Scheduling of Vehicles in Transportation Networks

Dariusz Kowalski[1], Zeev Nutov[2], and Michael Segal*[3]

[1] Department of Computer Science, University of Liverpool, Liverpool, UK
[2] Department of Computer Science, The Open University of Israel, Raanana, Israel
[3] Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract. In this paper we consider online vehicle scheduling problems for different network topologies under various objective functions: minimizing the maximum completion time, minimizing the largest delay, and minimizing the sum of completion times and present a number of provable approximate solutions.

1 Introduction

Classical traffic scheduling and control theory has been recently compromised by new challenges arising from green transportation, vehicular networks and automated vehicles. This paper studies vehicle scheduling algorithms in different types of networks motivated by scheduling strategies in various types of underlying network topologies and under different measures of performances, optimizing global and local latencies, as well as simplified measure of energy consumption (or CO2 emission). A network is represented as an undirected graph $G = (V, E)$. An (undirected) edge $(u, v) \in E$ between two nodes $u, v \in V$ represents a unit segment (of road) between two locations u and v. A set $C = \{c_1, \ldots, c_k\}$ of k vehicles should be routed through the network, where each vehicle has its own source s_i and destination node d_i. The capacity of each node (in terms of the number of vehicles it can keep at a time) is unlimited; however, each time only one vehicle can pass an edge in either direction but not in both. We consider scenarios in which all vehicles start their routes at the same time. We define a completion time of a vehicle as the time measured since the beginning till the vehicle reaches its destination. Let a delay of a vehicle be defined as a difference between the completion time of the vehicle and the length of its route (i.e., the number of edges in its route). Our goal is to move all the vehicles towards their destinations along some shortest path from its source to its destination (either given or to be chosen by the algorithm) under one or more of the following objectives:

– (*MinMakespan*) Minimizing the maximum completion time, i.e., the time when the last vehicle reaches its destination.

* The work by Michael Segal has been partly supported by France Telecom, European project FLAVIA and Israeli Ministry of Industry, Trade and Labor (consortium CORNET). Email: segal@cse.bgu.ac.il.

– (*MinMax*) Minimizing the largest delay of the vehicles.
– (*MinSum*) Minimizing the total sum of completion times over all vehicles.

The length of the scheduling path produced by some particular vehicle is the number of edges in this path. Observe that the optimum algorithm for the MinSum objective function is also the optimum solution if we replace "completion times" by "delays" in the specification of the MinSum objective function; this is because the sum of completion times is equal to the sum of delays plus the sum of the lengths of the routes, and the latter is fixed and straightforward to compute for a given input. We consider different graph topologies: grid, rooted tree and bipartite thrackle (without shared endpoints). In addition, we distinguish between the settings where, for every vehicle, the path from its source node to its destination node is given or not; in both cases, as we noted earlier, the routes must be shortest possible between the sources and destinations.

2 Related work

In [14], Leighton et al. showed that for any network and any set of packets whose paths through the network are fixed and edge-simple, there exists a schedule for routing the packets to their destinations in $O(c + d)$ steps using constant-size queues, where c is the congestion of the paths in the network, and d is the length of the longest path. Mansour and Patt-Shamir [19] and also Cidon et al. [7] showed that if packets are routed greedily on shortest paths, then all of the k packets reach their destinations within $d + k$ steps. These schedules may be much longer than optimal, however, because k may be much larger than c. Rabani and Tardos [22], and Ostrovsky and Rabani [20] extended the main ideas used in [14] and in the centralized algorithm presented in [16] to obtain online local control algorithms for the general packet routing problem that produce near-optimal schedules. Some related result for trees can be found in [6]. Liu and Zaks [18] studied the greedy algorithm for delivering messages with deadlines in synchronous networks. They considered bottleneck-free networks, in which the capacity of each edge leaving any processor is at least the sum of the capacities of the edges entering it. For such networks where there is at most one simple path connecting any pair of vertices, they have shown a necessary and sufficient condition for the initial configuration to have a feasible schedule, and proved that if this condition holds then the greedy algorithm, that chooses at each step the most urgent messages (those with closest deadlines), determines such a feasible schedule. Adler et al. [1] dealt with the time-constrained packet routing problem when one wants to schedule a set of packets to be transmitted through a multinode network, where every packet has a source and a destination (as in traditional packet routing problems) as well as a release time and a deadline. The objective is to schedule the maximum number of packets subject to deadline constraints. The problem is known to be NP-complete even when the underlying topology is a linear line [2]. For the buffered case, [1] provides logarithmic factor approximation algorithms for the time-constrained scheduling problem with

weighted packets on trees and meshes. Leung et al. [17] considered a problem of routing unit-length, -time messages in different types of networks under various restrictions of four parameters: source node, destination node, release time, and deadline. Peis et al. [21] considered different routing problems for fixed and variable paths for the grid topology. In [13], a line topology has been considered in the setting similar to this work: for directed lines optimal local algorithms have been designed, while for undirected lines no local algorithm is optimal, for each of the three objective functions. Instead, 2-approximation local solutions have been proposed for the latter setting, based on the optimal solutions for directed lines.

Many algorithms designing efficient gathering scheme in the context of wireless networks have been considered before, see e.g. [3–5, 11, 23]. The aim is to minimize the number of steps (*makespan*) needed to send all messages to the base station. Opposite to our model, a node cannot both receive and transmit simultaneously. Moreover, during a step only non interfering transmissions can be done.

This paper is organized as follows. First we consider grid network topology. Next, at Section 4, we turn our attention to the rooted tree networks. Finally, we consider bipartite thrackle networks. We conclude the paper at Section 6.

3 Grid networks

We start by considering the grid version of the problem. We are given a unit-sized grid of size $n \times n$. We assume that all of the paths for the routing vehicles are given, the vehicles are located in the bottom left quarter of the grid and all the paths are dimension order paths (as in [1]). Dimension order routing, which is commonly used on the mesh, requires that each vehicle travels along its source row to the correct column and then along that column to the correct row, or vise versa.

3.1 Optimizing MinMakespan objective

First, we notice that if every two shortest paths have different origin and destination nodes, then the approximation ratio for MinMakespan will be 2, since each path of length l can have at most $l-1$ intersections points with other paths. In the general grid problem, the total number of delays can be as large as $\Omega(n^2)$, and applying the longest-remaining-distance-first strategy, as was efficient for linear networks [13], might not work. In order to see this, suppose that we have $k = n/2$ vehicles starting at the same time and having the same distance to go for their destinations, as deployed at Figure 1. When two of them meet then one of them is delayed by 1. Then the same vehicle that caused this delay (by itself it is not delayed) meets another vehicle, which becomes delayed by 1, and so on. At the end of this part, there is one non-delayed vehicle and $n/2-1$ vehicles delayed by 1. If we apply the same procedure to these $n/2-1$ vehicles we get $n/2-2$ vehicles delayed by 2 and so on. Thus, at the end of this algorithm some of the vehicles may suffer a linear (in n) time delay. On the other hand, the

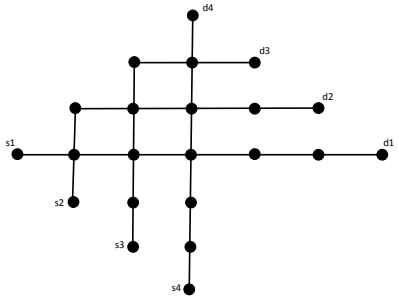

Fig. 1. A scenario producing many delays in a grid when longest-remaining-distance-first strategy is implemented. The number of vehicles/paths is $n/2$.

optimum solution for the example in Figure 1 has MinMakespan of at most the distance-to-go plus one, as this makespan can be obtained for example when in case of collision a priority is always given to a vehicle going up in the grid. In fact, Figure 1 provides a lower bound of 1.5 approximation ratio for *any* greedy strategy that can be applied in order ro resolve collisions between the vehicles, since the greedy strategy will produce the solution of at least the distance-to-go plus the distance-to-go divided by 2.

Next we argue that any algorithm (resolving a contention at node in an arbitrary way) achieves $o(n)$ approximation ratio. In order to see this, we first argue that if every path has at most $o(n)$ intersections we are done. Otherwise, assume that some path has $\Omega(n)$ intersections. If the length of this path is $\omega(1)$, we are again done. Otherwise assume that the length of this path is constant. If the length of some other path is $\omega(1)$, we are done (since the approximation factor will be $n/\omega(1) = o(n)$). Otherwise, the length of all paths is constant. Hence, we have a situation with all path lengths being constant (possibly different for different paths) and some path has $\Omega(n)$ intersections. In this case, the optimum solution schedule takes $\Omega(n)$ time as any algorithm.

Centralized Greedy Longest-to-Destination algorithm with stages In order to achieve better approximation factor we suggest another approach. Recall that all our paths are dimension order paths. Let p be the upper bound on the number of segments in each path (in our case $p = 2$). So, we first solve the problem for vehicles going up (we freeze all the vehicles that start horizontally). We can do it optimally for MinMakespan objective according to the following. We adopt the smallest slack time algorithm from [17], designed in a slightly different setting with deadlines. In this model, the goal is to maximize the number of vehicles that arrive to their destinations within their deadlines. The algorithm in [17] gives priority to the vehicle with the smallest maximum time span that this vehicle can be delayed by without missing its deadline (i.e., that minimizes time to the deadline minus the remaining length of the route). Since the deadlines in

this setting could be chosen arbitrarily large for all vehicles, this is equivalent to giving priority to the vehicle whose destination node is currently farthest away in our model; i.e., both executions — the one in the model with deadlines and the other in our model — result in the same vehicle schedules. Leung et al. [17] proved that this algorithm is optimal for the case of MinMakespan objective criteria for directed graph, in the model with deadlines. By equivalency of executions, the corresponding Furthest-from-Destination algorithm is optimal in our model when all vehicles travel in the same direction.

After all such vehicles arrived at their node of changing direction to the horizontal one, we run the second stage in which the optimum algorithm is applied only for horizontal vehicles (prioritizing the vehicle having largest remaining *horizontal* distance). We continue this procedure until the last segment; in total, we have at most $p+1$ stages.

We argue that the above algorithm achieves $p+1$ approximation. Suppose we are given a set of paths $\mathcal{P}$, each of at most p segments. Consider stage i of the algorithm, for $1 \leq i \leq p+1$. Consider a line L, either horizontal or vertical, in a grid such that L contains at least one segment of some path in $\mathcal{P}$ which is considered in stage i (i.e., either i-th segment, if the path started with vertical segment, or $(i+1)$-th segment otherwise). Observe that during stage i propagation of vehicles along different parallel lines L are independent, in the sense that they do not collide with each other. Let $\mathcal{L} = \mathcal{P}|_L$ denote the family of sets $P \cap L$, over all $P \in \mathcal{P}$. Observe that $\mathcal{L}$ contains one-directional sub-paths of L. Let $\mathrm{OPT}(\mathcal{P})$ denote the optimum solution for $\mathcal{P}$, and $\mathrm{OPT}(\mathcal{L})$ be the optimum solution for input $\mathcal{L}$. By $|\mathrm{OPT}(\mathcal{P})|$ we denote the makespan of $\mathrm{OPT}(\mathcal{P})$, and by $|\mathrm{OPT}(\mathcal{L})|$ we denote the makespan of $\mathrm{OPT}(\mathcal{L})$. Note that $|\mathrm{OPT}(\mathcal{L})|$ is at most $|\mathrm{OPT}(\mathcal{P})|$. Indeed, we could consider $\mathrm{OPT}(\mathcal{P})$ solution in the time period between the first vehicle starts to traverse its segment in $\mathcal{L}$ (i.e., arrives at the beginning of its segment) till the last vehicle finishes its segment in $\mathcal{L}$ (i.e., arrives at the end point of its segment in $\mathcal{L}$), with respect only to the vehicles traversing segments in $\mathcal{L}$. Since we have to show the algorithm on the input set $\mathcal{L}$, we need to modify the above execution as follows. If some of the vehicles has not yet arrived till the starting point of its segment in $\mathcal{L}$ during the considered period of $\mathrm{OPT}(\mathcal{P})$, in the corresponding algorithm for input $\mathcal{L}$ we assume that it stays idle in its starting point; similarly, if a vehicle leaves its segment in $\mathcal{L}$ during the considered part of the execution of $\mathrm{OPT}(\mathcal{P})$, in the corresponding algorithm for $\mathcal{L}$ we assume that it stays in its destination point. Clearly, the length of this time period is at most the length of the whole execution of $\mathrm{OPT}(\mathcal{P})$, the modified algorithm for the purpose of input $\mathcal{L}$ is of the same length and it constitutes a feasible solution of vehicle scheduling for $\mathcal{L}$. This concludes the proof that each stage is of length at most $|\mathrm{OPT}(\mathcal{P})|$. Finally, the analysis is concluded by recalling the upper bound $p+1$ on the number of stages.

Local algorithm. In each node, the decision which vehicle should be pushed to the next node in its path is taken according to the lexicographic order on pairs of parameters $(p+1-\mathit{stage}, \mathit{segm\_dist})$ defined for vehicles residing at this node,

where *stage* denotes the stage number of the vehicle and *segm_dist* stands for the remaining distance to travel by the vehicle in the current segment. The smallest vehicle according to this order is pushed forward to its next node. This algorithm is clearly local, i.e., a decision about pushing a vehicle is made based on the information carried by vehicles located at the node. Moreover, its Makespan is not smaller than in the previous centralized algorithm, i.e., it is bigger than the minimum Makespan by factor at most $p + 1$. To see this, it is enough to observe, using a straightforward inductive argument on the number of rounds, that at any time each vehicle is not further from its final destination than during the centralized scheduling described before.

3.2 Optimizing MinSum and MinMax objectives

In fact, any algorithm (resolving a contention at node in arbitrary way) provides a $O(\sqrt{k})$-approximation for the MinSum criteria for grid structure. For this, suppose that the optimal algorithm (minimizing the sum of delays) schedules all the vehicles to their destinations at maximal arrival time T. Let us look at some vehicle c' and estimate how many vehicles can delay c' on its way to the destination. Knowing that the maximal arrival time in optimal solution is T, we can conclude that the distance of any vehicle to its destination is at most T and the total number of vehicles that can be located at the same source is also T. Consequently, the maximal number of vehicles that can delay vehicle c' is at most $O(T^2)$. At the other hand this number can not be more than k. Taking $T^2 = k$, we obtain the required approximation bound. We note here, that this holds for any bounded degree graph as well.

In order to deal with MinMax objective function, we first solve the decision problem by looking at all the possible configurations of vehicles that allow all of them to arrive to their destinations under the constant maximum delay δ. Let us consider the example in Figure 2.

We explain our idea on linear network, though it works for any undirected graph with n nodes. Every linear network depicted in Figure 2 represents a possible situation in some particular moment of time. Initially some vehicle is located at point A and wants to go to point B. At the next moment ($T = 1$) the vehicle can still be delayed at A or moved one step towards B. This is shown in a horizontal linear network at time $T = 1$. All consequent networks show possible locations of the vehicle at every consequent moment of time. We also draw the directed arrows from the current possible location of the vehicle to the next possible location of the vehicle in the consequent moment of time. Since the distance between A and B is 5, the length of the directed path that shows the possible locations of the vehicle that was not suffering any delay should be 5 — this is depicted in Figure 2 as the directed black solid path. If we would allow this vehicle to suffer up to one delay, then we should also consider the paths defined by the directed red edges (in addition to the directed solid black edges). In such a way we can build, for each vehicle, all relevant edges that it can contribute to the obtained graph under the constraint that each vehicle can suffer up to δ delays. The question we are asking now is whether there are k vertex-disjoint

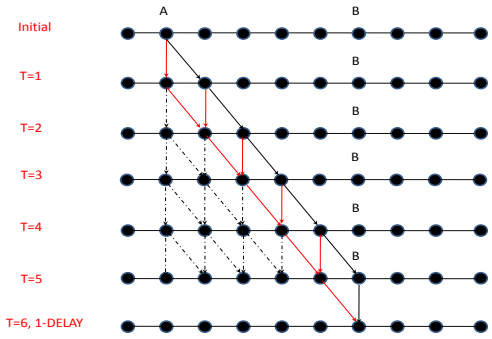

Fig. 2. Possible configurations for the movement of the vehicle from source A to destination B.

paths (we allow situations when the endpoint of one path coincides with some node of another path) between k pairs (s_i, d_i) of sources and destinations in the obtained directed acyclic graph?

There are several papers that studied the problem of finding k vertex-disjoint paths in graphs. For general undirected graphs, the vertex-disjoint paths problem is solvable in polynomial time for any fixed k [24] and is NP-complete if k is a part of the input [12] (here we mention that the problem can be 2-approximated [15] for minimizing the sum of the length of paths for $k = 2$, which can be used for solving our scheduling problem for MinSum objective function). For general directed graphs, the problem is NP-complete even when $k = 2$ [10]. For a directed acyclic graph and constant k, Li et al. [15] gave two efficient pseudo-polynomial time algorithms and Fleischer et al. [9] gave an efficient fully polynomial-time approximation scheme. The currently best solution is due to Yu et al. [25] who actually solve the more general problem of finding k pairwise vertex-disjoint paths such that the maximum length of these k paths is minimized. The optimal algorithm requires $O(n^{k+1}M^{k-1})$ time, and the presented approximation scheme requires $O((1/\varepsilon)^{k-1}n^{2k}\log^{k-1} M)$ time, where ε is the given approximation parameter and M is the length of the longest path in an optimal solution. Thus in order to obtain a solution to our problem we can use the algorithms in [25] in order to obtain a solution to the decision problem with a subsequent binary search scheme. Below we consider the problem of finding the maximum number of pairs from K via node-disjoint paths. We give a $\sqrt{n}$-approximation algorithm for this problem, where $n = |V|$, and show that the problem admits no $O\left(n^{1/2-\varepsilon}\right)$-approximation, unless P=NP.

In other words, the goal is to find a maximum size set of pairs

$$K' = \{(s_1, d_1), \ldots, (s_{k'}, d_{k'})\} \subseteq K$$

and a collection $\{P_1, \ldots, P_{k'}\}$ of pairwise node-disjoint paths in G, such that each P_i is an $s_i d_i$-path, i.e., a path from s_i to d_i. In the other version of the problem, the paths are required to be only pairwise internally-disjoint, so the end node of one path can be located in another path. We call this version the Internally-Disjoint Paths problem.

Theorem 1. Node-Disjoint Paths *and* Internally-Disjoint Paths *admit a* $\sqrt{n}$*-approximation algorithm.*

Proof. Given a graph $G = (V, E)$ and $K \subseteq V \times V$, let $\Pi_G(K)$ be a set of paths obtained by picking the shortest sd-path in G for every $(s, d) \in K$. The algorithm is a standard greedy algorithm, parameterized by ℓ, eventually set to $\ell = \sqrt{n}$.

Algorithm for Node-Disjoint Paths
Initialization: $K' \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$.
While there is $P \in \Pi_G(K)$ with at most ℓ nodes do:
Let $(s, d) \in K$ be the pair that P connects.
Add P to $\mathcal{P}$, move (s, d) from K to K',
and remove P from G.
EndWhile
If there is $P \in \Pi_G(K)$, add P to $\mathcal{P}$ and add to K'
the pair $(s, d) \in K$ that P connects.

It is clear that the algorithm runs in polynomial time and computes a feasible solution to the problem. Let $\mathcal{P}'$ be the set of paths in $\mathcal{P}$ added during the while-loop, and note that every path in $\mathcal{P}'$ has at most ℓ nodes. Let opt be the optimal solution value to the problem, and let opt' be the optimal solution to the residual problem after the end of the while-loop. Clearly, $\mathsf{opt}' \leq n/\ell$. Whenever a path P is added to $\mathcal{P}$, the optimum solution value has decreased by at most the number of nodes in P. Hence $\mathsf{opt} - \mathsf{opt}' \leq \ell \cdot |\mathcal{P}'|$. We claim that $|\mathcal{P}| \geq \mathsf{opt}/\ell$ for $\ell \geq \sqrt{n}$, hence Theorem 1 follows by substituting $\ell = \sqrt{n}$. To see this, consider two cases. If $\mathsf{opt}' = 0$ then $|\mathcal{P}| = |\mathcal{P}'| \geq \mathsf{opt}/\ell$. If $\mathsf{opt}' \geq 1$ then $|\mathcal{P}| = |\mathcal{P}'| + 1 \geq \frac{\mathsf{opt}}{\ell} - \frac{\mathsf{opt}'}{\ell} + 1 \geq \frac{\mathsf{opt}}{\ell} - \frac{n}{\ell^2} + 1$. In both cases, we have $|\mathcal{P}| \geq \mathsf{opt}/\ell$ for $\ell \geq \sqrt{n}$, as claimed.

The algorithm for Internally-Disjoint Paths is similar, except that a path P is added to the partial solution $\mathcal{P}$ if it has at most ℓ internal nodes, and then we remove only the internal nodes of P from G. The proof of the approximation ratio is identical to the one of the Node-Disjoint Paths case. The theorem follows.

Theorem 2. *For any* $\varepsilon > 0$, Node-Disjoint Paths *and* Internally-Disjoint Paths *admit no* $O\left(n^{1/2-\varepsilon}\right)$*-approx-imation algorithm, unless P=NP.*

To prove Theorem 2, we reduce the problem to the Edge-Disjoint Paths problem, which has approximation threshold $|E|^{1/2-\varepsilon}$ for instances with $|E| \geq |V|$. This is done by a standard reduction that converts node-disjoint paths into edge-disjoint paths.

Given an instance $< G = (V, E), K >$ of Edge-Disjoint Paths, define an instance $< G' = (V', E'), K' >$ of Node-Disjoint Paths as follows. The graph G' is obtained from G by replacing every node $v \in V$ by the two nodes v^{in}, v^{out} and the edge (v^{in}, v^{out}), and then replacing every edge $(u, v) \in E$ by the edge (u^{out}, v^{in}). The set of pairs K' is defined by $K' = \{(s^{in}, d^{out}) : (s, d) \in K\}$. Every solution to the Edge-Disjoint Paths problem for input $< G, K >$ corresponds to some solution to the Node-Disjoint Paths problem for the corresponding input $< G', K' >$ of the same size, and vice versa. Moreover, given one of the solutions the other can be computed in polynomial time.

Clearly, $|E'| = |E|+|V|$. Now suppose to the contrary that there exists an approximation algorithm for the Edge-Disjoint Paths problem with ratio $|E'|^{1/2-\varepsilon}$, which by definition must hold for every input $< G', K' >$. This implies that there exists an approximation algorithm for the the Node-Disjoint Paths problem with ratio $(|E| + |V|)^{1/2-\varepsilon} \leq (2|E|)^{1/2-\varepsilon} \leq \sqrt{2}|E|^{1/2-\varepsilon}$, by considering the correspondence between inputs $< G', K' >$ and $< G, K >$ for the Edge-Disjoint Paths and the Node-Disjoint Paths problems, respectively. For $|E|$ large enough, this implies approximation $|E|^{1/2-\delta}$ for some $\delta > 0$. This contradicts the approximation threshold for the Edge-Disjoint Paths problem.

At this stage we are able to provide an $O(\sqrt{n})$ approximation to the number of vehicles that arrive to their destination and suffer no more than δ delay. The question is how the delay will change if we allow all the vehicles to reach their destinations. For this, we apply $O(\sqrt{n})$ approximation algorithm explained above a number of times. After the first time we obtain a number of paths for vehicles that allow them to reach their destinations with δ delay. Call this paths 1^{st} *schedule group*. Next, we remove all of these paths, and repeat the algorithm by finding the paths obtained at second run (2^{nd} *schedule group*). We continue this procedure until no paths for vehicles can be produced and all the vehicles arrived to their destination. Notice, that this process can be repeated no more than $\sqrt{n} \log n$ times. It means that we obtained $\sqrt{n} \log n$ schedule groups. In order to schedule all the vehicles, we divide the time slots into $\sqrt{n} \log n$ sets corresponding to $\sqrt{n} \log n$ schedule groups. The i^{th} schedule group receives $(i \mod \sqrt{n} \log n)^{th}$ time slot to schedule the vehicles on their paths. In such way, we guarantee that all vehicles arrive at their destinations after each vehicle experiences at most $\delta\sqrt{n} \log n$ delays.

We mention here that in order to solve the problem under MinMax delay performance measure, we can follow the same approach as presented above for linear networks, by obtaining 3-dimensional directed acyclic graph and running algorithm for finding k vertex-disjoint paths between sources and destinations.

4 Rooted trees

In the rooted tree version problem, the vehicles should go from theirs sources to their destinations by their shortest paths. Since we have a tree, every path is defined uniquely, i.e., it can be either either vertically upward, vertically downward or upward-downward: go up to the least common ancestor of source and

destination and then downward. For our problem on rooted tree we can use approach similar to the one for grids: we first push only vehicles going up until all of them arrive to their highest point and then looking at the vehicles only going down. In such a way we achieve 3 approximation for this case for MinMakespan criterion.

In order to show the approximation ratio for MinSum criteria we apply shortest remaining distance first algorithm. In this algorithm we always give a priority to the vehicle with the shortest remaining distance to go, where ties are broken arbitrary. For this, suppose that the optimal algorithm (minimizing the sum of delays) schedules all the vehicles to their destinations at maximal arrival time T. Let us look at some vehicle c' and ask a question about how many vehicles can delay c' on its way to destination. Knowing that the maximal arrival time in optimal solution is T, we can conclude that the distance of any vehicle to its destination is at most T and therefore, there are at most $3T$ vehicles that can delay c' on every path's segment (upward or downward) (T vehicles from behind and $2T$ vehicles ahead of c'). We notice that, in general, some vehicle c'' following the same direction as c' can delay c' more than once; however, in this case c'' should also be delayed by some other vehicle. Moreover, each such "repeated" delay can be charged to the vehicle from the set of $2T$ vehicles that will never delay c'. It follows that the total number of delays is at most $6T$ (3T downward and 3T upward), which leads to 7-approximation proof.

5 Bipartite thrackle network

A topological graph G is a graph drawn in the plane with vertices as points and edges as curves connecting its endpoints such that any two edges have at most one point in common. G is called a *thrackle* if every pair of edges intersects. Suppose, we are give a bipartite thrackle (see Figure 3).

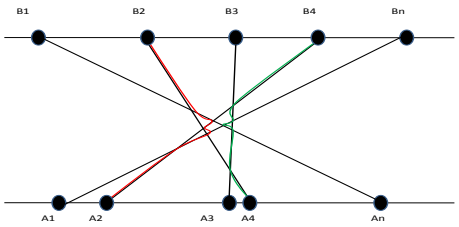

Fig. 3. Bipartite thrackle with switching paths.

In a bipartite thrackle every two lines are intersected in one point and no three lines have a joint point. We assume that passing each segment by vehicle in obtained bipartite thrackle graph takes one unit of time.

The vehicles are located at $A_1, \ldots, A_n$ points (as shown in Figure 3) and should arrive to $B_1, \ldots, B_n$ destinations. Thus, in our case $s_i = A_i$ and $d_i = B_i$. Let us consider the properties of the paths obtained by the following algorithm. When a vehicle wants to move from A_i to B_i, we start at A_i and move up. Whenever we meet another intersecting line, the vehicle moves up to this line (as shown in Figure 3 by even numbered paths). Denote by l_i the length of such path between A_i and B_i in terms of number of segments in this path. First we will make few observations regarding the obtained paths.

1. Path that starts at A_i will surely end at B_i.
2. Every segment of each line belongs to only one path.
3. Adjacent paths (say, k^{th} and $(k+1)^{th}$) are touching by vertices, where k^{th} path is always to the left of $(k+1)^{th}$ path. Moreover, non-adjacent paths do not have joint points.

In fact, if we simply rotate Figure 3 by 90° degrees, we get an arrangement of n lines, where every pair intersects. The paths described above are simple the k-level paths in the arrangement – all points on the k-th path have exactly k-1 segments below them (that is, to their left). By Observation 3, if we take odd (or even) numbered paths, we would not have any intersections between them. Thus in order to schedule our vehicles to satisfy objective MinMakespan, we apply our algorithm separately for odd numbered vehicles, and then for even numbered vehicles.

It is well known that the upper bound for the length of the obtained paths (in terms of the number of segments) produced by our algorithm is $O(n^{\frac{4}{3}})$ [8].

Claim. On the other hand, at least one of the paths produced by the optimal solution algorithm satisfying MinMakespan criterion has at least length n.

Proof. Let us look at the middle path of order $m = \frac{n+1}{2}$ if n is odd, and of order $m = n/2$ if n is even. The path $A_{\frac{n}{2}}B_{\frac{n}{2}}$ partitions the area into 2 parts: each line $A_1B_n, A_2B_{n-1}, \ldots, A_nB_1$ starts in one part and ends at another part. It means that, the path $A_{\frac{n}{2}}B_{\frac{n}{2}}$ is intersected at least $n-1$ times.

Following the discussion above, we get that our algorithm achieves $O(n^{\frac{1}{3}})$ approximation for MinMakespan criterion.

Regarding the MinSum criteria, we note that by Observation 2 and Observation 3 we have $l_i \leq l_{i-1} + l_{i+1}$, where $2 \leq i \leq n-1$. On the other hand, it is clear the the best possible solution is achieved when every vehicle does not experience any delay and the total sum of completion times of all vehicles in this case is at least $\sum_{i=1}^{n} l_i$. If we apply the same algorithm as for the MinMakespan case, we obtain that the total sum of completion times of all vehicles is at most $3\sum_{i=1}^{n} l_i$. Approximation 3 for the sum of completion times follows.

6 Conclusions

At this paper we have considered vehicle scheduling problems for grid, rooted trees and bipartite thrackle networks under criteria of minimizing the maximum completion time, minimizing the largest delay, and minimizing the sum of

completion times. We obtained polynomially solved approximate solutions for our problem. It would be interesting to generalize our scheme to more general networks' topologies.

References

1. M. Adler, S. Khanna, R. Rajaraman and A. Rosen, "Time-constrained scheduling of weighted packets on trees and meshes", *Algorithmica*, 36, pp. 123-152, 2003.
2. M. Adler, A. L. Rosenberg, R. K. Sitaraman and W. Unger, " Scheduling time-constrained communication in linear networks.", in *Proc. 10th ACM Symp. on Parallel Algorithms and Architectures*, pp. 269-278, 1998.
3. J.-C. Bermond, N. Nisse, P. Reyes and H. Rivano, "Minimum delay data gathering in radio networks", in *Ad-Hoc, Mobile and Wireless Networks*, Lecture Notes in Computer Science 5793, pp. 69–82, 2009.
4. V. Bonifaci, R. Klasing, P. Korteweg, L. Stougie and A. Marchetti-Spaccamela, "Data Gathering in Wireless Networks", in *Graphs and Algorithms in Communication Networks*, Springer-Verlag, 2009.
5. V. Bonifaci, P. Korteweg, A. Marchetti-Spaccamela and L. Stougie, "An approximation algorithm for the wireless gathering problem", *Oper. Res. Lett.*, 36(5), pp. 605–608, 2008.
6. C. Busch, M. Magdon-Ismail, M. Mavronicolas and R. Wattenhofer, "Near-Optimal Hot-Potato Routing on Trees", *Euro-Par*, pp. 820–827, 2004.
7. I. Cidon, S. Kutten, Y. Mansour and D. Peleg, "Greedy Packet Scheduling", *SIAM J. Comput.*, 24(1), pp. 148–157, 1995.
8. T. K. Dey, " Improved Bounds for Planar k -Sets and Related Problems", *Discrete & Computational Geometry*, 19(3), pp. 373–382, 1998.
9. R. Fleischer, Q. Ge, J. Li and H. Zhu, "Efficient algorithms for k-disjoint paths problems on dags", in *Proc. of the 3rd Int. Conf. on Algorithmic Aspects in Information and Management*, pp. 134-143, 2007.
10. S. Fortune, J.E. Hopcroft and J. Wyllie, "The directed subgraph homeomorphism problem", *Theoret. Comput. Sci.*, 10, pp. 111-121, 1980.
11. L. Gargano, "Time optimal gathering in sensor networks", in *Proc. SIROCCO*, 4474, pp. 7-10, 2007.
12. R.M. Karp, "On the computational complexity of combinatorial problems", *Networks*, 5, pp. 45-68, 1975.
13. D.R. Kowalski, E. Nussbaum, M. Segal, and V. Milyeykovsky, "Scheduling Problems in Transportation Networks of Line Topology", 2011, manuscript. `http://www.cs.bgu.ac.il/~segal/sch.pdf`.
14. F. T. Leighton, B. M. Maggs and S. B. Rao, " Packet routing and job-shop scheduling in O(congestion + dilation) steps", *Combinatorica*, 14(2), pp. 167–180, 1994.
15. C.-L. Li, S.T. McCormick and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function", *Discrete Appl. Math.*, 26(1), pp. 105-115, 1990.
16. F. T. Leighton, B. M. Maggs, and A. Richa, "Fast algorithms for finding O(Congestion + Dilation) packet routing schedules", *Combinatorica* 19(3), pp. 375–401, 1999.
17. J. Y-T. Leung, T. Tam and G. Young, "On-line routing of real-time messages", *J. of Paral. and Dist. Computing*, 34, pp. 211–217, 1996.

18. K.-S. Liu and S. Zaks, "Scheduling in synchronous networks and the greedy algorithm", *Theor. Comput. Sci.*, 220(1), pp. 157–183, 1999.
19. Y. Mansour and B. Patt-Shamir, "Greedy packet scheduling on shortest paths", *Journal of Algorithms*, 14, pp. 449–465, 1993.
20. R. Ostrovsky and Y. Rabani, "Universal $O(\text{congestion+dilation+}\log^{1+\varepsilon} N)$ local control packet switching algorithms", in *STOC*, pp. 644–653, 1997.
21. B. Peis, M. Skutella and A. Wiese, "Packet routing on the grid", in *LATIN 2010*, LNCS 6034, pp. 120-130, 2010.
22. Y. Rabani and E. Tardos, " Distributed packet switching in arbitrary networks", in *STOC*, pp. 366–375, 1996.
23. Y. Revah and M. Segal, "Improved algorithms for data-gathering time in sensor networks II: ring, tree, and grid Topologies", *International Journal of Distributed Sensor Networks*, 5, pp. 463-479, 2009.
24. N. Robertson and P.D. Seymour, "Graph minors. XIII. The disjoint paths problem", *J. Combin. Theory Ser. B*, 63(1), pp. 65-110, 1995.
25. C.-C. Yu, C.-H. Lin and B.-F. Wang, "Improved algorithms for finding length-bounded two vertex-disjoint paths in a planar graph and minmax k vertex-disjoint paths in a directed acyclic graph", *Journal of Computer and System Sciences*, 76, pp. 697-708, 2010.