The Operational Traffic Control Problem: Computational Complexity and Solutions

Wolfgang Hatzack and Bernhard Nebel

Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Geb. 52, D-79110 Freiburg, Germany
E-mail: ⟨*last name*⟩@informatik.uni-freiburg.de

Abstract. The operational traffic control problem comes up in a number of different contexts. It involves the coordinated movement of a set of vehicles and has by and large the flavor of a scheduling problem. In trying to apply scheduling techniques to the problem, one notes that this is a job-shop scheduling problem with blocking, a type of scheduling problem that is quite unusual. In particular, we will highlight a condition necessary to guarantee that job-shop schedules can be executed in the presences of the blocking constraint. Based on the insight that the traffic problem is a scheduling problem, we can derive the computational complexity of the operational traffic control problem and can design some algorithms to deal with this problem. In particular, we will specify a very simple method that works well in fast-time simulation contexts.

1 Introduction

Assume a set of vehicles (or physical agents) with starting places, starting times, and (perhaps multiple, sequential) goal locations. The problem is now to move the vehicles as fast as possible to the respective goal locations. This is a problem one encounters when trains in a railway system have to be coordinated, when airplanes have to be coordinated in the air or on the ground, when autonomously guided vehicles (AGVs) in a factory or warehouse have to be coordinated, or when a multi-robot group coordinates the movement of the single robots. Interestingly, the problem does not come up in traditional AI planning domains such as *Logistics* (or more generally *transportation* domains [3]). In these domains we never assume that there are capacity restrictions for locations, which implies that vehicles never interfere with each other when moving around.

In all traffic control problems, we can distinguish between the *strategic*, the *tactic*, and the *operational* level. These levels refer to the time span of a day, a few hours, and a few minutes, respectively. We are mainly interested in how to solve the short-time problem, which is, of course, an *on-line* problem in the sense that we do not know the complete input before we start to solve the problem. However, we will consider only the static variant of the problem in the sequel.

In order to solve the problem, we will make some simplifying assumptions. This will help us in finding a satisfycing solution in acceptable time and will at the same time provide us with enough flexibility in the solution that will allow to accommodate new information.

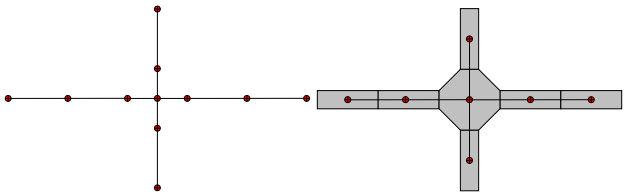
The main simplification we consider is that we assume that the road map for the movements of the vehicles has been fixed in advance. In general, one may want to find solutions independently of a road map. However, this problem can be computationally very demanding. If we are operating in a two-dimensional, rectangular environment and want to coordinate the movement of two-dimensional objects, the decision of whether a goal configuration can be reached is PSPACE-complete [4].

Assuming that the road map is fixed simplifies the problem considerably. However, the problem of finding the minimal number of steps one needs to move all vehicles to the goal locations is still NP-hard as witnessed by the generalized 15-puzzle [10]. For this reason, we will simplify this problem even more. We will assume that the routes the vehicles take are pre-planned and that we only have to *schedule* the movements along these routes. Although this restriction sounds very severe, it is often used in traffic contexts. Furthermore, although simplifying the problem even more, it is still NP-hard to find an optimal solution (as we show below). The resulting problem is similar to the multi-robot *path-coordination* problem [5, p. 379].

The rest of the paper is structured as follows. In Section 2, we formalize the traffic control problem. We then introduce in Section 3 terminology and notions from *scheduling* and show that the traffic problem is a *job-shop* scheduling problem with a *blocking* constraint, which implies that the problem is NP-hard. In Section 4, we will have a look at conditions that guarantee the existence of a solution, and in Section 5 we have a look at methods that allow for "fast-time" simulations. Finally, in Section 6, we report on some experimental data using those methods.

2 The Traffic Control Problem

Each traffic system is based on a specific *infrastructure* that provides facilities on which traffic movements take place, for example roads, airways or waterways. An infrastructure is often represented as a simple graph $G = (V, E)$, where V is a set of intersections or way-points, and E a set of legs. In this paper, however, we represent an infrastructure as a graph where the nodes correspond to a set of resources $R = \{r_1, \ldots, r_m\}$. For our purpose, this allows for a more adequate modeling of infrastructural elements such as intersections, as depicted in Figure 1.



(a) Graph representation (b) Resource representation

Fig. 1. Different representations of an intersection

With $V = \{v_1, \ldots, v_n\}$ we denote a fleet of vehicles that move along the resources of a given infrastructure, where each $v_i \in V$ is associated with a *start time* t_i and an

arbitrary but fixed *route* $\rho_i = (\rho_{i,1} \ldots, \rho_{i,k_i}) \in R^{k_i}$ that might be the result of a path search in the infrastructure or retrieved from a route library, for example. The minimum time it takes v_i to travel along resource $\rho_{i,j}$ is denoted by $\tau_{i,j}$. A *traffic problem* $P = (R, V)$ is a set R of resources and a set V of vehicles with their associated start times and routes. If vehicles never leave the infrastructure, P is called a *closed* traffic problem, whereas in an *open* traffic problem vehicles enter and leave the infrastructure.

The act of v_i moving along resource $\rho_{i,j}$ is called *movement activity* $a_{i,j}$, which allows us to model the movement of all $v_i \in V$ as a *movement plan* $[a_{i,1}, \ldots, a_{i,k_i}]$. A traffic flow arises when vehicles move from their start position at their assigned start time and travel along their specified route to their final position. Formally, a traffic flow F is a set of movement activities $a_{i,j}$ to which a time interval $[\sigma_{i,j}, \phi_{i,j}]$ has been assigned, written $\langle v_i, \rho_{i,j}, [\sigma_{i,j}, \phi_{i,j}]\rangle$. For an orderly movement of vehicles, the following conditions have to be satisfied:

$$\sigma_{i,1} \geq t_i \tag{1}$$

$$\phi_{i,j} - \sigma_{i,j} \geq \tau_{i,j} \tag{2}$$

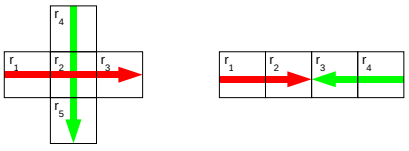
$$\phi_{i,j} = \sigma_{i,j+1}, j \in \{1, \ldots, k_i - 1\} \tag{3}$$

These condition assure that the movement of $v_i \in V$ does not start before its assigned start time t_i, that the actual travel time $\phi_{i,j} - \sigma_{i,j}$ on resource $\rho_{i,j}$ does not fall short of the minimum travel time $\tau_{i,j}$, and finally that the given order of movement activities is preserved without a temporal gap.

For example, if $P = (R, V)$ is a traffic problem with, $V = \{v_1, v_2\}$, $R = \{r_1, r_2, r_3, r_4, r_5\}$, and the connections between the resources as shown in Figure 2 (a), then

$$F = \left\{ \begin{array}{l} \langle v_1, r_1, [0, 10]\rangle, \langle v_1, r_2, [10, 20]\rangle, \langle v_1, r_3, [20, 30]\rangle, \\ \langle v_2, r_4, [0, 10]\rangle, \langle v_2, r_2, [10, 20]\rangle, \langle v_2, r_5, [20, 30]\rangle \end{array} \right\}$$

is a traffic flow for P that satisfies conditions (1) - (3), illustrated in Figure 2 (a).



(a) (b)

Fig. 2. Illustration of traffic flows

If we take a closer look at F, we can see that vehicles v_1 and v_2 plan to use the same resource r_2 simultaneously. Such conflicts can be excluded with the following condition:

$$\rho_{i,j} = \rho_{r,s} \Rightarrow a_{i,j} = a_{r,s} \vee [\sigma_{i,j}, \phi_{i,j}] \cap [\sigma_{r,s}, \phi_{r,s}] = \emptyset \tag{4}$$

However, there is another type of conflict that is an artifact of our movement model. Consider the infrastructure as in Figure 2 (b) and the following traffic flow:

$$F = \left\{ \begin{array}{l} \langle v_1, r_1, [0,10]\rangle, \langle v_1, r_2, [10,20]\rangle, \langle v_1, r_3, [20,30]\rangle, \langle v_1, r_4, [30,40]\rangle, \\ \langle v_2, r_4, [0,10]\rangle, \langle v_2, r_3, [10,20]\rangle, \langle v_2, r_2, [20,30]\rangle, \langle v_2, r_1, [30,40]\rangle \end{array} \right\}$$

Although this traffic flow satisfies conditions (1) - (4), both vehicles are going to exchange their positions at resources r_2 and r_3, which obviously leads to a frontal collision, as sketched in Figure 2 (b). Such situations are avoided if the following condition is satisfied:

$$\rho_{i,j} = \rho_{r,s+1} \wedge \rho_{i,j+1} = \rho_{r,s} \Rightarrow \phi_{i,j} \neq \sigma_{r,s+1} \tag{5}$$

We say that a traffic flow is *safe* if it satisfies conditions (1) - (5). In general, safety has to be established by explicitely resolving conflicts, either by delaying vehicles or assigning new routes to them.[1] Such intervention can be done by human controllers or automatically applying rule-based conflict resolution strategies, for example.

Conflict resolution affects the efficiency of traffic flows, e.g., when a vehicle has to stop in front of an intersection in order to give way to another one. There is a variety of criteria for assessing efficiency of traffic flows. From an infrastructural point of view, an optimal utilization of available resources is desirable, which, for a given set of vehicles, can be achieved by minimizing the latest time at which a vehicle completes its movement. From a vehicle point of view, the most efficient traffic flow minimizes the delay accumulated on the way from its start to finish position.

The completion time C_i and delay D_i of vehicle v_i are defined as follows:

- $C_i = \phi_{i,k_i}$,
- $D_i = \sum_{j=1}^{k_i} \left[(\phi_{i,j} - \sigma_{i,j}) - \tau_{i,j} \right]$.

Based on these definitions, the *maximum completion* time $C_{\max}$ and the *total delay* TD can be defined:

- $C_{\max} = \max_{1 \leq i \leq n} C_i$,
- $TD = \sum_{i=1}^{n} D_i$.

A traffic flow F is *optimal* for TD or $C_{\max}$, if it minimizes the given optimality criterion.

3 Scheduling the Movements: Job-Shop Scheduling with Blocking

Scheduling is concerned with the optimal allocation of scarce resources to activities over time [6]. As we will see in this section, there is a close analogy between finding a safe and optimal traffic flow for a given traffic problem and finding a feasible and optimal schedule for a certain type of scheduling problem.

A scheduling problem $P = (M, J)$ is a set of machines $M = (\mu_1, \ldots, \mu_m)$ and a set of jobs $J = (j_1, \ldots, j_n)$ that have to be processed on machines in M.[2] Typically,

[1] Since we assume routes to be fixed, we do not consider the possibility of re-routing vehicles.

[2] For a general introduction to scheduling, there exists a number of textbooks [1, 2, 8].

scheduling problems are classified in terms of a classification scheme $\{\alpha|\beta|\gamma\}$ [9]. The first field $\alpha = \alpha_1\alpha_2$ describes the machine environment. If $\alpha_1 = O$, we have an *open shop* in which each job j_i consists of a set of operations $\{o_{i,1}, \ldots, o_{i,k_i}\}$ where $o_{i,j}$ has to be processed on machine μ_j for $p_{i,j}$ time units, but the order in which the operations are executed is irrelevant. If $\alpha_1 \in \{F, J\}$, an ordering is imposed on the set of operations corresponding to each job. If $\alpha_1 = F$, we have a *flow shop*, in which each j_i consists of a sequence of m operations $(o_{i,1}, \ldots, o_{i,m})$ and $o_{i,j}$ has to be processed on μ_j for $p_{i,j}$ time units. If $\alpha_1 = J$, we have a *job shop*, in which each each j_i consists of a sequence of k_i operations $(o_{i,1}, \ldots, o_{i,k_i})$ and $o_{i,j}$ has to be processed on a machine $\mu_{i,j} \in M$ for $p_{i,j}$ time units, with $\mu_{i,j} \neq \mu_{i,j+1}$ for i $= 1, \ldots, k_{j-1}$. Note that in a flow shop the *machine routing* is for all jobs the same, while in a job shop the routing is arbitrary but fixed. With α_2 the number of machines can be specified. The second field β indicates a number of job characteristics, for example

- $\{\text{pmtn}\} \subseteq \beta$ (preemption): job splitting is allowed, i.e., the processing of any operation may be interrupted and resumed at a later time.
- $\{\text{nowait}\} \subseteq \beta$: a job must leave a machine immediately after processing is completed. For example, this restriction can be found in the domain of steel production, where molten steel expeditiously has to undergo a series of operations while it has a certain temperature.
- $\{\text{block}\} \subseteq \beta$: a job has to remain on a machine after processing if the next machine is busy. During that time, no other job can be processed on that machine. For example, this phenomena occurs in domains without (or limited) intermediate buffer storage.

The last field γ refers to an optimality criterion which has to be minimized and is a function based on the completion times of jobs which in turn depends on the schedule. The *completion time* of operation $o_{i,j}$ is denoted by $C_{i,j}$ and the time job j_i exits the system is denoted by C_i. Sometimes, for each job j_i a *release date* r_i and a *due date* d_i is specified on which j_i becomes available for processing or should be completed, respectively. With this, the lateness of job j_i can be defined as $L_i = C_i - d_i$ and the *unit penalty* as

$$U_i = \begin{cases} 1 \text{ if } C_i > d_i \\ 0 \text{ else} \end{cases}$$

Typically, γ is one of the following criteria:

- $C_{\max} = \max_{1 \leq i \leq n}\{C_i\}$ is the finish time of the last job.
- $L_{\max} = \max_{1 \leq i \leq n}\{L_i\}$ the maximum lateness.
- $\sum_{i=1}^{n} C_j$ the total flow time
- $\sum_{i=1}^{n} w_i U_i$ the total weight of late jobs

A *schedule* is an allocation of a time interval $[\sigma_{i,j}, \phi_{i,j}]$ on machine $\mu_{i,j}$ to each operation $o_{i,j}$ of all jobs $J_i \in J$. A schedule is *feasible* if no job is processed before its release date (if given), the interval allocated to an operation does not fall short of its specified processing time, no two time intervals allocated to the same job overlap and no two time intervals on the same machine overlap. In addition, a number of specific

requirements concerning machine environment and job characteristic have to be met. In addition, a schedule is *optimal*, if it minimizes a given optimality criterion.

In the sequel, we are mainly interested in job-shop scheduling with *blocking*. Interestingly, job-shop scheduling with blocking is a rather unusual combination. For example, Pinedo states that blocking is a phenomenon that occurs only in flow shops [8] and in the survey article of Hall and Sriskandarajah complexity results only for job shop with no-wait but not with blocking can be found [7]. One reason why most of the research has focused on flow shops may be that most practical applications of *blocking* and *no wait* are in flow shops. However, our traffic control problem is best considered a job-shop problem with blocking. Solutions for such problems have to satisfy the following conditions:

$$\sigma_{i,j} \geq r_i \tag{6}$$

$$\phi_{i,j} - \sigma_{i,j} \geq p_{i,j} \tag{7}$$

$$\phi_{i,j} = \sigma_{i,j+1} \tag{8}$$

$$\mu_{i,j} = \mu_{r,s} \Rightarrow o_{i,j} = o_{r,s} \vee [\sigma_{i,j}, \phi_{i,j}] \cap [\sigma_{r,s}, \phi_{r,s}] = \emptyset \tag{9}$$

Condition (6) states that job i should start at or after the release date of job i and condition (7) requires that the time on machine of the jth subtask of job i is not less than the minimum time required for that subtask. Condition (8) formalizes the *blocking* constraint and, finally, condition (9) states that machines can only be exclusively used.

While these conditions seem to be enough to guarantee that the schedule can be executed (and in fact, for flow-shop problems these conditions are sufficient), in a job-shop environment it might be the case that two jobs with opposite machine routing meet face to face, which is obviously a deadlock and might result in a complete breakdown of the whole system. Therefore, condition

$$\mu_{i,j} = \mu_{r,s+1} \wedge \mu_{i,j+1} = \mu_{r,s} \Rightarrow \phi_{i,j} \neq \sigma_{r,s+1} \tag{10}$$

should also be satisfied. Interestingly, this condition has not been discussed in the scheduling literature yet. The main reason is probably that, as mentioned above, blocking usually happens in flow-shop contexts and the blocking constraint has not been seriously considered for job-shop environments.

To model the traffic problem as a scheduling problem, we consider infrastructural resources as machines, vehicles as jobs and movement activities as operations. Therefore, we have to choose the job shop machine environment $\alpha = J$, which allows us to equate the sequence of movement activities of a vehicle with a jobs sequence of operations. A necessary job characteristic is $\{\text{block}\} \subseteq \beta$, since if a vehicle v_i wants to move from resource $r_{i,j}$ to $r_{i,j+1}$ but $r_{i,j+1}$ is blocked by another vehicle, v_i has to wait on $r_{i,j}$ until $r_{i,j+1}$ becomes available. Finally, the optimality criterion we choose is $\gamma = C_{\max}$, i.e., the minimization of the maximal completion time. In terms of the classification scheme introduced in section 3, $\{J|\text{block}|C_{\max}\}$ is our type of scheduling problem we are going to use for solving traffic problems.

The transformation of a traffic problem into a scheduling problem is straightforward: If $P_{\text{traff}} = (R, V)$ is a traffic problem with resources $R = \{r_1, \ldots, r_m\}$ and vehicles $V = \{v_1, \ldots, v_n\}$ where each vehicle v_i is associated with a movement plan

$[a_{i,1}, \ldots, a_{i,k_i}]$, then $P_{\text{sched}} = (R, V)$ is the corresponding scheduling problem where R is interpreted as a set of machines and V as a set of tasks. Each movement activity $a_{i,j}$ that has to be performed on resource $\rho_{i,j}$ corresponds to an operation $o_{i,j}$ that has to be performed on machine $\mu_{i,j} \in R$. Obviously, a feasible schedule S directly corresponds to a safe traffic flow F. It is obvious that conditions (1) - (5) for safe traffic flows are equivalent to conditions (6) - (10) for feasible schedules.

Proposition 1. *If S is a feasible schedule for a job shop scheduling problem with blocking, then S represents a safe traffic flow for the corresponding traffic problem.*

From this correspondence we can immediately derive a complexity result.

Theorem 1. *The traffic control problem is strongly*[3] *NP-hard if we want to optimize the maximum completion time.*

Proof. As shown by Hall and Sriskandarajah [7], the problem $F_3|\text{blocking}|C_{\max}$ is strongly NP-hard. This however, is clearly a special case of $J_3|\text{blocking}|C_{\max}$, which implies that the traffic control problem with three resources is already strongly NP-hard.

While this result is not surprising, it nevertheless shows that the traffic control problem is a computationally difficult problem. Moreover, the result implies that we should look for heuristic approaches in order to solve it.

4 Solution Existence and the Infrastructure

If we know that regardless of the movements of our vehicles the goals can be reached, we can concentrate on finding a schedule that minimizes the overall costs. Conversely, if it is possible that a system state can be reached from which some vehicles cannot proceed to the goal positions, then we better focus on avoiding such states and consider optimization as secondary.

Let us first consider the situation in Figure 3. Clearly, regardless of what we do, the

Fig. 3. A traffic problem instance without a solution

depicted problem instance does not have a solution. Conversely, if we consider problem instances such that the start and final points are not on the routes of other vehicles, then the problem instance has a solution. The reason is that we could move each vehicle to

[3] Strong NP-hardness means that even if the numbers in the problem description are coded in unary way, the problem remains to be NP-hard.

its final destination, starting with a new vehicle once the starting time has come and the previous vehicle has reached its final destination. This guarantees that we can move all vehicles collision-free to its goals – provided it is enough to start a vehicle movement at some point after its start time. If we have to begin the vehicle movement exactly at the start time, we may run into problems. While the entire restriction sounds very severe, the restriction is satisfied, for example, in *open* infrastructures, such as airports and train stations. For example, at airports we might delay the landing of an airplane for as long as the taxi ways are blocked.

However, even if the problem instance is solvable, it might be possible that a system state is reachable from which the goal cannot be achieved. For example, in Figure 4 a

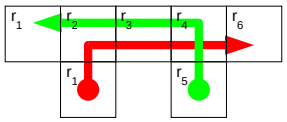


Fig. 4. A traffic problem instance with a possible deadlock situation

situation with a possible deadlock situation is depicted. Such deadlocks can, of course, be anticipated and avoided in the scheduling procedure. However, due to the on-line nature of the problem, it can happen that while executing the activity plan, one vehicle is delayed and the deadlock happens accidentally. In order to avoid that, often the use of resources is restricted in certain ways. For instance, often roads can only be used in a uni-directional way resulting sometimes in detours but avoiding head-on conflicts such as the possible one in Figure 4.

5 Very Fast Approximations for Fast-Time Simulations

In this section, we introduce a very fast algorithm for creating a safe traffic flow for a given traffic problem. Our simulation results indicate that the efficiency of the automatic generated traffic flow can keep up with the efficiency of a traffic flow obtained by a human controller. As a possible application we show how such an automatic traffic controller can be used for assessing the capacity limit of a given infrastructure.

For the corresponding job shop problem with blocking, we built a feasible schedule by incrementally inserting jobs in a first-come-first-served manner into the schedule. The order in which jobs are inserted is determined by their release date. If S is a schedule, then $S_{\mu_j} = \{o_{r,s} \in S | \mu_{r,s} = \mu_j\}$ is the *machine schedule* of $\mu_j \in M$ and $\text{idle}(\mu_j, \sigma, \phi)$ is true if and only if no operation is processed on μ_j during time $[\sigma, \phi]$. Furthermore, $\Phi_{\mu_j} = \{\phi_{r,s} | o_{r,s} \in S_{\mu_j}\}$ is the set of finish times at μ_j and

$$\sigma^*_{i,j} = \begin{cases} r_i & j = 1 \\ \phi_{i,j-1} & j > 1 \end{cases}$$

is the *earliest possible start time* of $o_{i,j}$. If $o_{i,j}$ has to be inserted into schedule $S_{\mu_{i,j}}$, we consider only a finite number of potential start times:

$$\Sigma_{i,j} = \{\sigma^*_{i,j}\} \cup \{\phi \in \Phi_{\mu_{i,j}} | \phi > \sigma^*_{i,j}\}$$

For the example shown in Figure 5, $\Sigma_{i,j} = \{7, 9, 14, 19\}$. Finally, the predicate insertable$(S, o_{i,j}, \sigma)$

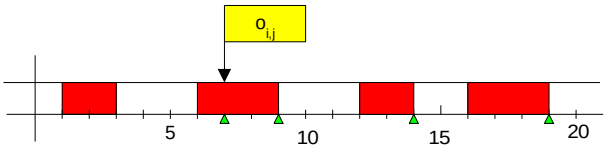


Fig. 5. Potential start times considered for insertion

is true, if and only if in the given schedule S

1. idle$(\mu_{i,j}, \sigma, \sigma + p_{i,j})$ is true, i.e., no other operation is planned on $\mu_{i,j}$ during $[\sigma, \sigma + p_{i,j}]$.
2. Condition (8) can be satisfied, i.e., if $j > 1$, idle$(\mu_{i,j-1}, \phi_{i,j-1}, \sigma)$ has to be true.
3. Condition (10) is not violated, i.e., $o_{i,j}$ does not exchange machines with any other $o_{r,s}$.

As stated above, the basic idea is to sequentially insert jobs into the schedule, so the main algorithm *AutoController* is very simple:

Algorithm AutoController
Params: sequence $(j_1, \ldots, j_n)$ of jobs with $j_i = (o_{i,1}, \ldots, o_{i,k_i})$
Returns: feasible schedule S

$S \leftarrow \emptyset$
for all $j_i \in \{j_1, \ldots, j_n\}$ **do**
  inserted = $false$
  ScheduleActivity$(S, o_{i,1}$, inserted)
end for
return S

For every job j_i, the recursive procedure *ScheduleActivity* is called. In a nutshell, this procedure inserts a given operation $o_{i,j}$ into S and continues recursively with the subsequent operation $o_{i,j+1}$, until finally the last operation o_{i,k_i} has been inserted, causing the boolean variable *inserted* to be set to $true$:

Procedure ScheduleActivity
Params: schedule S, operation $o_{i,j}$, bool *inserted*

if $j \leq k_i$ **then**
  $\Sigma_{i,j} \leftarrow \{\phi^*_{i,j}\} \cup \{\phi \in \Phi_{\mu_{i,j}} | \phi > \sigma^*_{i,j}\}$ *// compute potential start times for* $o_{i,j}$

```
    while Σ_{i,j} ≠ ∅ ∧ ¬inserted do
      σ ← min Σ_{i,j} // get next pot. start time
      Σ_{i,j} ← Σ_{i,j} \ {σ}
      if Insertable(S, o_{i,j}, σ) then
        σ_{i,j} ← σ; φ_{i,j} ← σ_{i,j} + τ_{i,j} // assign start/end time to o_{i,j}
        if j > 1 then
          φ_{i,j−1} ← σ_{i,j} // adapt end time of preceeding operation
        end if
        ScheduleActivity(S, o_{i,j+1}, inserted) // continue recursion with o_{i,j+1}
        if ¬inserted then
          S ← S \ {o_{i,j}}
          if j > 1 then
            φ_{i,j−1} ← σ_{i,j−1} + τ_{i,j−1} // reset end time of preceeding operation
          end if
        end if
      end if
    end while
  else
    inserted ← true // last operation of task j_i has been inserted
  end if
```

Proposition 2. *For a given job shop scheduling problem with blocking, the* AutoController *algorithm returns a feasible schedule.*

That a solution is returned follows from the fact that a new job can always be inserted at the end of a partial schedule. In particular, for every operation $o_{i,j}$ the set of useful start times $\Sigma_{i,j}$ is never empty and the predicate insertable $(S, o_{i,j}, \max(\Sigma_{i,j}))$ is always true. It is easy to show that conditions (6) - (10) are satisfied after each insertion of an operation, so it follows that the returned schedule is feasible.

6 Experimental Results

We tested the AutoController in a traffic simulation based on the infrastructure partially displayed in Figure 6. It is an open traffic system where vehicles dynamically arrive at entry resources, receive a fixed route randomly taken from a standard route library (with average route length 60 resources) and move along that route to a loading point. After a 10-15 minute stay, they move to an exit resource and leave the traffic system. The minimum time needed to travel along a resource is 10 secs for all vehicles.

Both a human controller and the *AutoController* have been confronted with the same random sequence of 58 vehicles whose start times are equally distributed over 1 hour. The resulting traffic flow contains 123 conflicts and is even for skilled controllers very demanding. Since the latest leave time strongly depends on the arrival times of the last few vehicles, both the human controller and *AutoController* achieved the same completion time. Hence, we use average delay as our secondary optimization criterion that plays an important role for an economic utilization of the traffic system. As can

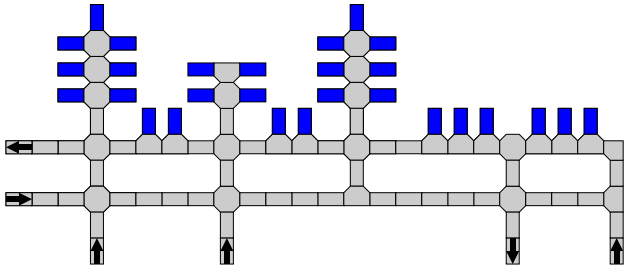
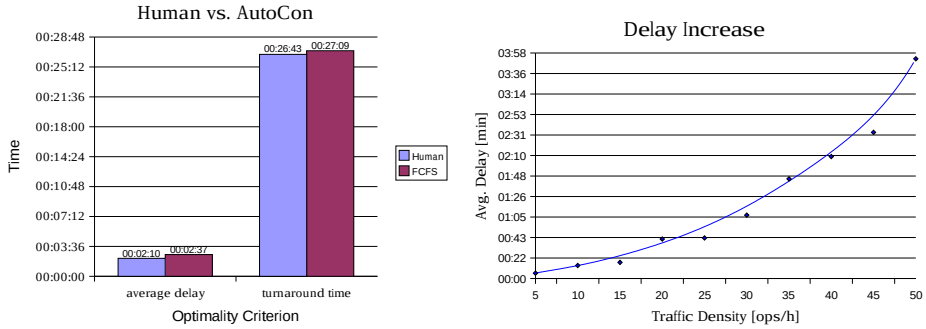


Fig. 6. Simulated infrastructure

be seen in Figure 7 (a), the AutoControllers traffic flow includes 27 secs more average delay than the human controllers traffic flow, which is a difference in performance of 20.77%.



(a) Comparison of human and automatic control (b) Traffic Density and Average Delay

Fig. 7. Simulation Results

However, when taking into account the total time a vehicle spends on the infrastructure (turnaround time), this amounts to a loss of efficiency of merely 1.62%.

Although the *AutoController* is a backtracking algorithm, in our simulation hardly any backtracking occurred. A total of 3429 movement activities has been scheduled and only 83 times an insertion was reversed. Consequently, the algorithm is extremely fast and it took less than 0.5 secs to compute a safe traffic flow on a 300 MHz PC, while on the other hand even a skilled human controller has to run the simulation most of the time in real time which takes about 30-45 minutes.

A typical application that requires the coordination of vehicle movements are fast-time traffic simulations that, among others, are used to evaluate the impact of infrastructural or operational changes in terms of capacity increase or decrease. The capacity limit of a given infrastructure is assessed by gradually increasing traffic density [ops/h] until an acceptable delay limit is exceeded. With the *AutoController* algorithm, the re-

lationship between traffic density and average delay can be determined within minutes, even if an entire day has to be simulated. An example for our infrastructure is shown in Figure 7. In our experiment with the infrastructure as depicted in Figure 6, the capacity limit is 45 [ops/h] if 2:30 min is the acceptable delay limit.

7 Conclusions

The traffic problem is a very common problem occurring when multiple vehicles have to be coordinated. Examples are airport ground traffic coordination, train station coordination, and multi-robot path coordination. We have shown that this problem is a particular kind of scheduling problem, namely, a *job-shop* scheduling problem with *blocking*. This is a rather unusual scheduling problem and it turns out that it is necessary to consider new constraints on schedules, which have not been discussed in the scheduling literature yet, in order to guarantee executability. Nevertheless, the reformulation of the traffic control problem as a scheduling problem allows us to derive the computational complexity of the traffic control problem. Furthermore, on the practical side, the reformulation suggests methods to generate schedules.

We consider restrictions on the problem which guarantee the existence of a solution and we specify a simple, albeit powerful method that is able to generate schedules that are reasonably good. In particular, this method is so fast that it can be used in fast-time traffic simulations, which are needed when doing infrastructure assessments. In an experiment we demonstrate that the simulation method is reasonably good and fast enough to simulate a traffic flow in an infrastructure in a fraction of the time necessary to execute this flow in real-time.

References

1. K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.
2. R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling.* Addison-Wesley, 1967.
3. M. Helmert. On the complexity of planning in transportation and manipulation domains,. Diplomarbeit, Albert-Ludwigs-Universität, Freiburg, Germany, 2001.
4. J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness for the 'warehousman's problem'. *Int. J. Robotics Research*, 3:76–88, 1984.
5. J.-C. Latombe. *Robot Motion Planning.* Kluwer, Dordrecht, Holland, 1991.
6. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves, P. H. Zipkin, and A. H. G. Rinnooy Kan, eds., *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science*, vol. 4, pp. 45–522. North-Holland, 1993.
7. C. Sriskandarajah N.G. Hall. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3), 1996.
8. M. Pinedo. *Scheduling - Theory, Algorithms, and Systems.* Prentice-Hall, 1995.
9. J. Lenstra R. Graham, E. Lawler and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
10. D. Ratner and M. Warmuth. Finding a shortest solution for the (N x N)-extension of the 15-puzzle is intractable. *J. Symbolic Computation*, 10:111–137, 1990.