# Scheduling Problems in Transportation Networks of Line Topology[*]

Dariusz R. Kowalski[†] Eyal Nussbaum[‡] Michael Segal[‡] Vitaly Milyeykovsky[‡]

**Abstract**

In this paper we consider online scheduling problems for linear topology under various objective functions: minimizing the maximum completion time, minimizing the largest delay, and minimizing the sum of completion times. We show that for the two-directional versions of each problem, no online algorithm can deterministically achieve the optimal solution for any of the considered objective functions, and propose a 2-approximation on-line algorithm for each minimization objective.

## 1 Introduction

In this paper we consider various scheduling problems that arise in the area of traffic scheduling and control. A network is represented as an undirected line $G = (V, E)$. 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 $k$ cars should be routed through the network of $n$ nodes, where each car $i$ has its own source $s_i$ and destination node $d_i$. The capacity of each node — in terms of the number of cars it can keep at a time — is unlimited; however, each time only one car can pass an edge in either direction. All cars start their routes at the same time. We define a *completion time* of a car as the time measured since the beginning of the execution until the car reaches its destination. Let a *delay* of a car be defined as a difference between the completion time of the car and the length of its route (i.e., the number of edges in its route). The goal is to move all the cars towards their destinations, satisfying at least one of the following objectives:

- (*MinMakespan*) Minimizing the maximum completion time, i.e., the time when the last car reaches its destination.
- (*MinMax*) Minimizing the largest delay of the cars.
- (*MinSum*) Minimizing the total sum of completion times over all cars.

MinMakespan objective optimizes global time complexity of car scheduling, which may be important from perspective of managers of the road infrastructure. MinMax objective, in turn, optimizes latency of a single road user. An important criteria of energy consumption and $CO_2$ emission is roughly captured by the MinSum objective.

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 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 focus on solving the above stated problems in linear networks using on-line algorithms as described in [15]. An on-line algorithm is a local algorithm that routes traffic through each edge separately without any knowledge of future arrivals, i.e., an edge traversal decision made at some time step $T$ is based solely on information available to that edge up until time step $T$. We define the information available for decision making by some edge $u, v$ at time step $T$ to be the source and destination nodes of each car occupying nodes $u$ and $v$ at time step $T$. If at least two cars could be scheduled to traverse an edge at the same time, we say that a *collision* occurs at this edge at that time.

An on-line algorithm is said to be optimal for a minimization problem if for any deployment of cars in the network, the algorithm produces an optimal minimum solution. We show that for the two-directional versions of each problem, no on-line algorithm can deterministically achieve the optimal solution for any of the considered objective functions, and propose a 2-approximation on-line algorithm for each minimization objective (an X-approximation on-line algorithm for a minimization problem is one where the result acheived by the algorithm is at most X times larger than the optimally achieved result). We also show that each algorithm requires only a limited-buffer queue for each node in the network.

[*]Work on this paper was partially supported by the Engineering and Physical Sciences Research Council [grant numbers EP/G023018/1, EP/H018816/1], US Air Force European Office of Aerospace Research and Development, grant FA8655-09-1-3016, Deutsche Telecom, France Telecom, European project FLAVIA and Israeli Ministry of Industry, Trade and Labor (consortium CORNET)

[†]Department of Computer Science, University of Liverpool, Liverpool, United Kingdom.

[‡]Communication Systems Engineering Department, Ben-Gurion University of the Negev, Beer-Sheva, Israel.

In the final section, we adjust our model slightly by adding the option of "weighted vehicles", where the delay of different cars have different costs for the system. We investigate a 2-weighted version of this model and show that in this case, no on-line algorithm can acheive the optimal solution for any of our objective funtions, even for the 1-directional network scenarios.

## 2 Related work

In [12], 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 [17] 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 however may be much longer than the optimal ones, because $k$ may be much larger than $c$. Rabani and Tardos [20] and Ostrovsky and Rabani [18] extended the main ideas used in [12] and in the off-line algorithm presented in [14] to obtain on-line local control algorithms for the general packet routing problem producing near-optimal schedules. Some related result for trees can be found in [6].

Liu and Zaks [16] studied a 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. [15] 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. [19] considered different routing problems for fixed and variable paths for the grid topology.

Many algorithms designing efficient gathering scheme in the context of wireless networks have been considered before, see e.g. [3, 4, 5, 10, 21]. 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.

**Our Contribution.** We deliver on-line (local) solutions, for each considered objective, which are optimal if all cars travel in the same direction, and automatically give 2-approximations in the general two-directional setting. We also show that in the general two-directional setting it is impossible to deliver optimal on-line (local) solutions for none of the considered objectives.

## 3 MinMakespan objective

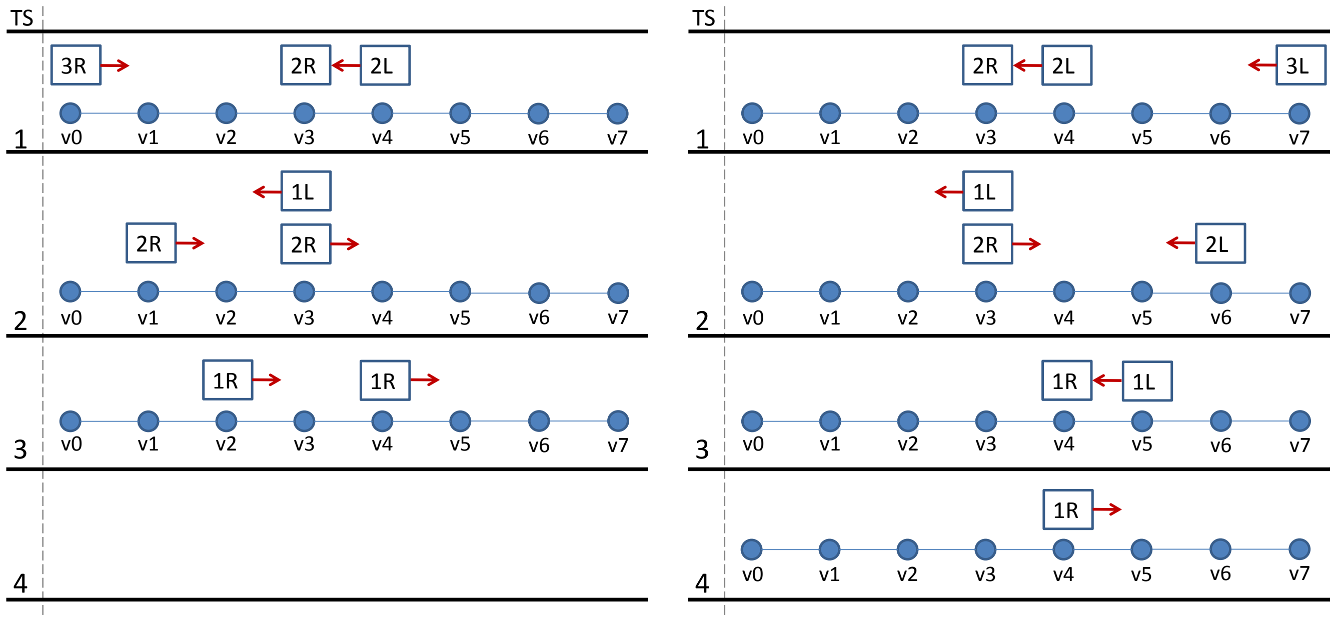
For the MinMakespan objective we adopt the smallest slack time algorithm from [15], designed in a slightly different setting with deadlines. In this model, the goal is to maximize the number of cars that arrive to their destinations within their deadlines. The algorithm in [15] gives priority to the car with the smallest maximum time span that this car 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 cars, this is equivalent to giving priority to the car 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 car schedules. Leung et al. [15] 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 cars travel in the same direction. In order to transform this algorithm into the two-directional scenario, we interleave the one-way algorithm applied separately to the cars moving right and moving left.

Observe that since in the uni-directional scenario we solve the problem optimally, always moving one car from each non-empty queue and at most adding one car to each queue, our algorithm performs with limited buffer queues no larger than their starting queue sizes, with an exception of a maximum needed buffer size of 1 for nodes starting with 0 cars in their queue. The same extends to the general two-directional scenario.

## 3.1 Non feasibility of on-line algorithm

To show that no on-line algorithm exists for optimally solving the two-directional version of the MinMakespan problem, we present the following example depicted in Figure 1. There are two scenarios. In each of them there are two cars, starting at nodes $v_3$ and $v_4$, moving by two hops to the right and two hops to the left respectively. Additionally, in the first scenario, shown in Figure 1(a), there is a third car starting at node $v_0$ and moving by three hops to the right, while in the second "symmetric" scenario, shown in Figure 1(b), the third car starts at node $v_7$ and moves by three hops to the left. Figure 1(a) shows the MinMakeSpan to be 3 in the first scenario (and hence in the same scenario as they are symmetrical). This can be achieved by an off-line algorithm.

Suppose, to the contrary, that there is an on-line algorithm that achieves MinMakespan 3 in both scenarios. In the first round the algorithm decides which of the first two cars, i.e., starting at nodes $v_3$ and $v_4$, passes the link $(v_3, v_4)$ first. This decision is made locally, therefore it is the same in both scenarios. If the car starting at node $v_4$ passes first (which is optimal for the first scenario), then in the second scenario one of the other two cars will yield MinMakespan 4. Mainly, since the car starting at $v_7$ must reach its destination in 3 steps, it means that in the third step it must traverse link $(v_4, v_5)$, and similarly the car that started at node $v_3$; both have one more step to do, therefore at least one of them performs 4 steps, which is a contradiction. In the remaining case when the car starting at node $v_3$ passes to node $v_4$ in the first step, the analysis of the first scenario is symmetric to the analysis of the previous case and also results in a contradiction. Therefore, the MinMakespan of any on-line algorithm for the considered example must be at least 4 and is bigger than the optimum (achieved by some off-line algorithm). This also shows a competitive ratio of $fracoptimum + 1optimum$ for any online algorithm.



(a) Original scenario with optimal MinMakeSpan

(b) Symmetrical scenario with non-optimal MinMakespan

**Figure 1:** LEFT: Optimal strategy for MinMakeSpan is achieved by selection of $2L$ over $2R$ to traverse link $(v_3, v_4)$ at time step 1, resulting in no further collisions. MakeSpan = 3. RIGHT: Lowest possible MakeSpan if $2L$ is selected over $2R$ (in symmetrical scenario) to traverse link $(v_3, v_4)$ at time step 1, resulting in a required decision at time step 3 when a collision occurs. MakeSpan = 4.

## 4 MinMax objective

### 4.1 MinMax off-line algorithm

In case of MinMax objective function, we follow a different approach. First consider scenarios when all cars move into the same direction. We solve the decision version of the problem, namely, given a value $\delta$, we answer whether there is a scheduling scheme for all the cars such that all of them can arrive to their destinations while each one of them suffering a delay which does not exceed $\delta$. Having solved the decision problem, we can apply a binary search on the value of $\delta$ in order to find the optimal solution. In order to solve the decision problem, we again adopt the smallest slack algorithm from [15], when for every car we set a deadline as $\delta + dist(s_i, d_i)$. In this way we solve the problem optimally for uni-directional scenarios. We point out that we do not need to know the possible range of $\delta$ values, since the binary search can be performed on increasing values $1, 2, 4, \ldots, 2^i$, starting at the smallest value. However, the described algorithm is not local, as it requires simulating different executions of the algorithm from [15].

In order to solve the problem in the two-directional case, we divide the time into odd and even time slots. In every odd time slot we give priority to the cars moving in the right direction, while at even slots we give priority to the cars moving in the left direction. Note that the priority resolves collisions between the cars moving in opposite directions; if no collision happens then a car can still move opposite to its priority time slot. This provides us with approximation factor of 2. In the next section we will describe an on-line (local) algorithm for the MinMax objective that is optimal for uni-directional scenarios, and therefore it can be used in order to obtain 2 approximation as an on-line algorithm.

## 4.2 MinMax on-line algorithm

In this section we focus on developing optimum on-line algorithm in uni-directional case. In general two-directional case, a simple scheme of interleaving the two uni-directional algorithms, as explained in the previous section, gives on-line algorithm with approximation factor 2.

We first explain the solution for scenarios when all the cars share the same destination point. Let us enumerate the track sections, starting from the destination, with serial numbers from 1. Section number indicates car's distance in hopes from the destination. We use the following strategy:

Algorithm MaxDelaySolve: The priority is given to the car which was originally closer to the destination.

**Lemma 4.1.** *Algorithm* MaxDelaySolve *optimally solves the problem for the case of one destination.*

*Proof.* Enumerate the cars with continuous numbers from 1 to $k$, starting from the section closest to the destination. Without loss of generality, we can assume that the order of cars' arrivals at the destination is according to their serial numbers. Suppose MaxDelaySolve strategy was violated. Thus, we have the car $c_l$, which started from a section $p$, for some $p \geq 1$, and arrived at destination with order number $l + t$, for some $t > 0$, and the car $c_{l+t}$, which started from a position $p + m$, for some $m > 0$, and arrived with order number $l$. In this case, the delay of car $c_l$ is $l + t - p$, and the delay of car $c_{l+t}$ is equal to $l - p - m$. Consequently, the maximum delay is no less than $\max(l + t - p, l - p - m)$. Subject to MaxDelaySolve strategy, the delay of the car $c_l$ is $l - p$, and the delay of car $c_{l+t}$ is $l + t - p - m$. Thus, the maximum delay that can be produced by $c_l$ and $c_{l+t}$ using MaxDelaySolve strategy is $\max(l - p, l + t - p - m)$. One can see that $\max(l + t - p, l - p - m) > \max(l - p, l + t - p - m)$, since $l + t - p > l - p - m > l - p > l + t - p - m$ for any $l, t, p, m$. Consequently, the violation of MaxDelaySolve strategy only increases the maximum delay. ☐

Next, we propose a solution in case when there are two destination points. We denote the destinations by $A$ and $B$ ($A$ is to the left of $B$ and all the cars go to the left). Let us denote any car having $A$ as the destination point by $c^a$, and any car that goes to $B$ as $c^b$.

Algorithm 2-MaxDelaySolve: The priority is given to the car which was originally closer to $A$.

**Lemma 4.2.** *Algorithm 2-*MaxDelaySolve *optimally solves the problem for the case of 2 destinations.*

*Proof.* Cars to the right of $B$ move towards $B$ in accordance with 2-MaxDelaySolve strategy, since the algorithm operates in all areas of network equally. Suppose the maximum delay in the arrival of these cars to $B$ is equal to $f_b$, and the maximum delay in the arrival of the cars between $A$ and $B$ to $A$ is equal to $f_a$. Assume there is another strategy $\mathcal{A}$, that gives a maximum delay of cars, $m$, which is less than produced by 2-MaxDelaySolve strategy. The maximum delay in the arrival of the cars between $A$ and $B$ at $A$ is $\bar{f}_a \geq f_a$ (due to violation of 2-MaxDelaySolve strategy). Assume that when using $\mathcal{A}$ the last car of type $c^a$ comes to $B$ with delay $d < f_b$. That means there is a car of type $c^b$, that comes to $B$ with delay $e \geq f_b$. If $\bar{f}_a < d$, then 2-MaxDelaySolve strategy gives a solution $f_b$, and $\mathcal{A}$ gives $m \geq e \geq f_b$. If $\bar{f}_a \geq d$, then 2-MaxDelaySolve strategy gives a solution $\max(f_a, f_b)$, and $\mathcal{A}$ gives $\max(\bar{f}_a, e)$ delay. Since $e \geq f_b$, then $\max(\bar{f}_a, e) \geq \max(f_a, f_b)$. Suppose now that $d \geq f_b$ and $e \leq f_b$. If $\bar{f}_a < d$, then 2-MaxDelaySolve strategy gives a solution $max(f_a, f_b)$, and $\mathcal{A}$ gives $m \geq d \geq \max(f_a, f_b)$. If $\bar{f}_a \geq d$, then 2-MaxDelaySolve strategy gives a solution $f_a$, and $\mathcal{A}$ gives $m \geq \bar{f}_a$. It can be seen that any violation from 2-MaxDelaySolve strategy does not improve the solution, since maximal delay is $\max(f_a, f_b)$. ☐

Now, we turn out to deal with the general case of $p$ destinations, with the cars going left. Assume that the destinations are ordered with $d_1$ being the leftmost destination. We call any car that goes to $d_i$ to be *of type* $c^{d_i}$.

Algorithm $p$-MaxDelaySolve: The priority is given to the car which was originally closer to $d_1$.

We will prove the optimality of $p$-MaxDelaySolve strategy by induction on the number of destinations. We have proved that $p$-MaxDelaySolve solves the problem with $p$ destinations for $p = 1, 2$. Let us assume that $p$-MaxDelaySolve solves the problem with $p-1$ destinations. Look at the problem with $p$ destinations. Let us remove destination $d_p$ and $c^{d_p}$ cars. We got a problem with $p-1$ destinations, which we can solve optimally by induction hypothesis. Notice that once we return back $d_p$ and all $c^{d_p}$. Cars to the right of $d_p$, move to $d_p$ according to $p$-MaxDelaySolve, since the algorithm operates in all areas equally. Assume the maximum delay in the arrival of these cars at $d_p$ is equal to $f_p$, and the maximum delay in the arrival of the objects at $d_i, i = 1, \ldots, p-1$ is equal to $f$. Assume there is another algorithm $\mathcal{A}$ that produces a maximum delay, $m$, which is less than one produced by $p$-MaxDelaySolve. Now, the maximum delay in the arrival of the cars at $d_i, i = 1, \ldots, p-1$ is $\bar{f} \geq f$ (due to violation of $p$-MaxDelaySolve). Denote the delay of the arrival of the cars $c^{d_i}, i = 1, \ldots, p-1$ at $d_p$ as $D_i, i = 1, ..., p-1$. Let $D = \max\limits_{i=1,\ldots,p-1} D_i$. Assume that when using $\mathcal{A}$, $D < f_p$. This means that there is a car of type $c^{d_p}$ that comes to $d_p$ with delay $e \geq f_p$. If $\bar{f} < D$, then $p$-MaxDelaySolve gives a solution $f_p$, and $\mathcal{A}$ produces $m \geq e \geq f_p$ delay. If $\bar{f} \geq D$, then $p$-MaxDelaySolve gives a solution $\max(f, f_p)$, and $\mathcal{A}$ gives us $\max(\bar{f}, e)$ delay. Since $e \geq f_p$, then $\max(\bar{f}, e) \geq \max(f, f_p)$. Suppose now that $D \geq f_p$ and $e \leq f_p$. If $\bar{f} < D$, then $p$-MaxDelaySolve gives a solution $\max(f, f_p)$, and $\mathcal{A}$ creates $m \geq D \geq \max(f, f_p)$ delay. If $\bar{f} \geq D$, then $p$-MaxDelaySolve gives a solution $f$, and $\mathcal{A}$ leads to $m \geq \bar{f}$ delay. It can be seen that any violation from $p$-MaxDelaySolve does not improve the solution. Maximal delay is $\max(f, f_p) = \max\limits_{i=1,\ldots,p} f_i$, where each $f_i, i = 1, \ldots, p$ corresponds to maximum delay in the arrival of the objects between at $d_i$ and $d_{i+1}$ at $d_i$. As before, our algorithm performs with limited buffer queues no larger than their starting queue sizes, with an exception of a maximum needed buffer size of 1 for nodes starting with 0 cars in their queue.

### 4.3 Non feasibility of optimal on-line algorithm

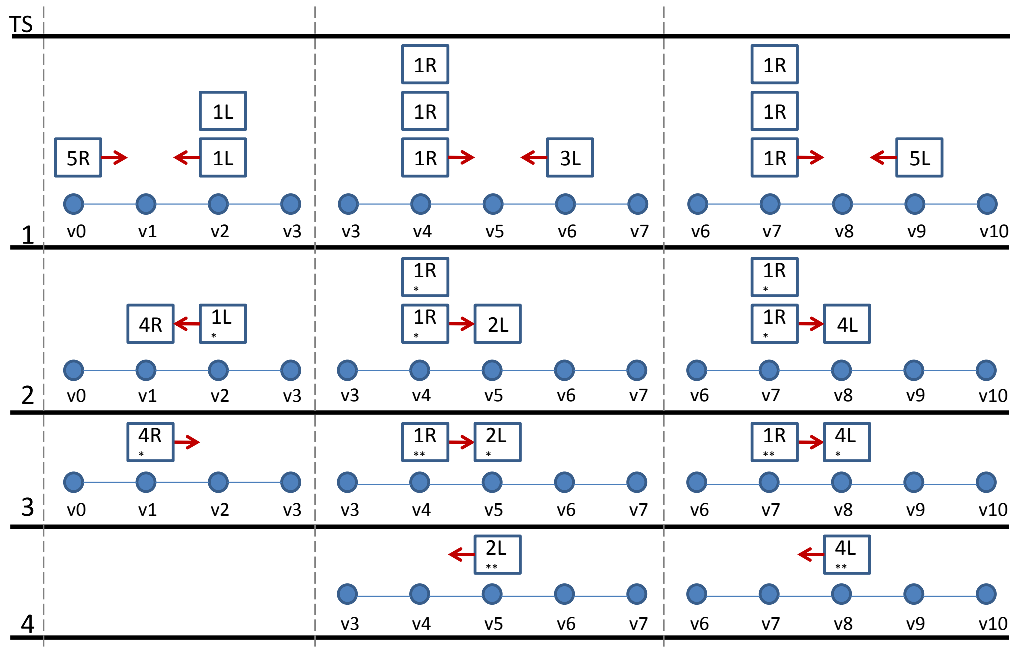


**Figure 2:** Notations: L/R indicates left/right moving car. Number indicates travel distance left. Asterisks indicate number of delays accumulated by car. Example shows decisions needed for achieving optimal MinMax delays in individual scenarios occurring on same network at non-overlapping times.

We begin the proof by examining three simple scenarios occurring in the same network at adjacent segments, but at different times, as shown in Figure 2. All three scenarios are similar, showing a queue of cars needing to proceed in the same direction to the next node as their final destination, while another car is traveling towards them in the opposite direction and will require traversing the same link (causing further delay). In order to achieve the optimal MinMax delay for each scenario, all cars from the original queue must proceed before the arriving car is allowed to traverse the congested link. As such, any optimal on-line algorithm must behave in the same way when given this local information.

We now examine the scenario in Figure 3, which depicts the same three scenarios from the previous example with one difference - all three scenarios occur at the same time on the network. Notice that until time step 5, the two instances are indistinguishable by any node in the network. Since the scenarios are indistinguishable until this time step, an optimal on-line algorithm must behave the same way for each collision as it did for the separate instances, leading to the collisions beginning at time step 5. From this point on, Figure 3 demonstrates the lowest achievable MinMax solution yielded by any algorithm once the situation in time step 5 has been reached, giving a MinMax delay of 3.

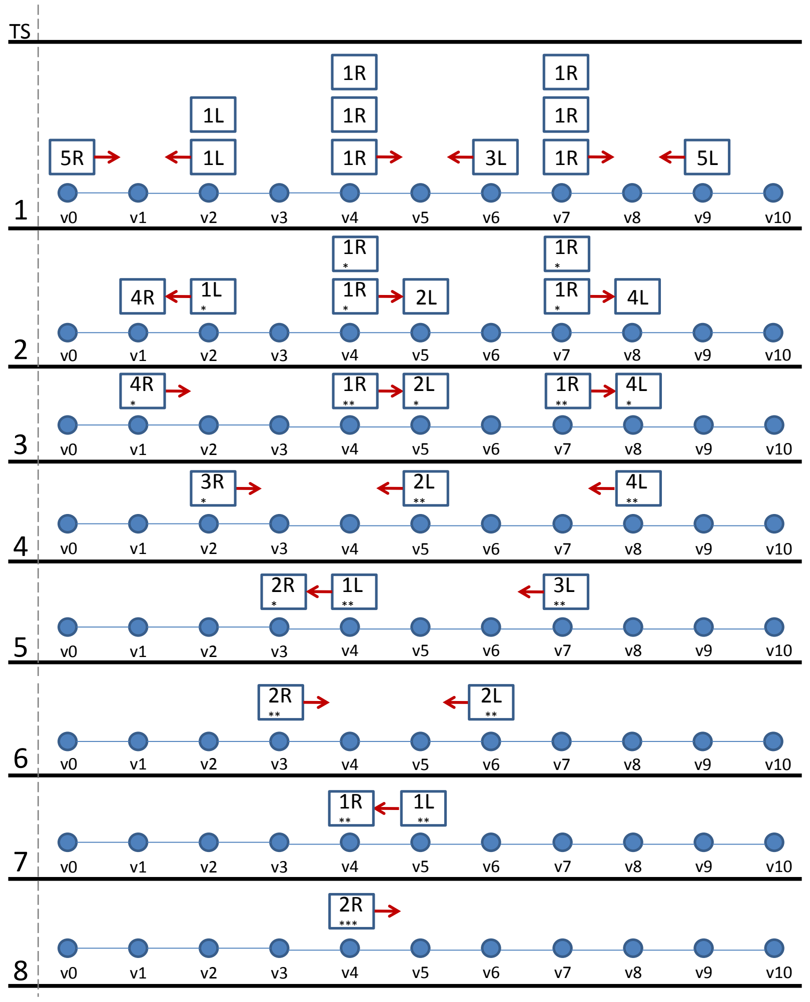


**Figure 3:** Notations: L/R indicates left/right moving car. Number indicates travel distance left. Asterisks indicate number of delays accumulated by car. Example shows MinMax achieved on combined scenarios using the same decisions needed in individually occurring scenarios to achieve optimal MinMax solution. Note that beginning at time step 4, no decision can be made which achieves a lower MinMax value than the one depicted. MinMax delay achieved = 3.

However, as seen in Figure 4, a lower (and in fact optimal) solution with a MinMax delay of 2 can be reached if we change the decision in time step 2, allowing the car traveling right (instead of the cars in the queue with one hop remaining) to traverse the congested link. In this case, the car will continue unimpeded and cause no further collisions while the overall MinMax in the system remains the same (this is due to the fact that in the original individual scenario, the MinMax delay was lower than that of the other two individual scenarios and could be increased without affecting the overall MinMax of the system). As we have shown, there exist at least 2 different scenarios which require conflicting local decisions based on the same information in order to achieve an optimal MinMax solution, and therefore no on-line algorithm can be used to achieve such a solution. This also shows a competitive ratio of $fracoptimum + 1optimum$ for any online algorithm.

## 5 MinSum objective

In this section we show that no on-line algorithm exists for optimally solving the two-directional version of the MinSum problem, and base a 2-approximation algorithm on the ShortestToGo policy (STG for short), which we prove to be and optimal on-line algorithm for the uni-directional version of the problem.

First note that the sum of times spent by each car in the system can be represented as the sum of the remaining cars in each iteration until traversal completion. This is due to the fact that each car increases the sum of the time spent by one in each iteration it remains in the system. Denote by $G_i$ the number of cars in the system at iteration $i$ and $I$ to be the number of iterations until all cars reached their destinations, then the sum of times spent in the system by all cars is $S = \sum_{i=1}^{I} G_i$.

### 5.1 MinSum on-line algorithm

We start by describing the optimal on-line algorithm for solving the uni-directional version of the MinSum problem.

**Definition 1.** *Strategy*: A policy which decides for each node at each time step, which car (if any) proceeds to the next node. A strategy is valid if for a given node and time step (iteration) at most one car proceeds, and all cars

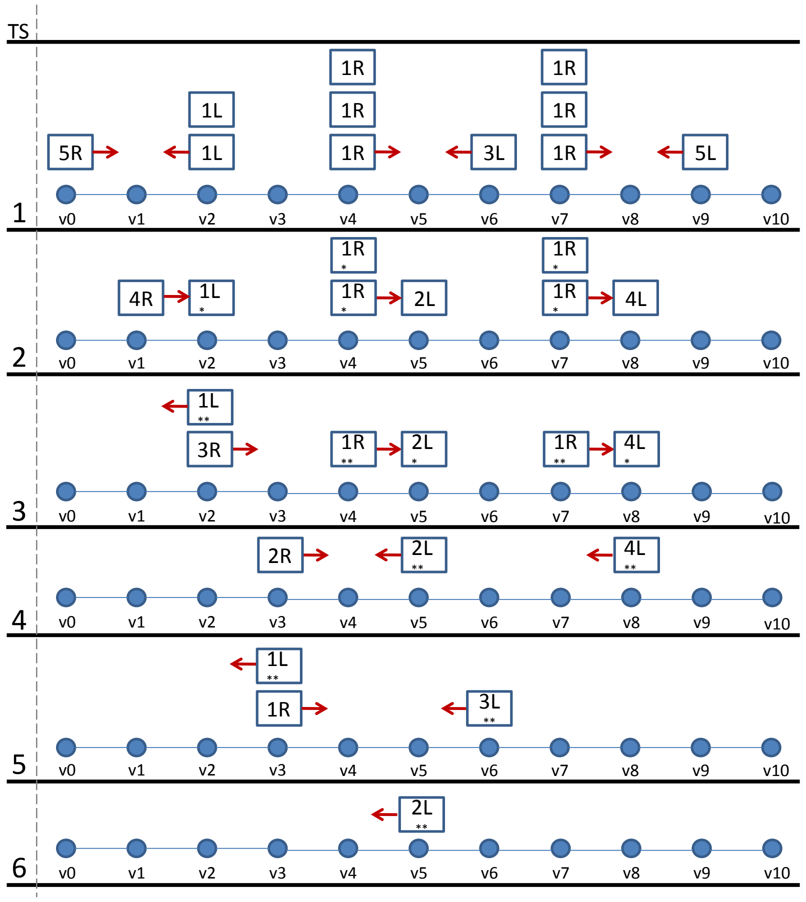


**Figure 4:** Notations: L/R indicates left/right moving car. Number indicates travel distance left. Asterisks indicate number of delays accumulated by car. Example shows MinMax achieved on combined scenarios using a different decision for link $(v_1, v_2)$ at time step 2 than decision made for individually occurring scenarios to achieve optimal MinMax solution. After this decision is made, no further delays are needed and optimal MinMax solution is reached. MinMax delay achieved = 2.

reach their destinations.

**Definition 2.** *Execution*: Given a set of $k$ cars with a source and destination for each car, and some strategy $\sigma$, an execution is the resulting movements of each car in each time step $i \in [0, 1, \ldots, I]$ derived from $\sigma$.

**Definition 3.** *Instruction Set*: An ordered list $P_{c_q}$ of time steps given to a car $c_q$. A car carries out the instructions one at a time, moving to the subsequent node when the next time step in the list has been reached. We label each node $v_i$ with $i$ corresponding to its position in the linear graph from left to right ($i \in N = [0, 1, \ldots, n-1]$). For each car $c_q$ and node $v_i$ we have an instruction $P_{c_q}(i) = t_i^{c_q}$ where $t_i^{c_q}$ is the time step at which $c_q$ leaves node $v_i$. For any node $v_i$ not in the path of $c_q$ we define $t_i^{c_q} = (-1)$.
An instruction set for $c_q$ is valid if it abides by the following constraints:

1. For each $v_i$ not in the path of $c_q$, $P_{c_q}(i) = (-1)$.
2. For each $v_i$ in the path of $c_q$, $0 < P_{c_q}(i) < P_{c_q}(i+1) < \infty$.

Note that for a valid instruction set, exactly $d_{c_q}$ instructions are greater than 0, where $d_{c_q}$ is the number of edges the car must traverse until its destination.
We denote $\Pi$ as the assignment of one instruction set to each car in the system.

Correspondingly, we denote $T_i^{c_q}$ to be the time step in which a car $c_q$ reaches node $v_i$. We define two functions $\phi : C \to N$ and $\psi : C \to N$, which map each car to the index of its source and destination node respectively. Note that $T_{\phi(c_q)}^{c_q} = 0, T_{\phi(c_q)+1}^{c_q} = t_{\phi(c_q)}^{c_q}, \ldots, T_{\phi(c_q)+d_{c_q}+1}^{c_q} = t_{\phi(c_q)+d_{c_q}}^{c_q} = T_{\psi(c_q)}^{c_q}$. (Each car visits one node more than the amount of edges in its path.)

**Definition 4.** *Instruction Based Strategy (IBS)*: Given an instruction set assignment $\Pi$, $IBS(\Pi)$ is the strategy of having each car carry out its instruction set in order. We say $\Pi$ is a valid assignment if the instruction set of each car is valid and $IBS(\Pi)$ is a valid strategy.

If $IBS(\Pi)$ is valid, the contribution of each car $c_q$ to the sum $S$ derived from the resulting execution is: $T^{c_q}_{\psi(c_q)} = T^{c_q}_{\phi(c_q)+d_{c_q}+1} = t^{c_q}_{\phi(c_q)+d_{c_q}}$. The sum $S$ of times spent in the system by all cars in the execution is: $S = \sum_k T^{c_q}_{\psi(c_q)}$.

**Lemma 5.1.** *Any execution based on a valid strategy can be transformed into a valid instruction set assignment $\Pi$, such that using the same initial car placements and following $IBS(\Pi)$ recreates the original execution.*

*Proof.* First set $P_{c_q}(i) = (-1)$ for each car $c_q$. Given an execution based on a valid strategy, mark for each car the time step at which it moved from $v_i$ to a subsequent node as $t^{c_q}_i$. Since in a valid strategy, each car $c_q$ made $d_{c_q}$ ordered moves beginning at $v_{\phi(c_q)}$ and ending at $v_{\phi(c_q)+d_{c_q}}$, we have $d_{c_q}$ values of $t^{c_q}_i$. For each $i$ corresponding to node $v_i \in [v_{\phi(c_q)}, \ldots, v_{\phi(c_q)+d_{c_q}}]$ set $P_{c_q}(i) = t^{c_q}_i$. We now have a valid instruction set for each car and an assignment $\Pi$. It is easy to see that $IBS(\Pi)$ is valid and takes the same actions for each time step as the original strategy. $\square$

**Theorem 5.2.** *Define $S^*$ to be the sum of times spent by all cars in the system using the shortest to go (STG) strategy. For any valid strategy which yields a sum $S$, $S^* \leq S$.*

As the above theorem holds for any strategy, it holds for any optimal strategy solving the problem of MinSum in a uni-directional case. Denote the sum of delays achieved by any such optimum strategy as $S^{1D}_{opt}$. We have $S^{1D}_{opt} \leq S^*$ and $S^* \leq S^{1D}_{opt}$ and, thus, $S^* = S^{1D}_{opt}$. Therefore, for the problem of MinSum in a one-directional linear network, the local STG strategy yields the optimal result. In the remainder of this section, we prove Theorem 5.2.

The proof is by contradiction: suppose that there is a strategy leading to a sum $S$ strictly smaller than $S^*$. By Lemma 5.1 we can create a corresponding instruction set assignment $\Pi$ and achieve the sum $S$ using $IBS(\Pi)$. Let $\tau$ be the first step in this execution where at some node $v_i$ the next car to proceed does not have the shortest remaining distance to go among all cars waiting at node $v_i$ in step $\tau$. Denote this car $c$, and let $c_{STG}$ be a car residing in node $v_i$ in step $\tau$ of the shortest distance to go. There are two possible scenarios:

Scenario 1: Cars $c$ and $c_{STG}$ will meet at some node $v_j$ in the future.

Scenario 2: Cars $c$ and $c_{STG}$ will not meet at any node in the future.

We show that in both scenarios we can modify the instruction set assignment so that the resulting set satisfies the following three properties: has the same sum $S$, all car movements before step $\tau$ follow the STG rule, and the set of cars that do not follow the STG rule in step $\tau$ is a strict subset of the set of cars that do not follow STG in the original instruction set assignment $\Pi$. This argument applied finite number of times would result in final instruction set assignment without any step with a car do not observing STG, thus contradicting our main assumption in the beginning of the proof. In what follows, we first analyze the two scenarios, and then show in more details how to iterate them to obtain final execution with sum $S$ based entirely on STG.

We begin by showing that if two cars meet in two locations then we can switch their instructions in the interval between these locations without changing the value of the objective function.

**Lemma 5.3.** *For an execution based on a given $\Pi$, assume two cars occupied the same node $v_i$ at time step $\tau_1$, and at time step $\tau_2 > \tau_1$ both occupied a different node $v_j$. The instruction set segments $P_{c_q}(i), P_{c_q}(i+1), \ldots, P_{c_q}(j-1)$ can be switched between the two cars creating a new valid $\tilde{\Pi}$. The resulting execution of $IBS(\tilde{\Pi})$ yields a new sum $\tilde{S} = S$.*

*Proof.* Denote the two cars $c_a$ and $c_b$. Since $\Pi$ is valid and both cars occupied nodes $v_i$ and $v_j$ at times $\tau_1$ and $\tau_2$ respectively, we have the following observations:

1. $P_{c_a}(i-1) < \tau_1 \leq P_{c_a}(i) < P_{c_a}(i+1) < \ldots < P_{c_a}(j-1) < \tau_2 \leq P_{c_a}(j)$.
2. $P_{c_b}(i-1) < \tau_1 \leq P_{c_b}(i) < P_{c_b}(i+1) < \ldots < P_{c_b}(j-1) < \tau_2 \leq P_{c_b}(j)$.
3. For any car $c_m \neq c_a$ and any node $v_g$, $P_{c_a}(g) \neq P_{c_m}(g)$.
4. For any car $c_m \neq c_b$ and any node $v_g$, $P_{c_b}(g) \neq P_{c_m}(g)$.

We now switch these instruction set segments between $c_a$ and $c_b$ and denote $\tilde{P}_{c_a}$ and $\tilde{P}_{c_b}$ to be the new instruction sets of $c_a$ and $c_b$ respectively. We have:

a. $\tilde{P}_{c_a} = P_{c_a}(1), \ldots, P_{c_a}(i-1), P_{c_b}(i), P_{c_b}(i+1), \ldots, P_{c_b}(j-1), P_{c_a}(j)$.

b. $\tilde{P}_{c_b} = P_{c_b}(1), \ldots, P_{c_b}(i-1), P_{c_a}(i), P_{c_a}(i+1), \ldots, P_{c_a}(j-1), P_{c_b}(j)$.

We first note that since the instruction sets of all other cars remain unchanged. Also, observations 3 and 4 hold since no new instructions have been allocated to either car, only instructions previously used by them were exchanged. Secondly, we note that observations 1 and 2 hold for the newly created instruction sets $\tilde{P}_{c_a}$ and $\tilde{P}_{c_b}$:

1. $P_{c_a}(i-1) < \tau_1 \leq P_{c_b}(i) < P_{c_b}(i+1) < \ldots < P_{c_b}(j-1) < \tau_2 \leq P_{c_a}(j)$.
2. $P_{c_b}(i-1) < \tau_1 \leq P_{c_a}(i) < P_{c_a}(i+1) < \ldots < P_{c_a}(j-1) < \tau_2 \leq P_{c_b}(j)$.

Therefore, $\tilde{\Pi}$ is valid. Since the last instruction was not changed for any car $c_x$ in the new assignment, it remains that $\forall x : \tilde{T}^{c_x}_{\psi(c_x)} = T^{c_x}_{\psi(c_x)}$, where $\tilde{T}^{c_q}_i$ is the new arrival time of car $c_q$ at node $v_i$ achieved by following the new assignment $\tilde{\Pi}$, and we have:

$$\tilde{S} = \sum_q \tilde{T}^{c_q}_{\psi(c_q)} = \sum_q T^{c_q}_{\psi(c_q)} = S.$$

□

Consider scenario 1. By Lemma 5.3, we can switch instruction segments $P_{c_q}(i), P_{c_q}(i+1), \ldots, P_{c_q}(j-1)$ between $c$ and $c_{STG}$. This gives us a new valid execution with sum $\tilde{S} = S$ where $c_{STG}$ leaves node $v_i$ at time step $\tau$.

Consider scenario 2. We note that the contribution of $c$ and $c_{STG}$ to $S$ are $T^c_{\psi(c)}$ and $T^{c_{STG}}_{\psi(c_{STG})}$ respectively. Denote $v_x$ to be the destination node of $c_{STG}$, $T^c_x$ to be the time step at which car $c$ reaches $v_x$ and $t^c_x$ as the time step instruction in which $c$ leaves this node. Denote the distance between node $v_i$ to the destination nodes of $c$ and $c_{STG}$ as $dist_c$ and $dist_{STG}$ respectively. Denote the difference between their remaining distances as $diff = dist_c - dist_{STG}$.

Suppose we switch the instruction set segments of $c$ and $c_{STG}$ in the line interval $[v_i, v_{i+dist_{STG}-1}]$ in such a way that the instructions of $c_{STG}$ are always ahead of $c$ in this interval. Denote the new instruction sets $\tilde{P}_c$ and $\tilde{P}_{c_{STG}}$ respectively (with the $tilde(\sim)$ notation corresponding with the new departure $(t)$ and arrival $(T)$ times as well). The new instruction set for $c_{STG}$ remains valid with a new $\tilde{T}^{c_{STG}}_{\psi(c_{STG})} = T^c_x$. However, the new instruction set of $c$ is only valid until $\tilde{P}_c(i + dist_{STG} - 1)$, since $\tilde{P}_c(i + dist_{STG} - 1) = T^{c_{STG}}_{\psi(c_{STG})} > \tilde{P}_c(i + dist_{STG}) = P_c(x) = t^c_x$ (otherwise the two cars would have occupied $v_x$ at the same time, which contradicts scenario 2).

**Lemma 5.4.** *Assume a car $c_q$ reached node $v_i$ at some time $\tilde{T}^{c_q}_i = t^{c_q}_i + \Delta > T^{c_q}_i$ for some $\Delta \in [0, 1, 2, \ldots, N]$, and therefore its instruction set is no longer valid. The corresponding instruction $P_{c_q}(i)$ can be changed to $\tilde{P}_{c_q}(i) = \tilde{T}^{c_q}_i + 1$ to create a new valid instruction set without delaying any other car.*

*Proof.* We look at each car $c_l$ occupying node $v_i$ at time step $\tilde{T}^{c_q}_i$. If no car $c_l$ has instruction $P_{c_l}(i) = \tilde{T}^{c_q}_i + 1$, we can safely assign $t^{c_q}_i = \tilde{T}^{c_q}_i + 1$. Otherwise we reassign instructions in order to free up time step $\tilde{T}^{c_q}_i + 1$ for $c_q$ using the following algorithm:

**Step 1:**
Reassign $\tilde{t}^{c_l}_i = max\{T^{c_l}_i + 1, t^{c_q}_i\}$.

**Step 2:**
If the first value $(T^{c_l}_i + 1)$ was chosen and some car $c_m$ has $P_{c_m}(i) = T^{c_l}_i + 1$, repeat Step 1 for $c_m$. Else, continue to step 3.

**Step 3:**
Set $t^{c_q}_i = \tilde{T}^{c_q}_i + 1$.

**Claim 5.5.** *For any car $c_m$ reaching Step 1, the new instruction can only reduce its delay.*

*Proof of Claim:* Originally $P_{c_m}(i) \geq T^{c_m}_i + 1$ (by definition) and $P_{c_m}(i) = T^{c_l}_i + 1 > t^{c_q}_i$ (otherwise $max\{T^{c_l}_i + 1, t^{c_q}_i\} = t^{c_q}_i$ which is not taken by any $c_m$). So, $P_{c_m}(i) > max\{T^{c_l}_i + 1, t^{c_q}_i\} = \tilde{P}_{c_m}(i)$. ■

We now show that the algorithm completes, since there exists at least one car $c_z$ which has reached step 1 and for which either $T^{c_z}_i + 1$ is available or $T^{c_z}_i + 1 \leq t^{c_q}_i$.

**Claim 5.6.** *The algorithm terminates.*

*Proof of Claim:* We have 3 scenarios:

1. $\Delta = 0$.
2. $\Delta > 0$ and some car $c_m$ that we performed step 1 on has $T^{c_m}_i < t^{c_q}_i$ (i.e. reached node $v_i$ prior to time step $t^{c_q}_i$).
3. $\Delta > 0$ and all cars on which step 1 was performed arrived at node $v$ between $t^{c_q}_i$ and $t^{c_q}_i + \Delta$.

Scenario 1 is trivial: if $\tilde{T}_i^{c_q}+1$ is taken, then it is taken by some car $c_z$ for which $T_i^{c_z}+1 \leq t_i^{c_q}+\Delta = t_i^{c_q}$, as $c_z$ arrived at least one time step before $c_q$.
Scenario 2 is also trivial: we say that $c_z = c_m$ and $T_i^{c_z}+1 \leq t_i^{c_q}$.
For scenario 3, we note that there are $\Delta$ time steps in the interval $[t_i^{c_q}, \ldots, t_i^{c_q}+\Delta]$, during which at most $\Delta$ cars arrived and can be processed through the algorithm. We look at some car $c_y$ which reaches step 1 of the algorithm. Since by definition of scenario 3 $T_i^{c_y}+1 > t_i^{c_q}$, we have $\tilde{t}_i^{c_y} = T_i^{c_y}+1$. Since we have at most $\Delta$ cars reaching step 1, we look at the last such car $\hat{c}_y$. If $T_i^{\hat{c}_y}+1$ is unavailable then it is taken by some car for which $T_i^{c_m} < t_i^{c_q}$. However, this contradicts scenario 3. Therefore, $T_i^{\hat{c}_y}+1$ is available. Mark $c_z = \hat{c}_y$.
We now have for each scenario one car $c_z$ which has reached step 1 and for which either $T_i^{c_z}+1$ is available or $T_i^{c_z}+1 \leq t_i^{c_q}$. ■

After all reassignments, we have a new assignment $\tilde{\Pi}$ with $\tilde{P}_{c_q}(i) = \tilde{T}_i^{c_q}+1$ without delaying any other car. Note that $\tilde{P}_{c_q}$ is now valid until $\tilde{P}_{c_q}(i)$. This completes the proof of the Lemma. □

By Lemma 5.4, we can set a new $\tilde{P}_c(x) = T_x^{c_{STG}}+1$ and increase the validity of $\tilde{P}_c$ from $\tilde{P}_c(i+dist_{STG}-1)$ to $\tilde{P}_c(i+dist_{STG}) = \tilde{P}_c(x)$ without delaying any other car. We now look at the instruction set segment $\tilde{P}_c(x), \ldots, \tilde{P}_c(x+diff-1)$. By Lemma 5.4, we can iteratively set a new $\tilde{P}_c(x+D) = \tilde{T}_{x+D}^c+1 = T_x^{c_{STG}}+D+1$ ($D \in [0, 1, \ldots, diff-1]$) for each non valid instruction $\tilde{P}_c(x+D)$. If for some value of $D$ the original instruction remains valid, then the remaining instruction set segment remains valid and we can cease reassignment. In this case, we have a new $\tilde{P}_c$ which is valid with $\tilde{P}_c(x+diff-1) = T_{\psi(c)}^c$ without delaying any other car in the system. If for all values of $D$ a reassignment was needed in order to maintain validity of $\tilde{P}_c$, we have a new valid $\tilde{P}_c$ with $\tilde{P}_c(x+diff-1) = \tilde{T}_{\psi(c)}^c = \tilde{T}_{x+diff-1}^c+1 = T_{\psi(c_{STG})}^{c_{STG}}+diff$ without delaying any other car.
We now show that for all cases of new $\tilde{P}_c$ and $\tilde{P}_{c_{STG}}$, the contribution of both cars to the sum $\tilde{S}$ achieved when using $IBS(\tilde{\Pi})$ cannot be more than their contribution to $S$ when using $IBS(\Pi)$.
The contribution of $c$ and $c_{STG}$ to the sum $S$ when using $IBS(\Pi)$ was $T_{\psi(c_{STG})}^{c_{STG}}+T_{\psi(c)}^c$, and to the sum $\tilde{S}$ when using $IBS(\tilde{\Pi})$ it $\tilde{T}_{\psi(c_{STG})}^{c_{STG}}+\tilde{T}_{\psi(c)}^c$.

1. $\tilde{T}_{\psi(c_{STG})}^{c_{STG}} = T_x^c$.
2. $\tilde{T}_{\psi(c)}^c = max\{T_{\psi(c_{STG})}^{c_{STG}}+diff\, T_{\psi(c)}^c\}$.

Note that $T_x^c < T_{\psi(c_{STG})}^{c_{STG}}$ & $T_x^c \leq T_{\psi(c)}^c - diff$. We have two possible outcomes:

**Case 1:** $T_{\psi(c_{STG})}^{c_{STG}}+diff < T_{\psi(c)}^c$.
We obtain $\tilde{T}_{\psi(c_{STG})}^{c_{STG}}+\tilde{T}_{\psi(c)}^c = T_x^c+T_{\psi(c)}^c < T_{\psi(c_{STG})}^{c_{STG}}+T_{\psi(c)}^c$.

**Case 2:** $T_{\psi(c_{STG})}^{c_{STG}}+diff \geq T_{\psi(c)}^c$.
We obtain $\tilde{T}_{\psi(c_{STG})}^{c_{STG}}+\tilde{T}_{\psi(c)}^c = T_x^c+T_{\psi(c_{STG})}^{c_{STG}}+diff \leq T_{\psi(c)}^c-diff+T_{\psi(c_{STG})}^{c_{STG}}+diff = T_{\psi(c)}^c+T_{\psi(c_{STG})}^{c_{STG}}$.

Since the contributions of all other cars remain unaffected, for the sum $\tilde{S}$ achieved in the execution using $IBS(\tilde{\Pi})$ we get:

$$\tilde{S} = \sum_q \tilde{T}_{\psi(c_q)}^{c_q} \leq \sum_{kq} T_{\psi(c_q)}^{c_q} = S.$$

We now have a valid $\tilde{\Pi}$ in which for time step $\tau$ the car leaving $v_i$ has the STG distance of all cars at node $v_i$. This completes the analysis of Scenario 2.

We continue the procedure based on Scenarios 1 and 2 for each node $v$ which does not have the car with the STG distance remaining leaving it at step $\tau$. This gives us a new $\tilde{\Pi}$ for which $\tilde{S} \leq S$ and for each time step until time step $\tau$, any car leaving a node has the STG distance of all cars at that node. Since this can be done for any time step $\tau$, we again repeat the procedure for $\tau+1, \tau+2, \ldots \tau_{end}$ where $\tau_{end}$ is the time step after which no other car remains in the system. Denote the resulting assignment $\hat{\Pi}$ (and its corresponding sum of delays achieved as $\hat{S}$) and note that in the execution resulting from $IBS(\hat{\Pi})$, any car leaving a node at any time step has the STG distance remaining among any other car at that node. It is easy to see that the execution of the STG strategy and $IBS(\hat{\Pi})$ for the same initial deployment of cars yield the same results, and so $\hat{S} = S^*$. Since we have shown that the execution of any valid strategy yielding a sum $S$ can be transformed into $IBS(\hat{\Pi})$ with $\hat{S} \leq S$, we have: $S \geq \hat{S} = S^*$.

Now that we have an optimal on-line algorithm for the uni-directional MinSum problem, we divide the two-directional problem into 2 sets of uni-directional problems by considering left and right moving vehicles in odd and even time steps respectively. Denote $S_{opt}$ to be the minimum sum for the two-directional problem, $S_{min}^R$ to be the minimum contribution to $S_{opt}$ from all right moving vehicles, and $S_{min}^L$ to be the minimum contribution to $S_{opt}$ from all left moving vehicles ($S_{opt} = S_{min}^R + S_{min}^L$). Denote $S_{opt}^R/S_{opt}^L$ to be the minimum sum for all right/left moving cars in the system assuming that the problem was uni-directional and there were no left/right moving cars in the system. It is easy to see that $S^R_{opt} \leq S^R_{min}$ and $S^L_{opt} \leq S^L_{min}$. Applying the STG strategy to left and right moving vehicles at alternating time steps will yield a total sum of times for cars in system of $2(S^R_{opt} + S^L_{opt})$, since each car stays in the system twice as long as needed for the uni-directional problem. Denote this sum as $S^{2D}_{STG}$. We now have a 2-approximation algorithm: $S^{2D}_{STG} = 2(S^R_{opt} + S^L_{opt}) \leq 2(S^R_{min} + S^L_{min}) = 2S_{opt}$. Again, our algorithm performs with limited buffer queues no larger than their starting queue sizes.

## 5.2 Non feasibility of on-line algorithm

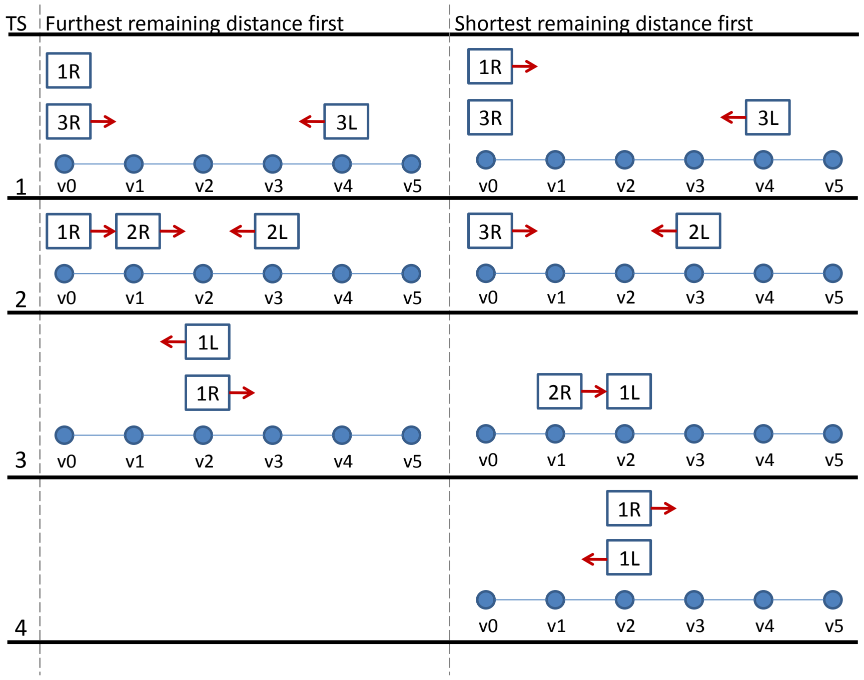

TS
Furthest remaining distance first
Shortest remaining distance first
1R
3R
3L
2R
2L
1L
1
2
3
4
v0
v1
v2
v3
v4
v5

**Figure 5:** LEFT: Optimal strategy for MinSum is achieved by selection of $3R$ over $1R$ to traverse link $(v_0, v_1)$ at time step 1, resulting in no further collisions. Sum = 3 + 3 + 2 = 8. RIGHT: Lowest possible sum if $1R$ is selected over $3R$ to traverse link $(v_0, v_1)$ at time step 1, resulting in a required decision at time step 3 when a collision occurs. Sum = 3 + 2 + 2 + 2 = 9.

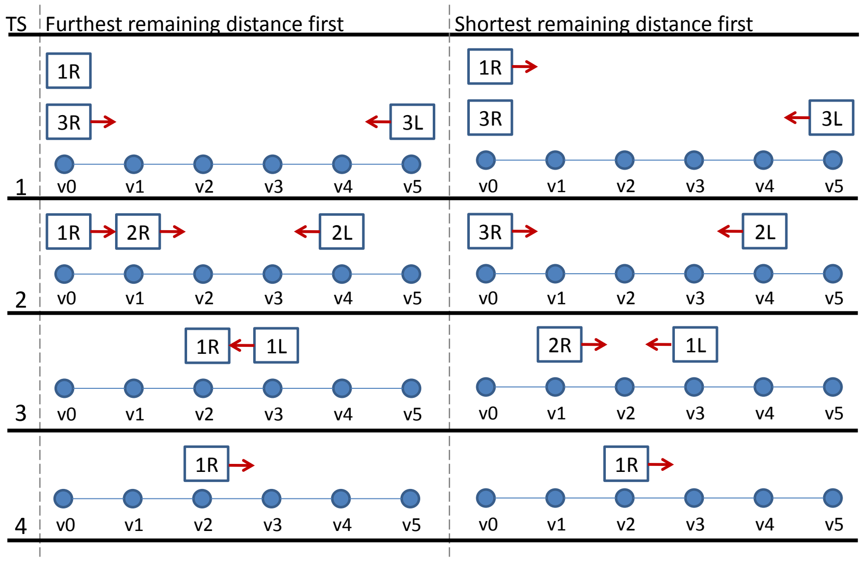

TS
Furthest remaining distance first
Shortest remaining distance first
1R
3R
3L
2R
2L
1L
1
2
3
4
v0
v1
v2
v3
v4
v5

**Figure 6:** LEFT: Lowest possible sum if $3R$ is selected over $1R$ to traverse link $(v_0, v_1)$ at time step 1, resulting in a required decision at time step 3 when a collision occurs. Sum = 3 + 3 + 2 + 1 = 9. RIGHT: Optimal strategy for MinSum is achieved by selection of $1R$ over $3R$ to traverse link $(v_0, v_1)$ at time step 1, resulting in no further collisions. Sum = 3 + 2 + 2 + 1 = 8

Consider the scenarios in Figure 5 and Figure 6. Each car is marked with a number and letter corresponding to the distance remaining and direction of travel respectively. In Figure 5 two cars begin at $v_0$, one with $v_1$ as a destination node and the other with $v_3$. Another car begins at $v_4$ with its destination being $v_1$. At time step 1, a decision must be made as to which car will traverse edge $(v_0, v_1)$. As shown, selecting the car with furthest remaining distance to travel ($3R$) yields the optimal sum of 8 time units, while delaying this car in favor of selecting the car with the shortest remaining distance yields a sum of 9 time units (due to collision avoidance needed at time step 3). Hence, an optimal on-line algorithm must make the former decision.

Now consider the scenario in Figure 6, with the only difference being the source node of the left moving vehicle changed to $v_5$. Since any on-line algorithm has the same information for edge $(v_0, v_1)$ at time step 1, and we have shown that one such deployment requires selecting the car with furthest remaining distance in order to achieve the optimal solution, this same selection should be made in this scenario. However, we can see that with the new scenario, this selection causes a collision at time step 3, which leads to a sum of 9 time units for the algorithm. On the other hand, delaying the car with the furthest remaining distance in favor of advancing the car with the shortest remaining distance over edge $(v_0, v_1)$ yields a sum of 8 time units which is the optimal solution. Since the decision at time step 1 varies between two scenarios while the information remains the same in both of them, it is clear that no on-line algorithm can deterministically make the correct decision for both scenarios and therefore an on-line algorithm cannot be an optimal algorithm. This also shows a competitive ratio of $fracoptimum + 1optimum$ for any online algorithm.

## 6 Weighted Car Model

### 6.1 Model Description

In addition to the problems discussed above, preliminary research was also done into the weighted version of each optimization goal (MinSum, MinMax and MinMakespan). In the weighted model version, all original model characteristics remain the same, however we now add a weight function $w(c)$ which defines for each car its contribution value. We look at a 2-weight model, where $w(c) \in [w_1, w_2]$, and each car is given a constant weight value from the initialization of the system. For simplicity, we allow the weights to be natural numbers, and normalize the values by $min(w_1, w_2)$ such that each car has a weight of 1 or $W = \frac{max(w_1,w_2)}{min(w_1,w_2)}$. In this model, the contribution of each car in each time step is calculated as $w(c)$ instead of 1. In order to comply with the new model, the objective problems are redefined as follows:

- (*Weighted MinMakespan*) Minimizing the maximum weighted completion time, i.e., the minimum value $T_{fin}(c) \times w(c)$ over all cars, where $T_{fin}$ is the finish time of car $c$.
- (*Weighted MinMax*) Minimizing the largest weighted delay of any car, i.e., the minimum value $D(c) \times w(c)$ over all cars, where $D(c)$ is the delay of car $c$.
- (*Weighted MinSum*) Minimizing the total sum of weighted completion times over all cars, i.e., the minimum value of:
$$\sum_k (T_{fin}(c_k) \times w(c_k))$$

This model allows us to examine more complex network behavior, where not all cars are created equal. For example one could look at scenarios where delaying some cars, for example those who have time sensitive deliviries, is more costly than delaying others. Another possible scenario is that some cars are more resource demanding than others. While no optimal or approximation values have yet been found for the above objectives, the following section shows that even for the simple case of the 1-directional problem in a linear network, an optimal on-line algorithm does not exist for any of the objectives.

### 6.2 Non Feasibility of Optimal Algorithm in Weighted Model

#### 6.2.1 Weighted MinMakespan

We take a simple scenario into consideration. Assume 2 cars, each with a different weight, situated at some node $v$ in the network. Denote the car with the lower and higher weight as $c_l$ and $c_h$ respectively. $c_l$ has a distance to travel of $2 \times W + 1$ and $c_h$ has a distance of 1. Clearly, we should send car $c_l$ first and acheive a weighted MakeSpan on $2 \times W + 1$ instead of $2 \times W + 2$. However, assume the scenario with another car, weighted $x$, beginning at the node preceeding node $v$ and having a distance of 2 to travel. Denote this car $c_{h2}$ If the algorithm was optimal, it would again choose $c_l$ to depart node $v$ first. Since $c_{h2}$ now reaches node $v$ and both $c_h$ and $c_{h2}$ must traverse the same edge, the MakeSpan acheived by either car is $3 \times W$. Alternatively, had $c_h$ been chosen to depart node $v$ first, and then $c_{h2}$ in the next time step, the MakeSpan would be set at $2 \times W + 3 < 3 \times W$ by $c_l$ having been delayed twice. Therefore, no optimal on-line algorithm exists for the 1-directional weighted MinMakespan objective.

#### 6.2.2 Weighted MinMax

For the case of weighted MinMax delay we have a slightly more complex scenario. Let the leftmost node in the network be $v_0$ and mark each subsequent node with an increasing index. At node $v_{W+1}$ ($W + 1$ nodes after $v_0$) we place $W + 1$ cars weighted 1. At node $v_1$ we place one car weighted $W$. All nodes have the same destination $v_{W+2}$

It is easy to see that after $W$ timesteps, the queue at node $v_{W+1}$ will consist of 2 cars: a lower weighted car with a weighted delay of $W$ ($c_l$) and a higher weighted car with a weighted delay of 0 ($c_h$). Clearly, the optimal solution would be to delay $c_h$ and acheive a weighted MinMax delay of $W$. We now repeat the scenario but add 4 higher weighted cars to node $v_0$ (with the same destination node $v_{W+2}$). In this case, at time step $W$ we have the same cars at node $v_{W+1}$, and at nodes $v_W, v_{W-1}, v_{W-2}$ we have higher weighted cars with weighted delays of $0, W, 2 \times W$ and $3 \times W$ respectively. Assuming the same optimal algorithm from the previous scenario, we now delay $c_h$. However, this leads to a situation in which one of the higher weighted cars will have a weighted delay of at least $4 \times W$. Alternatively, we could have delayed $c_l$ 5 times, letting the higher weighted cars continue unimpeded, and acheiving a weighted MinMax delay of $W + 5 < 4 \times W$. Therefore, no optimal on-line algorithm exists for the 1-directional weighted MinMax objective.

### 6.2.3 Weighted MinSum

In the weighted MinSum version, we first give the simple example of two cars at some node $v_i$, one lower weighted with a distance to travel of 1 ($c_l$) and one higher weighted with a distance to travel of 2 ($c_h$). Clearly, an optimal algorithm must transmit $c_h$ first in order to achieve a weighted sum of $2 \times W + 2$ instead of $3 \times W + 1$ when transmitting $c_l$ first. Once again, we enhance the scenario and add 2 higher weighted cars to the following node $v_{i+1}$, each with a distance to travel of 1. Following the behavior of the algorithm in the first scenario, $c_h$ will either be delayed by one of cars at $v_{i+1}$ or delay it. Therefore, the weighted sum will be $6 \times W + 2$. Alternatively, transmitting $c_l$ first will now prevent the added delay at $v_{i+1}$ and achieve a weighted sum of $6 \times W + 1$. Therefore, no optimal on-line algorithm exists for the 1-directional weighted MinMax objective.

## 7 Conclusions

In this paper we considered online algorithms for scheduling problems in networks of linear topology under three objective functions: MinMakespan, MinMax and MinSum. We showed that for the two-directional setting, no on-line algorithm can deterministically achieve the optimal solution for any of the considered objective functions, and propose a 2-approximation on-line algorithm for each minimization objective with limited buffer queues no larger than their starting queue sizes. We end our paper by enhancing our original model to that of a weighted-car model, in which the delay of each car can contribute differently to each our proposed objective problems. We then show that in the case of the weighted model, even the 1-directional version of each problem cannot be solved optimally by any on-line algorithm.

There are several interesting directions following from our work. First, it would be interesting to investigate if a tight approximation ratio exists for the problems. Another intriguing question is whether there is an on-line solution for more than one of the considered objectives (which we find unlikely taking into account that the optimum solutions designed in this work are quite different for each measure separately), and if not — what is the tradeoff between these objectives. Finally, the picture becomes even more complex for more advanced network topologies or adjusted models (such as our 2-weighted model), and any work in this direction could be of potential interest.

## 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 Monograph 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] 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.

[9] S. Fortune, J.E. Hopcroft and J. Wyllie, "The directed subgraph homeomorphism problem", *Theoret. Comput. Sci.*, 10, pp. 111-121, 1980.

[10] L. Gargano, “Time optimal gathering in sensor networks”, in *Proc. Structural Information and Communication Complexity*, 4474, pp. 7-10, 2007.

[11] R.M. Karp, “On the computational complexity of combinatorial problems”, *Networks*, 5, pp. 45-68, 1975.

[12] 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.

[13] 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.

[14] 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.

[15] J. Y-T. Leung, T. Tam and G. Young, “On-line routing of real-time messages”, *Journal of Parallel and Distributed Computing*, 34, pp. 211–217, 1996.

[16] K.-S. Liu and S. Zaks, “Scheduling in synchronous networks and the greedy algorithm”, *Theor. Comput. Sci.*, 220(1), pp. 157–183, 1999.

[17] Y. Mansour and B. Patt-Shamir, “Greedy packet scheduling on shortest paths”, *J. of Algorithms*, 14, pp. 449–465, 1993.

[18] R. Ostrovsky and Y. Rabani, “Universal $O(\text{congestion}+\text{dilation}+\log^{1+\varepsilon} N)$ local control packet switching algorithms”, in *Proc. ACM Symp. on Theory of Computing*, pp. 644–653, 1997.

[19] B. Peis, M. Skutella and A. Wiese, “Packet routing on the grid”, in *LATIN 2010*, LNCS 6034, pp. 120-130, 2010.

[20] Y. Rabani and E. Tardos, “ Distributed packet switching in arbitrary networks”, in *Proc. of ACM Symp. on the Theory of Computing*, pp. 366–375, 1996.

[21] 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.

[22] N. Robertson and P.D. Seymour, “Graph minors. XIII. The disjoint paths problem”, *J. Combin. Theory Ser. B*, 63(1), pp. 65-110, 1995.

[23] 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.