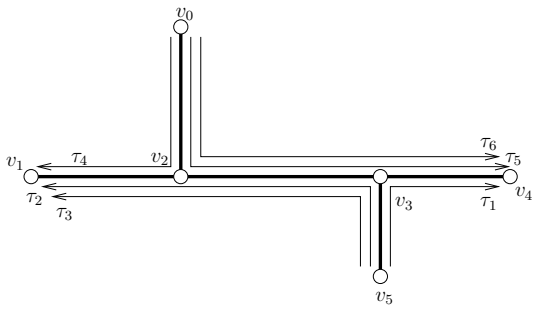

FIGURE 3.2. The instance $\bar{I}$ with $OPT_{indir}(\bar{I})/OPT_{dir}(\bar{I}) = 6/5$ (described in the proof of Proposition 14).

$task_{dir}((v_0, v_2), 1) = \tau_4$ and hence $\{task_{dir}((v_0, v_2), 0), task_{dir}((v_0, v_2), 2)\} = \{\tau_5, \tau_6\}$. However, this implies that either $task_{dir}((v_3, v_4), 1) = \tau_5$ or $task_{dir}((v_3, v_4), 1) = \tau_6$ which contradicts that $task_{dir}((v_3, v_4), 1) = \tau_1$ (which is implied by $task_{dir}((v_5, v_3), 0) = \tau_1$). We conclude that $OPT_{dir}(\bar{I}) \leq 5$ (and with the above procedure a schedule with five tasks can be found by omitting τ_5 or τ_6). □

4. Bidirected Grid Graph

We present our algorithms for the Max-Task-problem on bidirected grid graphs. We assume that all paths are row-column-paths, i.e., all paths move along the row of s_i to the column of t_i and then along the column of t_i to t_i itself. In this setting, we lose a factor of at most 2 if we restrict either to tasks which move left and up or right and down or to tasks which move right and up or left and down. In either case we can split the set of tasks into another two subsets which can be handled separately (with all tasks moving in the same direction). For the Max-Task-problem on such a subset we can use similar methods as for the Max-Task-problem on a bidirected tree. Hence, our algorithms on the grid achieve approximation ratios which are only by a factor 2 worse than the respective algorithms on bidirected trees. We also show that the price of directness in bidirected grids is bounded from above by 4 and bounded from below by 6/5.

Let $\overleftrightarrow{G}_{\#} = \left(V_{\#}, \overleftrightarrow{E}_{\#}\right)$ denote the infinite bidirected grid graph, i.e., $V_{\#} = \{v_{i,j} | i, j \in \mathbb{Z}\}$ and $\overleftrightarrow{E}_{\#} = \{\{v_{i,j}, v_{i',j'}\}, \{v_{i',j'}, v_{i,j}\} \,|\, |i - i'| + |j - j'| = 1\}$. As general notation, for a given instance $\left(\overleftrightarrow{G}_{\#}, T, p\right)$ of the problem, we split the tasks in the sets T_{lu}, T_{ru}, T_{ld}, T_{rd}. The set T_{lu} contains the tasks which move to the left and then up, the set T_{rd} the tasks which move to the right and then down, etc. Note that the tasks in the sets T_{lu} and T_{rd} do not interfere with each other, similarly the tasks in the sets T_{ru} and T_{ld} do not interfere with each other.

For our algorithms we will present subroutines which compute an approximative solution $ALG(T_{ru}) \subseteq T_{ru}$ for the set T_{ru}. Let $OPT(T_{ru}) \subseteq T_{ru}$ denote a feasible task selection from T_{ru} with maximum weight. The following lemma shows how these subroutines yield algorithms for the entire problem.

Lemma 15. *Assume there is a polynomial time algorithm ALG which computes a set $ALG(T_{ru}) \subseteq T_{ru}$ with $w(ALG(T_{ru})) \geq \frac{1}{\alpha} w(OPT(T_{ru}))$ for any set of tasks*

T_{ru} on the bidirected grid whose paths move to the right and then up. Then there is a polynomial time algorithm which computes a task selection $ALG(T) \subseteq T$ with $w(ALG(T)) \geq \frac{1}{2\alpha} w(OPT(T))$ for any instance $I = \left(\overleftrightarrow{G}_{\#}, T, p\right)$ of the Max-Task*-problem on the bidirected grid.*

Proof. Due to symmetry, ALG can also be applied to the sets T_{rd}, T_{lu}, and T_{ld}. We output the set with maximum weight among the sets $ALG(T_{ru}) \cup ALG(T_{ld})$ and $ALG(T_{rd}) \cup ALG(T_{lu})$. Let $OPT_{\diagdown}(T) := OPT(T) \cap (T_{lu} \cup T_{rd})$ and $OPT_{\diagup}(T) := OPT(T) \cap (T_{ld} \cup T_{ru})$. W.l.o.g. we assume that $w\left(OPT_{\diagdown}(T)\right) \geq \frac{1}{2} w(OPT(T))$. Then we conclude

$$\begin{aligned} w(ALG(T)) &\geq w(ALG(T_{rd}) \cup ALG(T_{lu})) \\ &= w(ALG(T_{rd})) + w(ALG(T_{lu})) \\ &\geq \frac{1}{\alpha} \cdot w(OPT(T_{rd})) + \frac{1}{\alpha} \cdot w(OPT(T_{lu})) \\ &\geq \frac{1}{\alpha} \cdot w\left(OPT_{\diagdown}(T)\right) \\ &\geq \frac{1}{2\alpha} \cdot w(OPT(T)) \end{aligned}$$

□

In the remainder of this section we present subroutines which compute subsets of T_{ru} with high weight for the different scenarios (weighted/unweighted, direct/indirect schedules). We assume the following ordering of the tasks in T_{ru}: For each task $\tau_i \in T_{ru}$ we define v_i to be the vertex at the bend of the path. If there is no bend we define $v_i := t_i$ if P_i moves only right and $v_i := s_i$ if P_i moves only up. We denote by $e_{i,1}$ and $e_{i,2}$ the edges of P_i which use v_i. We assume that the grid columns increase when moving to the right and the grid rows increase when moving down. We sort the tasks ascendingly by the grid column of their bend vertex. Ties are broken by sorting tasks whose bend vertex has the same grid column ascendingly by the grid row of their bend vertex. This ordering will be important for the greedy algorithm which we will use later as a subroutine.

4.1. **Unweighted Tasks on the Grid.** In this section we study the cardinality case on grid graphs. First, we consider the setting of indirect schedules. We show that for the tasks T_{ru} the greedy algorithm is a 2-approximation algorithm.

Lemma 16. *Let $I = \left(\overleftrightarrow{G}_{\#}, T, p\right)$ be an instance of the unweighted* Max-Task*-problem on the grid. Then $|GREEDY_{indir}(T_{ru})| \geq \frac{1}{2} |OPT_{indir}(T_{ru})|$.*

Proof. For a task τ_i denote by $P_{i,1}$ the part of P_i from s_i to v_i and by $P_{i,2}$ the part of P_i from v_i to t_i. We define a map $task_{GRDY} : GREEDY_{indir}(T_{ru}) \times E \to \{0, 1, ..., p-1, none\}$ with the following property:

- $task_{GRDY}(\tau_i, e) \neq none$ for all edges e which are on the path P_i
- $task_{GRDY}(\tau_i, e) = none$ for all edges e which are not on the path P_i
- $task_{GRDY}(\tau_i, e) = task_{GRDY}(\tau_i, e')$ for all two edges e, e' which lie on $P_{i,1}$
- $task_{GRDY}(\tau_i, e) = task_{GRDY}(\tau_i, e')$ for all two edges e, e' which lie on $P_{i,2}$

Computing values $task_{GRDY}(\tau_i, e)$ with these properties reduces to path coloring on interval graphs. Then, we define a map $task_{OPT} : OPT(T_{ru}) \times E \rightarrow \{0, 1, ..., p-1, none\}$ with the following properties:

- $task_{OPT}(\tau_i, e) = task_{GRDY}(\tau_i, e)$ for all tasks τ_i with $\tau_i \in OPT(T_{ru}) \cap GREEDY_{indir}(T_{ru})$ and all edges e
- $task_{OPT}(\tau_i, e) \neq none$ for all edges e which are on the path P_i
- $task_{OPT}(\tau_i, e) = none$ for all edges e which are not on the path P_i

Such a map clearly exists. In $task_{OPT}$ we say a *bend* occurs if there is a tuple (τ_i, v), consisting of a task τ_i, a vertex v, and two edges $e = (u, v)$ and $e' = (v, w)$ on P_i in the same row or in the same column such that $task_{OPT}(\tau_i, e) \neq task_{OPT}(\tau_i, e')$. In Figure 4.1 (left) task τ_i has a bend on the vertex v^*.

We denote by $cost(r)$ the sum of the grid columns of the bends which occur in a row r. Formally, let $bend(r)$ denote the set of bends in grid row r. We define

$$cost(r) := \sum_{(\tau,(r,i))\in bend(r)} i$$

We define $cost(c)$ similarly for the bends which occur in a column c. W.l.o.g. we assume that $task_{OPT}$ is a map with the above properties which has the minimum number of bends and among these maps one which maximizes the values $cost(r)$ and $cost(c)$ for each grid row r and each grid column c. Note that this can be achieved simultaneously for all grid rows and columns since the bends in all rows and columns are independent of each other.

Now we establish a connection between the tasks in $T_1 := OPT(T_{ru}) \backslash GREEDY_{indir}(T_{ru})$ and $T_2 := GREEDY_{indir}(T_{ru}) \setminus OPT(T_{ru})$. For each task $\tau_i \in T_1$ we find a task $\tau_j \in T_2$ which – informally speaking – prevents τ_i from being added to $GREEDY_{indir}(T_{ru})$. We assign a task $\tau_i \in T_1$ to a task $\tau_j \in T_2$ if P_i uses $e_{j,1}$ or $e_{j,2}$ and $task_{OPT}(\tau_i, e_{j,1}) = task_{GRDY}(\tau_j, e_{j,1})$ or $task_{OPT}(\tau_i, e_{j,2}) = task_{GRDY}(\tau_j, e_{j,2})$, respectively. Clearly, each task $\tau_j \in T_2$ has at most two tasks from T_1 assigned to it. If now each task $\tau_i \in T_1$ is assigned to a task $\tau_j \in T_2$ then $|T_1| \leq 2 \cdot |T_2|$ and hence

$$\begin{aligned} |OPT(T_{ru})| &= |OPT(T_{ru}) \setminus GREEDY_{indir}(T_{ru})| + |GREEDY_{indir}(T_{ru}) \cap OPT(T_{ru})| \\ &\leq 2 \cdot |GREEDY_{indir}(T_{ru}) \setminus OPT(T_{ru})| + |GREEDY_{indir}(T_{ru}) \cap OPT(T_{ru})| \\ &\leq 2 \cdot |GREEDY_{indir}(T_{ru})| \end{aligned}$$

In the remainder of this proof we show that each task $\tau_i \in T_1$ is indeed assigned to a task in T_2. Assume on the contrary that there is a task $\tau_i \in T_1$ which has not been assigned to any task in T_2. Denote by $GRDY_i \subseteq T_2$ all tasks in T_2 which were assigned to $GREEDY_{indir}(T_{ru})$ before iteration i. Since $\tau_i \in T_1$ there must be a task $\tau_j \in GRDY_i$ and an edge e_j such that $task_{OPT}(\tau_i, e_j) = task_{GRDY}(\tau_j, e_j)$ (since otherwise the algorithm would have assigned τ_i to $GREEDY_{indir}(T_{ru})$). W.l.o.g. we assume that e_j is a horizontal edge in row r. First we remark that P_i must use the edge $e_{j,1}$, since otherwise $\tau_j \notin GRDY_i$. According to the definition of $task_{GRDY}$ we observe that τ_i must have a bend between e_j and $e_{j,1}$ (since otherwise we would have assigned τ_i to τ_j). Let $v^* = (r, i^*)$ be the vertex of this bend. We show now that we can change $task_{OPT}$ such that we either save one bend or increase $cost(r)$ while keeping the total number of bends unchanged. This will yield a contradiction.

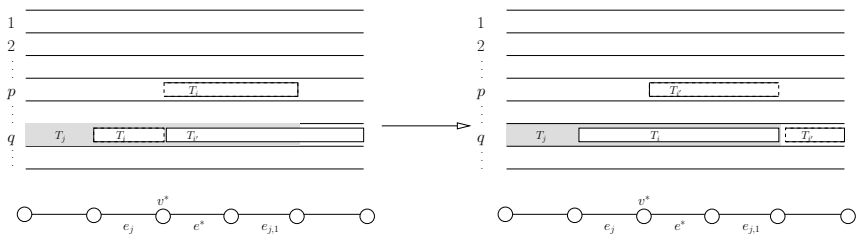

FIGURE 4.1. The figure depicts the edges for which the tasks τ_i, $\tau_{i'}$, and τ_j are assigned the values p and q in the maps $task_{OPT}$, $task'_{OPT}$, and $task_{GRDY}$. The gray area indicates the edges for which τ_j was assigned the value q in $task_{GRDY}$. The white bars on the left indicate the edges for which the tasks τ_i and $\tau_{i'}$ were assigned the values p and q in $task_{OPT}$. The white bars on the right show the edges for which the tasks τ_i and $\tau_{i'}$ were assigned the values p and q in $task'_{OPT}$. Note that on the left the task τ_i has a bend on the vertex v^* and on the right this bend was exchanged to a bend of $\tau_{i'}$ on a vertex further on the right.

Let $e^* = ((r, i^*), (r, i^* + 1))$. Let $\tau_{i'}$ denote the task such that $task_{OPT}(\tau_{i'}, e^*) = task_{OPT}(\tau_i, e_j)$ (if there is no such task $\tau_{i'}$ then $cost(r)$ can be increased by swapping $task_{OPT}(\tau_i, e^*)$ and $task_{OPT}(\tau_{i'}, e^*)$ while not changing the number of bends).

Denote by $P^*_{i',1} \subseteq P_{i',1}$ the largest subpart of $P_{i',1}$ which uses e^* and which does not contain a bend of $\tau_{i'}$. Similarly, denote by $P^*_{i,1} \subseteq P_{i,1}$ the largest subpart of $P_{i,1}$ which uses e^* and which does not contain a bend of τ_i. Let $P^* := P^*_{i,1} \cap P^*_{i',1}$. For all edges $e \in P^*$ we now swap $task_{OPT}(\tau_i, e)$ and $task_{OPT}(\tau_{i'}, e)$: Let $p := task_{OPT}(\tau_i, e^*)$ and $q := task_{OPT}(\tau_{i'}, e^*)$. We define a new map $task'$ by $task'_{OPT}(\tau_{i'}, e) := p$ and $task'_{OPT}(\tau_i, e) := q$ for all edges $e \in P^*$. For all other tasks τ and all other edges e we define $task'_{OPT}(\tau, e) = task_{OPT}(\tau, e)$ (see Figure 4.1). We define $cost'(r)$ based on bends of $task'_{OPT}$ (like $cost(r)$ based on $task_{OPT}$). We observe that $task'_{OPT}$ does not have more bends than $task_{OPT}$ and $cost'(r) > cost(r)$. This contradicts that $task_{OPT}$ is a map which maximizes $cost(r)$ in each grid row r while minimizing the total number of bends. □

Theorem 17. *There is a 4-approximation algorithm for the unweighted* MAX-TASK*-problem on the bidirected grid in the setting of indirect schedules.*

Proof. Follows from Lemmas 15 and 16. □

Now we study the setting of direct schedules on the bidirected grid (unweighted case). We prove an approximation guarantee for the greedy algorithm in this case.

Lemma 18. *Let* $I = \left(\overleftrightarrow{G}_{\#}, T, p\right)$ *be an instance of the unweighted* MAX-TASK*-problem on the bidirected grid. It holds that* $|GREEDY_{dir}(T_{ru})| \geq \frac{1}{\eta(p)} \cdot |OPT_{dir}(T_{ru})|$ *where* $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$.

Proof. The analysis is similar to the analysis used in Theorem 7. We define $DIFF := OPT_{dir}(T_{ru}) \setminus GREEDY_{dir}(T_{ru})$. Now consider a task $\tau_i \in DIFF$.

Since $\tau_i \notin GREEDY_{dir}(T_{ru})$ the task τ_i it was considered in the ith iteration but it was not added to $GREEDY_{dir}(T_{ru})$.

For each task $\tau_k \in GREEDY_{dir}(T_{ru})$ we introduce a variable β_k. Now let $\tau_i \in DIFF$ be a task. For each initial waiting time $w \in \{0,1,...,p-1\}$ for τ_i there must be an edge $e_j \in P_i$ and a task $\tau_{i(w)}$ with $i(w) < i$ such that $task(e_j, (w+j) \bmod p) = \tau_{i(w)}$. For each task $\tau_{i(w)}$ with $w \in \{0,1,...,p-1\}$ we increase $\beta_{i(w)}$ by $\frac{1}{p}$. We say that the tasks $\tau_{i(w)}$ *pays* for the task τ_i. We do this procedure for all tasks $\tau_i \in DIFF$. Note that after this $\sum_i \beta_i = DIFF$.

We define $T_1 := GREEDY_{dir}(T_{ru}) \cap OPT_{dir}(T_{ru})$ and $T_2 := GREEDY_{dir}(T_{ru}) \setminus OPT_{dir}(T_{ru})$. We claim that $\beta_i \leq 2 \cdot \frac{p-1}{p}$ for all tasks $\tau_i \in T_1$ and $\beta_i \leq 2$ for all tasks $\tau_i \in T_2$. Let $\tau_i \in GREEDY_{dir}(T_{ru})$ and denote by P_i the path of τ_i. Denote by e_1 and e_2 the edges on P_i which are incident to the bend vertex of P_i (the case where P_i has no bend can be handled similarly). Let $\tilde{\tau}$ be a task which τ_i pays for. This implies that $\tilde{\tau}$ was considered after τ_i and thus $\tilde{\tau}$ uses either e_1 or e_2. Since in $OPT_{dir}(T_{ru})$ there can be at most $2p$ such tasks we conclude that $\beta_i \leq 2$. Moreover, if $\tau_i \in OPT_{dir}(T_{ru})$ then there can be at most $2(p-1)$ tasks in $DIFF$ which use e_1 or e_2. Thus, if $\tau_i \in OPT_{dir}(T_{ru})$ then $\beta_i \leq 2 \cdot \frac{p-1}{p}$. We complete the proof by calculating that

$$\begin{aligned}
|OPT_{dir}(T)| &= |DIFF| + |T_1| \\
&\leq \sum_i \beta_i + |T_1| \\
&= \sum_{i:\tau_i \in T'_1} \beta_i + |T_1| + \sum_{i:\tau_i \in T'_2} \beta_i \\
&\leq 2 \cdot \frac{p-1}{p} \cdot |T_1| + |T_1| + 2 \cdot |T_2| \\
&\leq \max\left\{2, 1 + 2 \cdot \frac{p-1}{p}\right\} |GREEDY_{dir}(T_{ru})| \\
&= \eta(p) \cdot |GREEDY_{dir}(T_{ru})|
\end{aligned}$$

This shows that $|GREEDY_{dir}(T_{ru})| \geq \frac{1}{\eta(p)} |OPT_{dir}(T)|$. □

Now we can prove the following theorem.

Theorem 19. *Let $I = \left(\overleftrightarrow{G}_{\#}, T, p\right)$ be an instance of the unweighted* Max-Task- *problem on the bidirected grid. There is a polynomial time algorithm which computes a set of tasks $ALG(T) \subseteq T$ for which there exists a direct schedule and $|ALG(T)| \geq \frac{1}{2\eta(p)} |OPT(T)|$ where $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$.*

Proof. The existence of the direct schedule for the tasks in $ALG(T)$ follows from the map *task* as defined in the greedy algorithm. Then the claim follows from Lemmas 15 and 18. □

The analysis in Lemma 18 is the base of the approximation ratio given in Theorem 19. We now show with an example that this analysis is tight by giving a suitable family of examples. Fix a period length p. We consider the following instance with $3p^2 - 2p$ tasks. For each tuple $(i,j) \in \{1,...,p\}^2$ there is a task $\tau_{i,j}$ with path $((i,j),(i,j+1),(i-1,j+1))$. Denote by T_1 the set of these tasks. Also, for each

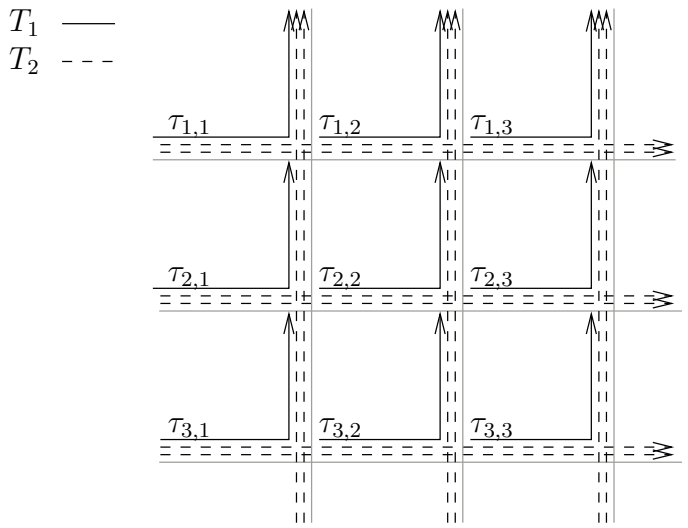

FIGURE 4.2. The tight example instance for Lemma 18 for the case $p = 3$. When all tasks in T_1 are assigned offset 0 then no task in T_2 can be scheduled in a direct schedule. However, assigning each task $\tau_{i,j} \in T_1$ the offset $(i + j) \bmod p$ allows to schedule all tasks in a direct schedule.

$i \in \{1, ..., p\}$ there are $p - 1$ identical tasks with path $((p + 1, i + 1), ..., (0, i + 1))$ and $p - 1$ identical tasks with path $((i, 1), ..., (i, p + 2))$, denoted by T_2 (see Figure 4.2). Now the greedy algorithm could choose all tasks $\tau_i \in T_1$ with initial delay $d_i = 0$. Then, no task in T_2 could be chosen anymore since each possible delay is blocked by some task in T_1. However, it is possible to schedule all tasks. One needs to assign each task $\tau_{i,j} \in T_1$ the initial delay $(i + j) \bmod p$. Then for the tasks in T_2 suitable offsets can be found greedily. Hence, we have that

$$\frac{|GREEDY_{dir}\,(T_1 \cup T_2)|}{|OPT_{dir}\,(T_1 \cup T_2)|} \geq \frac{|T_1| + |T_2|}{|T_1|} = \frac{p^2 + 2p(p-1)}{p^2} = 3 + \frac{2}{p}$$

4.2. **Weighted Tasks on the Grid.** Now we study the weighted MAX-TASK-problem on the grid. We employ similar methods as for the weighted setting on the bidirected tree.

First, we study the case of direct schedules. We formulate the problem as a weighted maximum independent set problem on a conflict graph G_{MIS}. For each task τ_i we introduce p vertices $\langle \tau_i, k \rangle$ with $0 \leq k < p$. A vertex $\langle \tau_i, k \rangle$ corresponds to scheduling the task τ_i to use the first edge on its path at time k. We connect two vertices $\langle \tau_i, k \rangle$, $\langle \tau_j, \ell \rangle$ by an edge if and only if either

- $\tau_i = \tau_j$ or
- P_i and P_j use an edge in the same direction and if τ_i and τ_j had the initial offsets k and ℓ, respectively, their packets would collide.

We assign each vertex $\langle \tau_i, k \rangle$ the weight w_i. Then any solution for weighted maximum independent set on G_{MIS} corresponds to a solution for I with the same weight and vice versa. The linear programs LP_I and LP_w are defined as in Section 3.3. In

our algorithm, we first solve the linear program LP_I optimally, obtaining a solution x^*. We define a cost value $c_{i,k}$ by $c_{i,k} := x^*_{i,k}$. Then we compute an approximative solution for LP_w.

Lemma 20. *There is a polynomial time algorithm which computes a solution y for LP_w with $\sum_{J\in\mathcal{J}} y_J \leq 3 \cdot \omega_{\mathcal{C}}(G_{MIS}, c)$.*

Proof. The algorithm works similarly as the respective algorithm for the linear program LP_w for the TCSWP-problem on grids in [5]: We order the vertices $\langle \tau_i, k\rangle$ first decreasingly by the column of the bend vertex of τ_i. Ties are broken by ordering the respective vertices decreasingly by the row of the bend vertex of τ_i. We consider the vertices in this order. We assign each vertex $\langle \tau_i, k\rangle$ (greedily) disjoint intervals of total length $c_{i,k}$ such that the intervals of two adjacent vertices do not intersect. Like in Lemma 8 we can prove that colors of total weight $3 \cdot \omega_{\mathcal{C}}(G_{MIS}, c)$ suffice. From this we can extract a value y_J for each independent set J. □

Finally, our algorithm outputs the independent set $J \in \mathcal{J}$ with maximum weight among the sets J with $y_J > 0$. Denote by $BG_{dir}(T_{ru})$ the set of tasks corresponding to the vertices in J.

Lemma 21. *Let I be an instance of the weighted* Max-Task*-problem on the bidirected grid. It holds that $w(BG_{dir}(T_{ru})) \geq \frac{1}{3} w(OPT_{dir}(T_{ru}))$.*

Proof. Can be shown similarly as in Theorem 9. □

We define $BG_{dir}(I)$ to be the set of tasks with maximal weight among the sets $BG_{dir}(T_{ru}) \cup BG_{dir}(T_{ld})$ and $BG_{dir}(T_{rd}) \cup BG_{dir}(T_{lu})$.

Theorem 22. *Let I be an instance of the weighted* Max-Task*-problem on the bidirected grid. It holds that $w(BG_{dir}(I)) \geq \frac{1}{6} w(OPT_{dir}(I))$.*

Proof. Follows from Lemmas 15 and 21. □

Similarly as for the case of bidirected trees, we can show that $BG_{dir}(I)$ is by at most a factor 6 away from the optimal weight of tasks which allow an indirect schedule.

Theorem 23. *Let I be an instance of the weighted* Max-Task*-problem on the bidirected grid. It holds that $w(BG_{dir}(I)) \geq \frac{1}{6} w(OPT_{indir}(I))$.*

Proof. Consider the set of tasks T_{ru} and the problem of finding a subset of tasks with maximum weight which allow an indirect schedule. We formulate this problem as an integer program and consider the LP-relaxation LP'_I. It turns out that any solution for LP'_I can be transformed to a solution for LP_I (the relaxation of the IP which finds the set of tasks of maximum weight which allows a direct schedule). Hence, the optimal value $w(x^*)$ for LP_I is also an upper bound on the total weight of a set of tasks which allows an indirect schedule. Hence, $BG_{dir}(T_{ru}) \geq \frac{1}{3} w(x^*) \geq \frac{1}{3} OPT_{indir}(T_{ru})$. Then the claim follows from Lemma 15. □

Now consider the weighted Max-Task-problem on the grid in the setting of indirect schedules. Restricted to the set T_{ru} we can formulate the problem as an integer program and take the LP-relaxation ${LP'_I}^{ru}$. With similar arguments as for trees we can bound the integrality gap by $\eta(p)$ and show how to obtain $\eta(p)$-approximative integral solutions. We conclude with the following theorem.

Theorem 24. *Let I be an instance of the weighted* Max-Task*-problem on the bidirected grid in the setting of indirect schedules. A solution $GR_{indir}(I)$ with weight $w\left(GR_{indir}(I)\right) \geq \frac{1}{2\eta(p)} w\left(OPT_{indir}(I)\right)$ can be found in polynomial time with $\eta(p) = \max\left\{2, 3 - \frac{2}{p}\right\}$.*

4.3. **Price of Directness on the Grid.** Now we show that the price of directness on the grid is bounded from above by 4 and bounded from below by 6/5.

Theorem 25. *The price of directness on the bidirected grid is at most 4.*

Proof. Assume we are given a period length p and a set of tasks T on the bidirected grid which allows an indirect schedule. We show how to select a set of tasks T' such that $w(T') \geq w(T)/4$. We split the set T into subsets T_{lu}, T_{ru}, T_{ld}, and T_{rd}. We note that either $w\left(T_{lu}\right) + w\left(T_{rd}\right) \geq w(T)/2$ or $w\left(T_{ru}\right) + w\left(T_{ld}\right) \geq w(T)/2$. W.l.o.g. we assume that $w\left(T_{ru}\right) + w\left(T_{ld}\right) \geq w(T)/2$. Note that the tasks in T_{ru} do not interfere with the tasks in T_{ld}. Hence, we discuss the two sets independently. Consider the set T_{ru}. We order the tasks descendingly by the column of their bend vertex. Ties are broken by ordering the respective tasks descendingly by the row of their bend vertex. Like in the proof of Theorem 13 we split T_{ru} into two sets T_{ru}^1 and T_{ru}^2 which both allow a direct schedule. For each set T_{ru}^k we maintain a map $task_k : E \times \{0, 1, ..., p-1\} \to T_{ru}$ which defines the direct schedule for the tasks in the respective set. We consider the tasks in the above order. Assume that in the ith iteration we consider the task τ_i. Since each edge is used by at most p tasks (recall that the tasks in T allow an indirect schedule) we can find an offset for τ_i such that it does not collide with any task in T_{ru}^1 or it does not collide with any task in T_{ru}^2 (according to the maps $task_k$). We assign τ_i to the respective set T_{ru}^k and update the map $task_k$ with a suitable offset for τ_i. Finally, we have that either $w\left(T_{ru}^1\right) \geq w\left(T_{ru}\right)/2$ or $w\left(T_{ru}^2\right) \geq w\left(T_{ru}\right)/2$. Denote by T_{ru}^* the set among T_{ru}^1 and T_{ru}^2 with highest weight. With a similar procedure we obtain T_{ld}^*. By construction, there is a direct schedule for the tasks in $T_{ru}^* \cup T_{ld}^*$. We conclude that

$$\begin{aligned} w\left(T_{ru}^* \cup T_{ld}^*\right) &= w\left(T_{ru}^*\right) + w\left(T_{ld}^*\right) \\ &\geq \frac{1}{2}\left(w\left(T_{ru}\right) + w\left(T_{ld}\right)\right) \\ &\geq \frac{1}{4} w(T) \end{aligned}$$

□

Now we give an instance which shows that the price of directness is at least 6/5. It is similar to the instance which shows that the price of directness on bidirected trees is at least 6/5.

Proposition 26. *There is an instance $\bar{I}_{\#}$ of the* Max-Task*-problem on the bidirected grid such that $OPT_{indir}\left(\bar{I}_{\#}\right)/OPT_{dir}\left(\bar{I}_{\#}\right) = 6/5$.*

Proof. We can embed the instance described in Proposition 14 into the bidirected grid (recall Figure 3.2). We then obtain the lower bound of 6/5 with the same reasoning. □

5. Complexity

Due to [4, 5] we already know that the Max-Task-problem on trees and bidirected grid graphs with row-column-paths is $MAXSNP$-hard. Now we show that

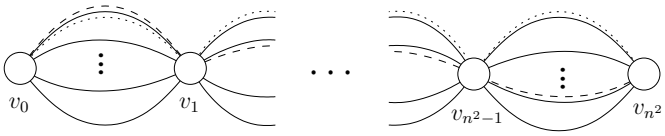

FIGURE 5.1. The dotted and dashed lines denote the path of two tasks τ_i and τ_j such that v_i and v_j are adjacent in G.

on arbitrary graphs and with fixed paths of the tasks the MAX-TASK-problem contains the INDEPENDENT-SET-problem and therefore, it cannot be approximated with a better factor than $|T|^{1-\epsilon}$ for all $\epsilon > 0$.

Theorem 27. *For all $\epsilon > 0$ it is NP-hard to approximate the* MAX-TASK*-problem with fixed paths on chain graphs with an approximation ratio of $|T|^{1-\epsilon}$. This holds for the setting of direct schedules as well as for the setting of indirect schedules.*

Proof. We present a reduction from INDEPENDENT-SET. Given a graph $G = (V, E)$ we construct an instance $I' = (G', T, p)$ of the MAX-TASK-problem such that for any $k \in \mathbb{N}$ the graph G has an independent set of size k if and only if there is a set of tasks $T' \subseteq T$ with $|T'| = k$ which allows a direct or an indirect schedule. Moreover, we will ensure that $|T| = |V|$. A similar reduction was presented in [10]. For all $\epsilon > 0$ it is NP-hard to approximate the INDEPENDENT-SET-problem with an approximation ratio of $|V|^{1-\epsilon}$, see [13] (note that the CLIQUE-problem is the same as the INDEPENDENT-SET-problem on the complement graph). This implies that it is NP-hard to approximate the MAX-TASK-problem with an approximation ratio of $|T|^{1-\epsilon}$ for all $\epsilon > 0$.

Now we present the construction. Let $G = (V, E)$ be a simple undirected graph in which we look for an independent set. Let $n = |V|$. We construct a series parallel graph $G' = (V', E')$ with $n^2 + 1$ vertices $v'_0, ..., v'_{n^2}$. Each pair of adjacent vertices is connected by n edges, see Figure 5.1 for a sketch. We define $|V|$ unit weight tasks. Each task $\tau_i = \left(v'_0, v'_{n^2}, 1\right)$ corresponds to a vertex $v_i \in V$. The paths of the tasks are defined such that the paths P_i and P_j of two tasks τ_i and τ_j share an edge if and only if $\{v_i, v_j\} \in E$. Since $|V'| = n^2 + 1 \geq |E| + 1$ this can clearly be achieved. Finally, we define the period length p of our MAX-TASK-instance by $p := 1$.

Since $p = 1$ no two tasks which share an edge can be assigned to a set of tasks which allows a direct or indirect schedule. This implies that each independent set in G yields a valid set of tasks for I' with the same cardinality and vice versa.

□

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. Andrews, A. Fernández, M. Harchol-Balter, F. Leighton, and L. Zhang. General dynamic routing with per-packet delay guarantees of O (distance + 1/session rate). *SIAM Journal of Computing*, 30:1594–1623, 2000.

[3] C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms*, 3:27, 2007.

[4] T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal of Discrete Mathematics*, 14:326–355, 2001.

[5] T. Erlebach and K. Jansen. Conversion of coloring algorithms into maximum weight independent set algorithms. *Discrete Applied Mathematics*, 148:107 – 125, 2005.

[6] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997.

[7] J. Könemann, O. Parekh, and D. Pritchard. Max-weight integral multicommodity flow in spiders and high-capacity trees. In *Proceedings of the 6th Workshop on Approximation and Online Algorithms*, volume 5426 of *LNCS*, pages 1–14. Springer, 2009.

[8] B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms*, volume 5893 of *LNCS*, pages 217–228, Berlin, 2010. Springer.

[9] B. Peis, S. Stiller, and A. Wiese. The periodic packet routing problem. Technical Report 008-2010, Technische Universität Berlin, April 2010.

[10] C. Puhl and S. Stiller. The maximum capacity of a line plan is inapproximable. Technical Report 028-2007, Technische Universität Berlin, August 2007.

[11] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics. J. Wiley & Sons, 1986.

[12] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

[13] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.