# Computer-Aided Complexity Classification of Dial-a-Ride Problems

Willem E. de Paepe • Jan Karel Lenstra • Jiri Sgall • René A. Sitters • Leen Stougie

*Cap Gemini Ernst & Young, P.O. Box 2575, 3500 GN Utrecht, The Netherlands*

*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

*Mathematical Institute, Academy of Sciences of the Czech Republic, Žitná 25, CZ-11567 Praha 1, Czech Republic*

*Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

*Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

*willem.de.paepe@cgey.nl* • *jkl@cwi.nl* • *sgall@math.cas.cz* • *r.a.sitters@tue.nl* • *l.stougie@tue.nl*

---

In dial-a-ride problems, items have to be transported from a source to a destination. The characteristics of the servers involved as well as the specific requirements of the rides may vary. Problems are defined on some metric space, and the goal is to find a feasible solution that minimizes a certain objective function. The structure of these problems allows for a notation similar to the standard notation for scheduling and queueing problems. We introduce such a notation and show how a class of 7,930 dial-a-ride problem types arises from this approach. In examining their computational complexity, we define a partial ordering on the problem class and incorporate it in the computer program DaRClass. As input DaRClass uses lists of problems whose complexity is known. The output is a classification of all problems into one of three complexity classes: solvable in polynomial time, NP-hard, or open. For a selection of the problems that form the input for DaRClass, we exhibit a proof of polynomial-time solvability or NP-hardness.

(*Vehicle Routing; Dial-a-Ride; Computational Complexity*)

---

## 1 Introduction

In a *dial-a-ride* problem, a set of *servers* with given capacities and speeds have to serve a set of *rides*. Every ride is characterized by a source and a destination, which are points in some metric space. A rich variety of dial-a-ride problems emerges from the characteristics of the servers, the rides, the metric space, and the objective function.

Dial-a-ride problems occur frequently in practice. Think for example of a taxi station, which has to execute a set of rides every day. Taxi cabs, the servers, have to be allocated to clients, the rides. In this example the client literally "dials a ride." Other examples occur when controlling a set of elevators in a building and planning routes for couriers.

Dial-a-ride problems form a substantial problem class in the field of combinatorial optimization. A large diversity of problem types can be interpreted as members of this class, among which is the classical traveling salesman problem. They are studied from an *off-line* as well as an *on-line* perspective. For off-line problems, all information is available at the outset and the routes for the servers are determined before the rides are actually started; in on-line problems the servers may have to change their routes because additional information becomes available after they started serving the rides. Many practical dial-a-ride problems have a natural on-line character. In this paper we study the computational complexity of off-line dial-a-ride problems. Knowledge about off-line problems can be helpful in finding appropriate ways to deal with their on-line variants. For more about on-line dial-a-ride problems, see Ascheuer et al. (2000) and Feuerstein and Stougie (2001).

Graham et al. (1979) present a short and unambiguous notation for machine scheduling problems. The idea is based on the notation introduced by Kendall (1953) for queueing systems and extended by Conway et al. (1967) to a broader class of queueing and scheduling problems. We propose a similar notation for dial-a-ride problems and describe the problem class that arises from it in Section 2.

Such a notation has an inevitable drawback. It captures many problems that are of interest in their own right, but it also generates a fair number of problems that would not have existed otherwise. While we consider the full class here in an attempt to obtain a complete overview, we feel that a disclaimer is in order. The bare fact that a problem originated in this paper is, in our opinion, not a sufficient reason to subject it to a serious independent investigation. A novel analysis in settling its status or a natural application seem to be required. The contribution of the paper is not to generate thousands of new problems, but to introduce a framework for a class of problems studied before and to show how complexity results can be derived within that framework.

In examining the computational complexity of the problem types in our classification, we aim to expose the borderline between "easy" and "hard" problems. A problem is *easy* if it is solvable in polynomial time, and *hard* if it is NP-hard. In Section 3 we first identify equivalence classes of problems that can be described in more than one way by our notation. We then define a partial ordering on our problem class. Roughly speaking, problems lower in the partial ordering are easier than are problems higher up. In Section 4 we introduce the computer program DaRClass, which systematically employs the partial ordering to decide, given initial sets of easy and hard problems, which of the other problems are easy, hard, or still open. DaRClass determines minimal classes of easy and hard problems necessary to describe the current state of knowledge and reveals critical points on which further research can focus. Lageweg et al. (1982) undertook a similar study for off-line machine scheduling problems.

In Sections 5 and 6 we present some typical results used as input for DaRClass. We prove polynomial-time solvability of some maximal easy problems in Section 5 and NP-hardness of some minimal hard problems in Section 6. An Online Supplement to this paper, available from this journal's website, contains a more elaborate documentation of the results. All proofs that have been omitted from this paper are given there.

In Section 7 we review the literature on the complexity of dial-a-ride problems, and we summarize and analyze the output of DaRClass.

Throughout the paper we tacitly assume that P $\neq$ NP. Finally, we will restrict ourselves to polynomial-time solvability and ordinary NP-hardness. Solvability in pseudopolynomial time and NP-hardness in the strong sense are beyond the scope of the present discussion.

## 2 Notation

We characterize dial-a-ride problems by the contents of four fields. In those fields, we give information about the servers, the rides, the metric space, and the objective function, respectively.

**Servers.** The first field consists of two entries. The first entry is an element of $B_1 = \{1, P, Q, R\}$ and specifies the type of server. There can be a single server or multiple servers. A single server, denoted by 1, travels at unit speed. In case of multiple servers, their number is given as part of the input. As in machine scheduling, we use $P$ for identical parallel servers, $Q$ for uniform parallel servers, and $R$ for unrelated parallel servers. Identical parallel servers all travel at unit speed. Uniform parallel servers travel at their own particular speed. Unrelated parallel servers travel at a speed depending on the number of rides they are executing simultaneously. In this case we assume that a server travels at its own particular speed if it is serving no ride at all and that its speed is a non-increasing function of the number of rides executed simultaneously. Notice that this definition differs from that of unrelated parallel machines, where the speed of a machine depends on the job it is processing.

The second entry of the first field is an element of $B_2 = \{cap1,\ capc,\ cap\infty,\ \circ\}$ and specifies the capacity of the servers. The capacity of a server is defined as the number of rides it can execute simultaneously while always being at one location at a time. In other words, when a ride corresponds to the transportation of an item, the capacity of a server is the number of items it can carry at the same time. The default, denoted by $\circ$, is that each server has its own capacity. Common capacities of 1, $c$, and $\infty$ are denoted by $cap1$, $capc$, and $cap\infty$, respectively. In case of $capc$, the common capacity $c$ is part of the input.

**Rides.** The second field specifies the features of the rides and consists of four entries. The first entry concerns the location of the sources and destinations of the rides and is an element of $B_3 = \{S,\ T,\ s = t,\ \circ\}$. Each ride $j$ has a source $s_j$ and a destination $t_j$, both points in some metric space. More information about the metric space is given below. If $s_i = s_j$ for all $i, j$, the rides have a common source, which is denoted by $S$. If $t_i = t_j$ for all $i, j$, the rides have a common destination, denoted by $T$. We assume that $S$ and $T$ are located in the origin of the metric space. If $s_j = t_j$ for all $j$, each ride has a coinciding source and destination: the rides have length 0, and the requested points just have to be visited, as in the traveling salesman problem; this case is denoted by $s = t$. The default is that none of the three situations described occurs.

The second entry specifies time-window constraints and is an element of $B_4 = \{(r_j, d_j),\ r_j,\ d_j,\ \circ\}$. A ride $j$ cannot be started before its release time $r_j$ and has to be finished by its deadline $d_j$. We denote the general case by $(r_j, d_j)$, the case that there are no deadlines (all $d_j = \infty$) by $r_j$, and the case that there are no release times (all $r_j = 0$) by $d_j$. The default is that there are neither release times nor deadlines.

The third entry is an element of $B_5 = \{pmtn,\ \circ\}$ and indicates whether or not preemption is allowed. If preemption is allowed, a server may start a ride at its source, interrupt it before reaching its destination in order to serve another ride, and resume the ride later at the point where it was interrupted. If there is more than one server, the ride may be resumed by another server than the one that started it; this is called *hand-over*. No preemption is the default.

The last entry is an element of $B_6 = \{prec, \circ\}$ and indicates if precedence constraints on the rides are to be respected. If so, an acyclic digraph $G = (V, A)$ is given. The vertex set $V$ is the set of rides. For each arc $(i, j) \in A$, ride $j$ cannot start before ride $i$ is finished. We will not distinguish between different structures of the precedence graph. No precedence constraints is the default.

**Metric space.** The third field specifies the metric space on which the problem is defined. There is only one entry, which is an element of $B_7 = \{line, tree, graph, \mathbb{R}^d, \circ\}$. On the line and on $\mathbb{R}^d$ we use the Euclidean metric. On trees and graphs we assume that the metric space contains all vertices of the graph; for any two vertices $x$ and $y$, $d(x, y)$ is the length of a shortest path in the graph between $x$ and $y$. The default is a general metric space. In every metric space we specify an origin, denoted by $O$, and we assume that each server starts and ends in the origin. Distances are symmetric and satisfy the triangle inequality.

**Objective.** The last field specifies the objective function and is an element of $B_8 = \{C_{\max}, \sum C_j, \sum w_j C_j\}$. The first objective is to minimize the makespan, i.e. the time all rides have been served and the last server is back in the origin. This will be denoted by $C_{\max}$. The second objective is to minimize the sum of completion times $\sum C_j$, where $C_j$ is the completion time of ride $j$. This objective is equivalent to minimizing the average completion time, also referred to as *latency*. The last objective is to minimize the sum of weighted completion times $\sum w_j C_j$, where every ride $j$ has a given weight $w_j$. In this field, no default is used. Note that the assumption that the servers have to end in the origin does not influence the solution in latency problems. Note also that $C_{\max}$ is a slight abuse of notation: the makespan is not necessarily equal to the maximum ride completion time, since the servers have to return to the origin. We will restrict ourselves here to these three objectives, although most of the other objectives used in machine scheduling occur in practical dial-a-ride problems as well.

A dial-a-ride problem is thus represented as $\beta_1, \beta_2 | \beta_3, \beta_4, \beta_5, \beta_6 | \beta_7 | \beta_8$, where $\beta_i \in B_i$, $i = 1, \ldots, 8$. An empty field means that all entries of that field have been given the default value.

Many problems studied in the operations-research literature can be cast in this notation. One may think, in the first place, of variants of the vehicle routing problem (Lenstra and Rinnooy Kan 1981). For example, $1|s = t||C_{\max}$ is the traveling salesman problem, with symmetric distances that satisfy the triangle inequality, and $1|s=t||\sum C_j$ is the traveling repairman problem, with its more client-orientied objective. $P|S||C_{\max}$ is the vehicle routing problem where vehicles, with individual capacities but a common speed, have to deliver unit loads from a single depot to customers, so as to minimize the time at which the last vehicle returns to the depot; $P|(r_j,d_j)||C_{\max}$ is the familiar pickup and delivery problem with time-windows. The problem $1, cap1||graph|C_{\max}$ covers several arc routing problems. When, for every ride $j$, $\{s_j,t_j\}$ is an edge of the graph, we have the stacker crane problem (Frederickson et al. 1978); when, for every edge $\{x,y\}$ of the graph, there is a ride $(x,y)$ or $(y,x)$, we have the directed Chinese postman problem.

The notation does have its limitations. Since the distances are symmetric, it cannot describe the asymmetric traveling salesman problem. Since the rides are directed, it does not capture the undirected and mixed Chinese postman problems either. It also assumes that the items to be transported have the same size and thereby ignores the packing aspect. This important issue in vehicle routing does appear in a more extensive classification due to Desrochers et al. (1990), which was developed for an entirely different purpose.

Problems with $\beta_2 = cap1$ (servers of unit capacity), $\beta_3 \in \{S,T\}$ (all rides from the origin, or all rides to the origin), and $\beta_5 = \circ$ (no preemption) are equivalent, or very closely related, to single or parallel machine scheduling problems. Many machine scheduling problems (Graham et al. 1979) have their counterpart in our notation. Preemption in dial-a-ride has a different interpretation than in machine scheduling, however.

# 3 Equivalence Classes and Partial Ordering

Let $\mathcal{S}$ be the class of dial-a-ride problems defined in Section 2. We would like to employ the *reducibility relation* $\propto$ in order to obtain complexity results within $\mathcal{S}$, following the work of Karp (1972). This relation is a polynomial transformation between problem types with the property that, for any two problems $\Pi$, $\Pi' \in \mathcal{S}$,

- if $\Pi \propto \Pi'$ and $\Pi'$ is easy, then $\Pi$ is easy, and
- if $\Pi \propto \Pi'$ and $\Pi$ is hard, then $\Pi'$ is hard.

However, we also would like to automate the derivation of the reductions. That is, DaR-Class should be able to construct reductions on its own, and the question is how much intelligence it can be endowed with. We will settle for the definition of a *subrelation* of $\propto$, denoted $\rightarrow$, which captures some types of reductions that are readily recognizable. DARCLASS will not replace the human effort of obtaining original polynomial-time algorithms or clever NP-completeness proofs, but it will clarify the consequences of each such new result.

In defining the relation $\rightarrow$, we encounter a phenomenon that was anticipated by Lageweg et al. (1982). It may happen that both $\Pi \rightarrow \Pi'$ and $\Pi' \rightarrow \Pi$. That is, there may be certain sets of problems of equivalent complexity with respect to $\rightarrow$. We will first identify several such *equivalence classes* within $\mathcal{S}$, and replace each of them by a single representative. We will then be able to define $\rightarrow$ as a proper *partial ordering* on the reduced class $\mathcal{S}$.

**Lemma 1** *Any two problems* $1, capc|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *and* $1, \circ|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. Obvious. □

We represent any such pair of problems by the latter one.

**Lemma 2** *Any three problems* $\Pi = R, capc|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$, $\Pi' = R, cap\infty|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$, *and* $\Pi'' = R, \circ|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. We will show that $\Pi \propto \Pi''$, $\Pi'' \propto \Pi'$, and $\Pi' \propto \Pi$. Clearly $\Pi$ is a special case of $\Pi''$. An instance of $\Pi''$ can be viewed as an instance of $\Pi'$ in which the speed of a server is equal to the speed of the corresponding server in $\Pi''$ if it carries no more items than the capacity of that server, and 0 if its load is higher. Finally, $\Pi' \propto \Pi$ when we give the servers in $\Pi$ a common capacity equal to the number of rides, which imposes no limitation. □

We will represent any such triple of problems by the third one.

Lemmas 1 and 2 show that some combinations of server type and capacity are redundant. To exclude these, we merge the sets $B_1$ and $B_2$ into the set $B_9 = \{(1, cap1), (1, cap\infty), (1, \circ), (P, cap1), (P, capc), (P, cap\infty), (P, \circ), (Q, cap1), (Q, capc), (Q, cap\infty), (Q, \circ), (R, cap1), (R, \circ)\}$.

**Lemma 3** *Any two problems* $\Pi = \beta_1, \beta_2|s = t, \beta_4, pmtn, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = \beta_1, \beta_2|s = t, \beta_4, \circ, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. There is no point in preempting a ride of length 0. □

We will represent any such pair of problems by the non-preemptive version.

Since the combination of coinciding sources and destinations and preemption will not occur, we merge the sets $B_3$ and $B_5$ into the set $B_{10} = \{S,\ T,\ s = t,\ (\circ, \circ),\ (S, pmtn),\ (T, pmtn),\ (\circ, pmtn)\}$. Every problem can now be written in the form $\beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|\beta_8$.

The equivalences proved below involve entries from more than one field.

**Lemma 4** *Any two problems* $\Pi = \beta_1, \beta_2|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = \beta_1, \beta_2'|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. If sources and destinations coincide, capacities are irrelevant. $\square$

Because of Lemma 4, we will not specify $\beta_2$ if $\beta_3$ is equal to $s = t$.

**Lemma 5** *Any two problems* $\Pi = Q, \beta_2|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = R, \beta_2|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. The servers always travel empty. $\square$

The problem with uniform parallel servers will represent any such pair.

**Lemma 6** *Any two problems* $\Pi = \beta_1, \beta_2|s = t, \beta_4, \beta_5, \circ|\beta_7|C_{\max}$ *and* $\Pi' = \beta_1, cap\infty|T, \beta_4, \beta_5, \circ|\beta_7|C_{\max}$ *with* $\beta_1 \in \{1, P, Q\}$ *and* $\beta_4 \in \{\circ, r_j\}$ *are equivalent.*

PROOF. Since the servers in $\Pi'$ have infinite capacity, they simply collect all items before returning to the origin, which is $T$. $\square$

**Lemma 7** *Any two problems* $\Pi = \beta_1, \beta_2|s = t, \beta_4, \beta_5, \circ|\beta_7|\beta_8$ *and* $\Pi' = \beta_1, cap\infty|S, \beta_4, \beta_5, \circ|\beta_7|\beta_8$ *with* $\beta_1 \in \{1, P, Q\}$ *and* $\beta_4 \in \{\circ, d_j\}$ *are equivalent.*

PROOF. Similar to the proof of Lemma 6. $\square$

The problem with coinciding source and destination will represent any pair of equivalent problems identified by Lemma 6 or 7.

**Lemma 8** *Any two problems* $\Pi = 1, cap\infty|\beta_3, \beta_4, pmtn, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = 1, cap\infty|\beta_3, \beta_4, \circ, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. Because there is only one server, hand-over is not possible. Preempting a ride is not useful since the server can carry all rides at no additional cost. □

The non-preemptive problem will represent any such pair.

**Lemma 9** *Any two problems* $\Pi = \beta_1, \beta_2|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|C_{\max}$ *and* $\Pi' = \beta_1, \beta_2|\beta_3', \beta_4', \beta_5, \beta_6|\beta_7|C_{\max}$ *with*

$$\beta_3' = \begin{cases} T & \text{if } \beta_3 = S, \\ S & \text{if } \beta_3 = T, \\ \beta_3 & \text{otherwise,} \end{cases}$$

$$\beta_4' = \begin{cases} r_j & \text{if } \beta_4 = d_j, \\ d_j & \text{if } \beta_4 = r_j, \\ \beta_4 & \text{otherwise,} \end{cases}$$

*are equivalent.*

PROOF. Consider the decision version of both problems, in which, given a value $D$, we ask if a solution with objective value $C_{\max} \leq D$ exists. For any instance of $\Pi$, construct an instance of $\Pi'$ as follows:

– reverse all rides, i.e., interchange the source and destination of every ride;

– replace every time-window $(r_j, d_j)$ by $(D - d_j, D - r_j)$;

– reverse every arc in the precedence graph.

For each feasible solution of $\Pi$, we obtain a feasible solution of $\Pi'$ by reversing the paths of the servers. Hence, $\Pi \propto \Pi'$. $\Pi' \propto \Pi$ follows by symmetry. □

The deadline version will represent any such pair. If there is a choice, the common source version will be used.

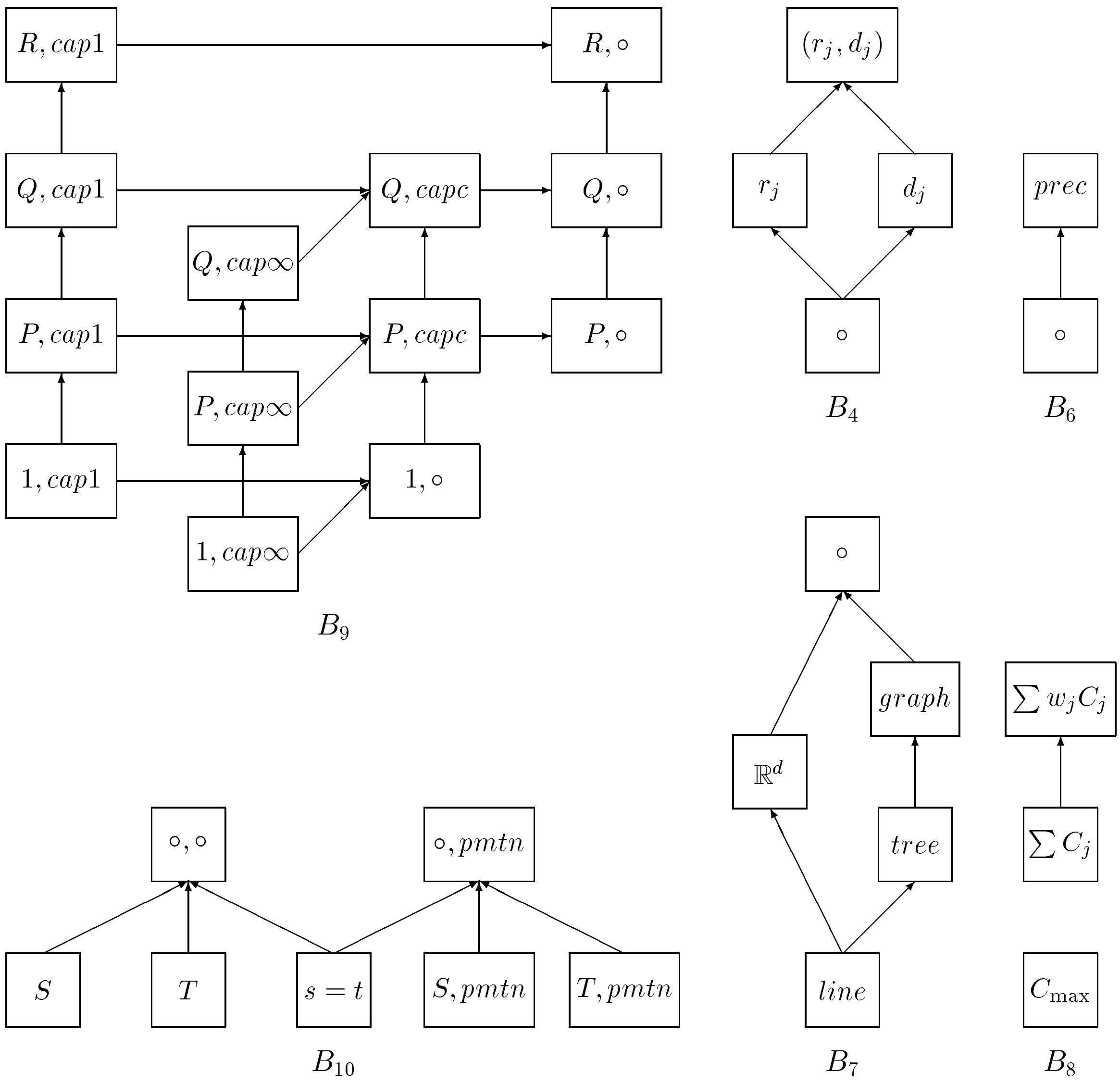


Figure 1: The Digraphs that Define a Partial Ordering on $\mathcal{S}$

By application of the above lemmas, we reduce the number of problem types in the class $\mathcal{S}$ by 48%, from 15,360 to 7,930.

In Section 2 we defined the sets $B_1, \ldots, B_8$ of possible values for the entries in every field. In this section we replaced $B_1$ and $B_2$ by $B_9$ and $B_3$ and $B_5$ by $B_{10}$ in order to eliminate some of the redundancy in $\mathcal{S}$. The sets $B_4, B_6, \ldots, B_{10}$ and relations between their elements are illustrated by the six digraphs in Figure 1. The nodes are the elements of $B_i$ $(i = 4, 6, \ldots, 10)$. There is an arc from one element to another if the latter is a direct generalization of the former.

We use these digraphs to define the partial ordering $\rightarrow$ on $\mathcal{S}$. For two problems $\Pi = \beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|\beta_8$ and $\Pi' = \beta'_9|\beta'_{10}, \beta'_4, \beta'_6|\beta'_7|\beta'_8$ in $\mathcal{S}$, we have $\Pi \rightarrow \Pi'$ if either $\beta_i = \beta'_i$ or there is a directed path from $\beta_i$ to $\beta'_i$ for $i = 4, 6, \ldots, 10$.

We augment the relation $\rightarrow$ by taking into account the following relationship between makespan and latency problems.

**Lemma 10** *For any two problems* $\Pi = \beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|C_{\max}$ *and* $\Pi' = \beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|\sum C_j$ *with* $\beta_4 \in \{d_j, (r_j, d_j)\}$*, we have* $\Pi \propto \Pi'$.

PROOF. Consider the decision version of $\Pi$, in which, for a given $D$, we ask for the existence of a solution with $C_{\max} \leq D$. For any instance of $\Pi$, create an instance of $\Pi'$ by taking identical rides, but change all deadlines larger than $D$ to $D$. $\square$

As said before, the relation $\rightarrow$ defined here is properly included in the reducibility relation $\propto$. It is worth pointing out that $\rightarrow$, with the exception of the augmentation defined by Lemma 10, implies more than just reducibility. It is *approximation preserving*, since our arguments are based on simple inclusions, which do not affect the objective value. Hence, if $\Pi \rightarrow \Pi'$, with the noted exception, then a lower bound on the polynomial-time approximability of $\Pi$ holds for $\Pi'$ as well, and a performance bound for $\Pi'$ applies to $\Pi$ too.

# 4 The Program DARCLASS

The program DARCLASS has been written to determine the computational complexity of the members of the class $\mathcal{S}$ of dial-a-ride problems. For any two problems $\Pi, \Pi' \in \mathcal{S}$ the program knows whether or not $\Pi \rightarrow \Pi'$, where $\rightarrow$ is the partial ordering defined in Section 3.

The input of DaRClass consists of a class $\mathcal{I}^*$ of known easy problems and a class $\mathcal{I}^!$ of known hard problems. Using the input and the partial ordering, DaRClass partitions $\mathcal{S}$ into three classes $\mathcal{S}^*$, $\mathcal{S}^!$, and $\mathcal{S}^?$ of easy, hard, and open problems, respectively:

$$\begin{aligned}
\mathcal{S}^* &= \{\Pi \in \mathcal{S} \mid \exists \Pi' \in \mathcal{I}^* : \Pi \to \Pi'\},\\
\mathcal{S}^! &= \{\Pi \in \mathcal{S} \mid \exists \Pi' \in \mathcal{I}^! : \Pi' \to \Pi\},\\
\mathcal{S}^? &= \mathcal{S} \backslash (\mathcal{S}^* \cup \mathcal{S}^!).
\end{aligned}$$

For correct input, $\mathcal{S}^* \cap \mathcal{S}^! = \emptyset$, as otherwise it would follow that P = NP, contradicting the assumption made in Section 1 and invalidating all of this research.

DaRClass also determines four subclasses of problems that are minimal or maximal within their class with respect to the partial ordering and the current state of knowledge:

$$\begin{aligned}
\mathcal{S}^*_{\max} &= \{\Pi \in \mathcal{S}^* | \nexists \Pi' \in \mathcal{S}^* \backslash \{\Pi\} : \Pi \to \Pi'\};\\
\mathcal{S}^?_{\min} &= \{\Pi \in \mathcal{S}^? | \nexists \Pi' \in \mathcal{S}^? \backslash \{\Pi\} : \Pi' \to \Pi\};\\
\mathcal{S}^?_{\max} &= \{\Pi \in \mathcal{S}^? | \nexists \Pi' \in \mathcal{S}^? \backslash \{\Pi\} : \Pi \to \Pi'\};\\
\mathcal{S}^!_{\min} &= \{\Pi \in \mathcal{S}^! | \nexists \Pi' \in \mathcal{S}^! \backslash \{\Pi\} : \Pi' \to \Pi\}.
\end{aligned}$$

The maximal easy and minimal hard problems represent the essential results, which imply all others. They bound the open problems from below and from above. The minimal and maximal open problems are obvious targets for further research. As long as $\mathcal{S}^?$ is not empty, $\mathcal{S}^*_{\max}$ and $\mathcal{S}^!_{\min}$ may change. Each time an open problem is resolved, the borderlines move closer.

DaRClass finally determines two subclasses of $\mathcal{S}^*_{\max}$ and $\mathcal{S}^!_{\min}$. An easy problem $\Pi$ is *borderline easy* if every $\Pi'$ for which $\Pi \to \Pi'$ is hard. A hard problem $\Pi$ is *borderline hard* if every $\Pi'$ for which $\Pi' \to \Pi$ is easy. These problems determine part of the exact borderline between the easy and the hard problems. If there are no more open problems left, all maximal easy problems are borderline easy, and all minimal hard problems are borderline hard. The status of these problems is no longer subject to change. Their number is in some sense a measure of the progress that has been made towards closing the gap.

In principle, the status of every problem can be determined by hand, on the basis of $\mathcal{I}^*$, $\mathcal{I}^!$, and $\to$. The size of $\mathcal{S}$ and the complexity of $\to$ make manual operations awkward, however. DaRClass has been designed to provide automated assistance.

DaRClass applies standard operations on a directed graph with the problem class $\mathcal{S}$ as its vertex set and an arc $(\Pi, \Pi')$ whenever $\Pi \to \Pi'$. It first computes the transitive closure of

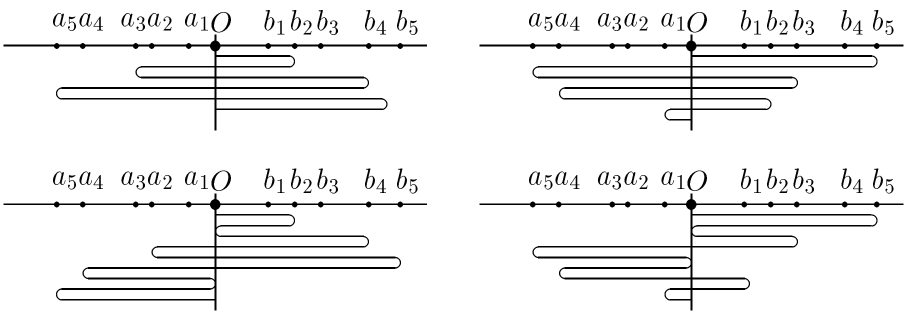


Figure 2: Possible Spiral Forms of Optimal Solutions to Easy Problems on the Line

the graph. Some care is needed in order to handle properly the unique representative of each strongly connected component corresponding to an equivalence class. Given $\mathcal{I}^*$ and $\mathcal{I}^!$, it then partitions the vertex set into three subsets and determines their sources and sinks.

# 5 Maximal Easy Problems

The Online Supplement to this paper gives a complete documentation of all of the polynomial-time algorithms and NP-hardness proofs that represent the state of the art. In this section we discuss three characteristic polynomial-time algorithms. The next section gives a representative selection of six NP-hardness proofs.

Many of the easy problems that have the line as their metric space have in common that an optimal solution has a certain "spiral" structure that allows for solution by dynamic programming. The dynamic program can often be solved in polynomial time, but not always. For example, the problem $1|s = t, d_j|line|\sum C_j$ is hard, although an optimal solution has a spiral structure (Afrati et al. 1986); see Section 7.

Figure 2 shows various kinds of spiral structures that can occur. The horizontal axis is the line on which the problem is defined. The spiral represents the path of the server. For example, if a solution has the shape of the upper left spiral, the server never turns around at a point between the origin and a point where it turned before. If we know that an optimal solution has one of these spiral structures, we only have to determine the turning points of the server in an optimal solution. In many cases, this can be done in polynomial time.

We will take the problem $1, cap\infty|T|line|\sum w_j C_j$ as an example, show that an optimal solution has the shape of the lower left spiral in Figure 2, and give the dynamic program that determines the turning points in an optimal solution. Our algorithm requires time $O(n^3)$, where $n$ is the number of rides.

**Theorem 1** *$1, cap\infty|T|line|\sum w_j C_j$ can be solved in time $O(n^3)$.*

PROOF. Label the sources of the rides on the left-hand side of the origin from right to left, $a_1, \ldots, a_p$, and those on the right-hand side of the origin from left to right, $b_1, \ldots, b_q$. The total number of rides is $p + q = n$. Let $a_0 = b_0 = O$. For any $x$, let $d(O, x)$ be the distance between $O$ and $x$.

Since the server has infinite capacity, there is an optimal solution in which every item is picked up as soon as its source is visited. Hence, if the server reaches $a_i$, all rides with source between $O$ and $a_i$ either have been completed already, or will be completed the first time the server is back in the origin. For the rides on the right-hand side of the origin the same argument holds. Therefore, an optimal solution has the shape of the lower left spiral in Figure 2.

An optimal solution can be computed by the following recursion. We define $V(a_i, b_j)$ as a lower bound on the optimum solution value, consisting of the minimum cost of serving all rides in $[a_i, b_j]$ plus the current time times the weighted number of unserved rides. Clearly, the optimum is equal to $V(a_p, b_q)$.

We initialize by setting $V(0, 0) = 0$, and compute $V(a_i, b_j)$ recursively by

$$\begin{aligned} &V(a_i, b_j) = \\ &\min\{\min_{0\le h\le j-1}\{V(a_i, b_h) + 2d(O, b_j)(\textstyle\sum_{i<k\le p} w_{a_k} + \sum_{h<k\le q} w_{b_k})\}, \\ &\qquad \min_{0\le g\le i-1}\{V(a_g, b_j) + 2d(O, a_i)(\textstyle\sum_{g<k\le p} w_{a_k} + \sum_{j<k\le q} w_{b_k})\}\}. \end{aligned}$$

To see why this is correct, note that the first minimand, for fixed $h$, consists of two components: $V(a_i, b_h) + 2d(O, b_j)\sum_{h<k\le j} w_{b_k}$, which is the cost of serving the requests in $[a_i, b_j]$, and $2d(O, b_j)(\sum_{i<k\le p} w_{a_k} + \sum_{j<k\le q} w_{b_k})$, which is the addition to the lower bound on the cost of serving the requests outside that interval, caused by the additional traveling time.

We have to compute $O(n^2)$ values of $V$, each requiring time $O(n)$. Thus, the entire dynamic program takes time $O(n^3)$. $\square$

Another basic problem is $1, cap\infty||line|C_{\max}$. An optimal solution has the nice structure shown in Figure 3.

**Theorem 2** (Feuerstein and Stougie 2001) *$1, cap\infty||line|C_{\max}$ can be solved in time $O(n \log n)$.*

PROOF. We first show that an optimal solution has the structure of Figure 3. Let $X$ and $Y$ be the left and right extreme points, and let $x$ and $y$ be the other turning points, where

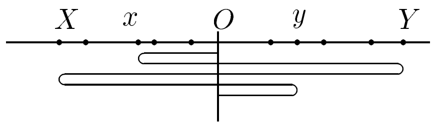


Figure 3: Form of an Optimal Solution (Theorem 2)

$X \leq x \leq 0 \leq y \leq Y$. First note that the server has to visit $X$ and $Y$ at least once, and that all rides that have a source and destination on the same side of the origin can be served while visiting the extreme points. Hence, turning around without crossing the origin is a waste of time. Suppose without loss of generality that the server visits $Y$ before $X$. Since the server can pick up all sources on the right-hand side while moving to $Y$, and since it has to go all the way to $X$ afterwards, it will not visit the right-hand side before the trip to $Y$ in an optimal solution. So an optimal tour starts with $O \to x \to O \to Y \to O$; it may be that $x = 0$. At time $2x + 2Y$ the server is back in the origin. It then only has to visit $X$ and deliver rides that have their source in $(X, x)$. Among these rides, let $y$ be the rightmost destination; if $y < 0$, set $y = 0$. A trivial lower bound on the makespan is $2x + 2Y + 2X + 2y$. This lower bound is attained if the tour has the claimed structure.

We now show how to determine an optimal tour among the tours of this structure. As shown above, for every $x$ we can determine the corresponding $y$ and the makespan $2x + 2Y + 2X + 2y$. Hence, we have to consider no more than $n + 1$ combinations $(x, y)$. After sorting the rides in time $O(n \log n)$, we determine all these combinations in time $O(n)$ and find an optimal solution by taking the minimum over the makespan values. □

Our third easy problem is $1, cap1|S, d_j, prec||C_{\max}$. In solving it, we use the striking similarity that exists between many scheduling and dial-a-ride problems. Celebrated scheduling algorithms such as Smith's shortest processing time rule (Smith 1956) and Jackson's earliest due date rule (Jackson 1955) can sometimes be used to solve dial-a-ride problems too. The algorithm presented below is a straightforward extension of the solution method for the scheduling problem $1|prec, d_j|C_{\max}$ due to Lawler and Moore (1969).

**Theorem 3** *$1, cap1|S, d_j, prec||C_{\max}$ can be solved in time $O(|A| + n \log n)$, where $A$ is the arc set of the precedence graph and $|A|$ denotes its cardinality.*

Proof. We first modify the problem by reversing the direction of every ride $j$ and adding $d(O, t_j)$ to the deadline $d_j$. Since the server has capacity 1, it has to visit the origin between every two rides. The transformation therefore changes neither the set of feasible solutions nor their values.

We now modify the deadlines once more to account for the precedence constraints. For each arc $(i, j)$ of the precedence digraph, we may set

$$d_i = \min\{d_i, d_j - 2d(O, t_j)\}$$

without changing the feasible solution set. The deadline modification can be carried out in time proportional to the number of arcs. Infeasibility is detected as soon as a deadline becomes negative.

We finally sort the rides in order of non-decreasing deadlines. This yields a solution that satisfies the precedence constraints and that, when the server is never idle, has minimum makespan. $\square$

# 6 Minimal Hard Problems

We next give a selection of the NP-hardness proofs that lead to the current state of the art. The core problems from which the reductions have been made are listed below.

PARTITION (Karp 1972): Given a finite set $N = \{1, \ldots, n\}$ and a weight $a_j \in \mathbb{Z}^+$ for every $j \in N$ such that $\sum_{j \in N} a_j = 2b$, does there exist a subset $N' \subseteq N$ such that $\sum_{j \in N'} a_j = b$?

3-PARTITION (Garey and Johnson 1975): Given a set $N = \{1, \ldots, 3n\}$ and a weight $a_j \in \mathbb{Z}^+$ for every $j \in N$ such that $\sum_{j \in N} a_j = nb$ and $b/4 < a_j < b/2$ for all $j \in N$, can $N$ be partitioned into $n$ subsets $N_1, \ldots, N_n$ such that $\sum_{j \in N_i} a_j = b$ for $1 \leq i \leq n$?

HAMILTONIAN PATH (Karp 1972): Given a graph $G = (V, E)$, does $G$ contain a Hamiltonian path?

CIRCULAR ARC COLORING (Garey et al. 1980): Given a finite set of intervals on the boundary of a circle and an integer $k$, can the intervals be colored with at most $k$ colors such that no two overlapping intervals have the same color?

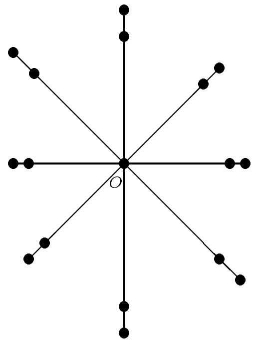


Figure 4: Spider Graph (Theorem 4)

MINIMUM VERTEX FEEDBACK (Karp 1972): Given a digraph $D = (V, A)$ and an integer $k$, does there exist a subset $U \subseteq V$ such that $|U| \leq k$ and the induced subdigraph on the vertex set $V \backslash U$ is acyclic?

SHORTEST COMMON SUPERSEQUENCE (Middendorf 1994): Given a finite set $L$ of equal-length strings over a two-symbol alphabet $\sigma$ and an integer $k$, does there exist a string $l$ over $\sigma$ of length at most $k$ such that each string in $L$ is a subsequence of $l$?

All of these problems are NP-complete in the strong sense, with the exception of PARTITION, which is NP-complete in the ordinary sense but solvable in pseudopolynomial time.

**Theorem 4** PARTITION $\propto$ $1|s = t, d_j|tree|C_{\max}$.

PROOF. For any instance of PARTITION we define an instance of the dial-a-ride problem as follows. Let $c = 2b$. We define a spider graph with $n$ legs; see Figure 4. For every element $j$ ($j \in N$), we introduce two zero-length rides $A_j$ and $B_j$, both on the $j$th leg of the spider. Ride $A_j$ is at distance $ca_j$ from the root and has deadline $2\sum_{i<j}(c+1)a_i + ca_j$. Ride $B_j$ is at distance $(c+1)a_j$ from the root and has deadline $\infty$ for $1 \leq j \leq n-1$ and $d_{B_n} = 4cb - (c-1)a_n + 2b$ for $j = n$. The deadlines are chosen such that each ride $A_j$ must be served immediately after $A_{j-1}$ or $B_{j-1}$. The deadline of $A_j$ allows for serving all $B_i$ ($i = 1, \ldots, j-1$) before $A_j$. However, the deadline of $B_n$ is chosen such that $2c\sum_{j=1}^{n-1} a_j + (c+1)a_n = d_{B_n} - 2b$. Thus, in any feasible solution, we must serve the rides $A_1, A_2, \ldots, A_n, B_n$ in this order. Moreover, for a subset $N' \subseteq N\backslash\{n\}$ with $\sum_{j \in N'} a_j \leq b$, we can insert the rides $B_j$ ($j \in N'$) in this order, each $B_j$ directly after $A_j$. Serving the

remaining rides $B_j$ ($j \notin N', j \neq n$) after $B_n$ gives a solution of makespan

$$2c\sum_{j=1}^{n-1} a_j + 2\sum_{j\in N'} a_j + 2(c+1)\sum_{j\notin N'} a_j \geq 2c\sum_{j=1}^{n-1} a_j + 2b + 2(c+1)b,$$

with equality if and only if we have a yes-instance of PARTITION. □

**Theorem 5** SHORTEST COMMON SUPERSEQUENCE $\propto$ $1|s=t, prec|line|C_{\max}$.

PROOF. Take any instance of SHORTEST COMMON SUPERSEQUENCE. Suppose that $\sigma = \{-1, 1\}$ and that each string in $L$ has length $n$. We define an instance of the dial-a-ride problem as follows. For every string $(l_1, l_2, \ldots, l_n) \in L$ we introduce $2n-1$ zero-length rides: one ride in each $l_i$ ($i = 1, \ldots, n$) and $n-1$ rides in $O$; these rides are subject to a chain-like precedence relation of the form $l_1 \to O \to l_2 \to O \to \ldots \to l_{n-1} \to O \to l_n$.

Suppose that there is a string $l$ of length $k$ that contains each string in $L$ as a subsequence. If we transform $l$ into a chain of $2k-1$ rides by inserting $k-1$ rides in $O$ as above, then we obtain a feasible solution to the dial-a-ride problem of length $2k$; note that the server starts and ends in the origin. Conversely, if the dial-a-ride problem has a solution of length $2k$, then the sequence of visits to the points 1 and $-1$ defines a string of length $k$ that provides a yes-answer to SHORTEST COMMON SUPERSEQUENCE.

Hence, the precedence-constrained traveling salesman problem on the line is hard, even if the precedence relation is a collection of chains. □

**Theorem 6** MINIMUM VERTEX FEEDBACK $\propto$ $1, cap\infty||tree|C_{\max}$.

PROOF. Take any instance of MINIMUM VERTEX FEEDBACK. Let $|V| = n$. We define an instance of the dial-a-ride problem as follows. The tree is a star graph with vertex set $\{O\} \cup V$ and $n$ unit-length edges $\{O, i\}$ ($i \in V$). For every arc $(i, j) \in A$ we introduce a ride from $i$ to $j$.

Suppose there is a subset $U \subseteq V$ of size at most $k$ such that $D' = (V \backslash U, A)$ is acyclic. Consider a solution for the dial-a-ride problem that first visits the vertices in $U$ in any order, then the vertices in $V \backslash U$ according to a topological ordering, and finally the vertices in $U$ again. Such a tour, starting and ending in $O$, is feasible and has length at most $2n + 2k$.

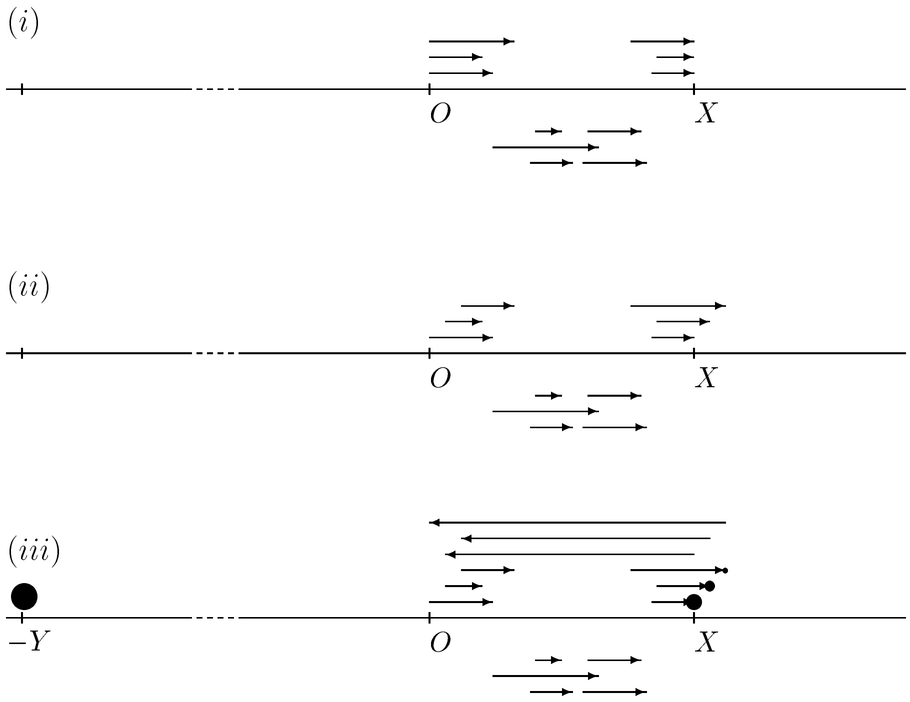


Figure 5: Construction of an Instance of $1, cap1||line|\sum C_j$ (Theorem 7). The Black Dots Denote the Sets of Zero-Length Rides. The Bigger the Dot, the More Rides are Located at the Same Point.

Conversely, suppose that the dial-a-ride problem has a solution of length at most $2n + 2k$ with $k \leq n$. Then at most $k$ vertices are visited more than once. Let $U$ be the set of such vertices. It follows that $D' = (V \backslash U, A)$ is acyclic. Hence, we have a yes-instance of MINIMUM VERTEX FEEDBACK if and only if the dial-a-ride instance has a solution of length at most $2n + 2k$. □

**Theorem 7** CIRCULAR ARC COLORING $\propto 1, cap1||line|\sum C_j$

PROOF. For any instance of CIRCULAR ARC COLORING we define an instance of the dial-a-ride problem as follows. Let $n$ be the number of intervals on the circle and let $X$ be its perimeter. We cut the circle at a point where $k$ intervals overlap; we may assume without loss of generality that such a point exists. We assume that both parts of every interval cut have length at least $k$; otherwise we scale the problem appropriately. The point where the circle is cut becomes point $O$ on the line; every other point on the circle gets a coordinate on the line equal to its distance in the clockwise direction from the cut point. For each interval on the line thus obtained we introduce a ride from left to right. This gives $n + k$ rides, including $k$ that start in $O$ and $k$ that end in $X$. We call these the *$A$-rides*; see Figure $5(i)$.

Label the rides such that ride $A_i$ has its source in $O$, ride $A_{k+i}$ has its destination in $X$, and $A_i$ and $A_{k+i}$ originate from the same interval on the circle, for $1 \leq i \leq k$. Label the other rides arbitrarily $A_i$, for $2k + 1 \leq i \leq n + k$. Now modify the rides $A_1, \ldots, A_k$ by deleting the initial part of $A_i$ such that it starts in $i - 1$, and modify the rides $A_{k+1}, \ldots, A_{2k}$ by extending $A_{k+i}$ such that it ends in $X+i-1$; see Figure $5(ii)$.

We next introduce the *$B$-rides* and the *$C$-rides*. For $i = 1, \ldots, k$, we define $(k-i+1)b$ zero-length $B$-rides in point $X+i-1$, for a value of $b$ to be determined later. For $i = 1, \ldots, k-1$, we define ride $C_i$ starting in $X + i - 1$ and ending in $i$, and we define ride $C_k$ starting in $X + k - 1$ and ending in $O$. Finally, we define $d$ zero-length $D$-rides, located in $-Y$, for values of $d$ and $Y$ to be determined later. See Figure $5(iii)$.

Suppose that the instance of CIRCULAR ARC COLORING has a $k$-coloring. The following tour serves all rides. Start in $O$. Serve $A_1$ and all rides that correspond to intervals with the same color as the interval corresponding to $A_1$. By construction, all such rides have been served when we reach the destination of $A_{k+1}$. Then serve the $kb$ $B$-rides in that point and serve $C_1$, which ends where $A_2$ starts. Next serve $A_2$ and all rides corresponding to intervals with the same color. Then serve the $(k-1)b$ $B$-rides in $X + 1$ and serve $C_2$. Continuing in this way, we serve all rides of type $A$, $B$, and $C$, and we are back in the origin at time $2kX$. We reach the $D$-rides at time $2kX + Y$.

In this solution, the $D$-rides contribute $(2kX + Y)d$ to the total completion time, the $B$-rides contribute $Xkb + (3X - 1)(k - 1)b + (5X - 2)(k - 2)b + \cdots + ((2k - 1)X - (k - 1))b$, and the rides of type $A$ and $C$ contribute at most $2kX(n+2k)$ together. Hence, the objective value of our solution is no more than

$$Z = (2kX + Y)d + Xkb + (3X - 1)(k - 1)b+$$

$$(5X - 2)(k - 2)b + \cdots + ((2k - 1)X - (k - 1))b + 2kX(n + 2k).$$

Now suppose that no $k$-coloring exists. In that case, the tour described above cannot serve all $A$-rides. We will choose the parameters $b$, $d$, and $Y$ such that every tour that deviates from this tour has a total completion time larger than $Z$. As a result, the dial-a-ride problem has a solution of value at most $Z$ if and only if a $k$-coloring exists.

First set $d = Xkb + (3X - 1)(k - 1)b + (5X - 2)(k - 2)b + \cdots + ((2k - 1)X - (k - 1))b + 2kX(n + 2k) + 1$, to make sure that the $D$-rides are served no later than $2kX + Y$. If they are delayed by one time unit, the total completion time exceeds $Z$.

Now set $Y = kXd + 1$ to make sure that the $D$-rides will be served only after all other rides have been served. Serving a ride after the $D$-rides increases its completion time by at least $2Y$. Serving the $D$-rides earlier yields a gain of at most $2kXd$. So by the choice of $Y$, serving the $D$-rides before any other ride increases the total completion time. This in combination with the choice of $d$ implies that the server must be in the origin at time $2kX$ after having served all rides of type $A$, $B$, and $C$. This forces the server to start with ride $A_1$ and to turn around only if it can start another ride immediately.

To reach our goal, we only have to make sure that $A_{k+1}, \dots, A_{2k}$ are served in the same order as $A_1, \dots, A_k$. Notice that this implies that $A_i$ is served before $A_j$ for $1 \leq i < j \leq k$. Set $b = k(n + 2k)$. Serving the rides in any other order such that the server is still back in the origin at time $2kX$ after having served all rides of type $A$, $B$, and $C$ increases the total completion time of the $B$-rides by at least $(2X - 1)b = (2X - 1)k(n + 2k) > 2kX(n + k)$. As only the $A$-rides can be served earlier in this way, the decrease is less than $2kX(n + k)$. The resulting total completion time is therefore larger than $Z$. □

**Theorem 8** HAMILTONIAN PATH $\propto 1, cap1|pmtn|tree|\sum C_j$.

PROOF. For any instance of HAMILTONIAN PATH with $V = \{1, \dots, n\}$, we define an instance of the dial-a-ride problem as follows. The tree is a star graph with vertex set $\{O, 1, \dots, n, n+1\}$, $n$ unit-length edges $\{O, 1\}, \dots, \{O, n\}$, and an edge $\{O, n + 1\}$ of length $b = 2n^4$. For each leaf vertex $i$ $(i = 1, \dots, n + 1)$ we introduce $c = 2b$ zero-length rides in $i$. For each edge $(i, j) \in E$, we define two rides, one from $i$ to $j$ and one from $j$ to $i$. We claim that there is a solution to the dial-a-ride problem with total completion time at most $Z = (n^2+2n+b)c+2(n+b)(2|E|-(n-2))+n^2$ if and only if a Hamiltonian path exists.

Suppose there is a Hamiltonian path $H$. We first visit each of the vertices $\{1, \dots, n\}$ once in the order in which they appear on $H$, and then vertex $n + 1$. In doing so, we serve the $n - 1$ rides corresponding to the edges of $H$ at cost

$$3 + 5 + \cdots + (2n - 1),$$

and the $c$ zero-length rides in each of the leaf vertices at cost

$$(1 + 3 + \cdots + (2n - 1) + (2n + b))c.$$

We next return to the origin at time $2(n+b)$, and serve the $2|E|-(n-1)$ rides corresponding to the edges that are not on $H$ in any order, at cost

$$2(n+b)(2|E|-(n-1))+3+7+\cdots+4(2|E|-(n-1))-1.$$

Adding these contributions, we find that the total completion time is less than

$$(n^2+2n+b)c+n^2+2(n+b)(2|E|-(n-1))+4n^4<Z,$$

where the last inequality follows from the choice of $b$.

Suppose no Hamiltonian path exists. If fewer than $n-1$ rides corresponding to edges of $E$ are served before $n+1$ is visited, then at least $2|E|-(n-2)$ rides are postponed until after time $2n+2b$, yielding a total completion time larger than $Z$. To prevent this, at least one leaf must be visited twice before all leaves have been visited once. Compared to the previous situation, this increases the costs by at least $2c$, because at least one of the groups of $c$ zero-length rides is delayed by 2 time units, again yielding a total completion time larger than $Z$.

Serving more than $n-1$ rides corresponding to arcs in $E$ before visiting $n+1$ decreases the total completion time by at most $2(n+b)$ per ride. But for every such ride, at least $c$ requests are delayed by 2 time units, so per ride the net increase of the total completion time is $2c-2(n+b)>0$. Therefore, if $\sum C_j \leq Z$, there is a Hamiltonian path.

Clearly, allowing preemption does not change the line of argument. □

**Theorem 9** 3-PARTITION $\propto P, cap1|S|line|\sum w_jC_j$

PROOF. For any instance of 3-PARTITION we define an instance of the dial-a-ride problem with $n$ servers and, for every element $j$ ($j \in N$), a ride $(O, a_j)$ with weight $a_j$. For the sake of argument, suppose that a server serves the rides $(O, a_1)$, $(O, a_2)$, $(O, a_3)$ in that order, without idle time. Its contribution to the weighted sum of completion times is

$$a_1^2+a_2(2a_1+a_2)+a_3(2a_1+2a_2+a_3)=(a_1+a_2+a_3)^2.$$

More generally, consider any solution to the problem, without idle time between the rides. For $i=1,\ldots,n$, let $N_i$ be the index set of the rides served by server $i$, and let $b_i=\sum_{j\in N_i} a_j$ denote their total length. Note that $\sum_{i=1}^n b_i = nb$. The solution has value $\sum w_jC_j = \sum_{i=1}^n b_i^2 \geq nb^2$, with equality if and only if $b_1=\cdots=b_n=b$, i.e., if and only if we have a yes-instance of 3-PARTITION. □

# 7 Summary and Analysis

An early version of DARCLASS (De Paepe 1998) generated 12,600 problem types and classified 218 problems (1.7%) as easy, 9,824 (78.0%) as hard, and 2,558 (20.3%) as open. Application of DARCLASS to the present class $\mathcal{S}$ of 7,930 problems with the current knowledge as input tells us that 180 (2.3%) of these are easy, 7,673 (96.8%) are hard, and 77 (1.0%) remain open. There are 26 maximal easy problems and 68 minimal hard ones. Among these, 19 problems are borderline easy and 49 are borderline hard. Moreover, all problems that are minimal in $\mathcal{S}$ are easy and all problems that are maximal in $\mathcal{S}$ are hard, so that the open problems are encapsulated by the easy and hard ones.

The input to DARCLASS consists of all known complexity results for the class $\mathcal{S}$. It includes what is known for the vehicle routing and machine scheduling problems mentioned at the end of Section 2. There are also a few results for single-server problems on the line or on a tree. We will review these before we analyze the output of DARCLASS.

Atallah and Kosaraju (1988) proved that the stacker crane problem on the line, $1, cap1||line|C_{\max}$, is easy. Frederickson and Guan (1992, 1993) studied "ensemble motion planning" problems, in our notation $1|||C_{\max}$. They showed that $1, cap1||tree|C_{\max}$ is hard, and that the preemptive version, $1, cap1|pmtn|tree|\ C_{\max}$, is easy. Guan (1998) proved that $1||line|C_{\max}$ and $1|pmtn|tree|C_{\max}$ are hard; the latter result follows from the hardness of $1, cap\infty||tree|C_{\max}$ (Theorem 6). Feuerstein and Stougie (2001) showed that $1, cap\infty||line|C_{\max}$ is easy (Theorem 2) and that both $P, cap1|S|line|C_{\max}$ and $P, cap1|T|line|C_{\max}$ are hard. Tsitsiklis (1992) proved that $1|s = t, d_j|line|C_{\max}$ is easy and that the version with time-windows, $1|s = t, (r_j, d_j)|line|C_{\max}$, is hard; the former result follows from our polynomial-time algorithm for $P|s = t, d_j|line|C_{\max}$ (see the Online Supplement to this paper).

As for the latency objective, Afrati et al. (1986) studied the traveling repairman problem on the line. They showed that $1|s = t|line|\sum C_j$ is solvable by dynamic programming in $O(n^2)$ time. For the problem with deadlines, $1|s = t, d_j|line|\sum C_j$, they proved hardness in the ordinary sense, and they gave a pseudopolynomial-time algorithm and a fully polynomial-time approximation scheme. The problem with release times, $1|s = t, r_j|line|\sum C_j$, is open Tsitsiklis (1992). Also the problem where, instead of release times or deadlines, there are *repair times* at the request points is open (Afrati et al. 1986, Tsitsiklis 1992). It can be formulated in terms of the problem $1|s = t|tree|\sum C_j$, where the tree has the special form of a caterpillar graph. Sitters (2002) proved that the general traveling repairman problem on a tree is hard, but his proof does not apply to caterpillar graphs. Tsitsiklis (1992) observed that the traveling repairman problem on the line with both release times and repair times is hard, because of the hardness of the scheduling problem $1|r_j|\sum C_j$ (Lenstra et al. 1977).

Almost all of the problems cited in this brief review turn out to be borderline easy or borderline hard. We refer to the Online Supplement to this paper for details.

In Table 1 we list all of the easy and open problems. To keep the list manageable, we use the following shorthand: $A/B$ denotes alternatives within a problem type, $(A)$ is an optional feature, and $*$ is a wildcard for an arbitrary type of metric or for weights in the objective.

We will examine the overall influence of each characteristic of a dial-a-ride problem on its complexity and point out some interesting open problems.

**Precedence constraints, release times and deadlines.** We have classified all problems with precedence constraints. Only the problems of type $1, cap1|S/T, (pmtn), (d_j), prec| * |C_{\max}$ are easy. The others are hard, including all problems with parallel servers or capacity not equal to 1 or latency objective.

All problems with time-windows, i.e., both release times and deadlines, are hard. Almost all problems with either release times or deadlines have been classified as well: only nine problems with deadlines and only eight with release times are open.

There are easy problems with deadlines. As in the case of precedence constraints these include a class of single-server problems ($E5$ and $E6$) but also one with parallel servers ($E3$). For the latency objective, no problem with release times is known to be easy. (For makespan, some are solvable by symmetry between deadlines and release times.)

The open problems with release times include some of the most challenging open questions in the area: $1|s=t, r_j|line|\sum C_j$ and $1, cap\infty|(S), r_j|line|\sum C_j$.

**Metric spaces.** It turns out that the distinction between *line* and more general spaces is the main breakpoint in the classification. Most of the open problems have the real line as a metric space, the exception being $O16$. Some problems are easy for all metric spaces. Almost all other problems are hard for the variants of metric spaces other than the line. The only exceptions, besides $O16$, are three problems solvable on trees: $E1$, $E9$ and $E14$.

**Sources and destinations.** Not surprisingly, the traveling salesman-like problems with rides with coinciding source and destination behave differently than do problems with more

Table 1: All Easy and Open Problems

| | | |
|---|---|---|
| **Easy problems** | | **180 problems** |
| **s=t rides** | | **10 problems** |
| $E1$ | $1\|s=t\|tree\|C_{\max}$ | 1 |
| $E2$ | $1/P\|s=t\|line\|\sum *C_j$ | 4 |
| $E3$ | $1/P\|s=t,(d_j)\|line\|C_{\max}$ | 4 |
| $E4$ | $Q\|s=t\|line\|C_{\max}$ | 1 |
| **remaining – deadlines** | | **61 problems** |
| $E5$ | $1,cap1\|S/T,(pmtn),d_j,(prec)\|*\|C_{\max}$ | 40 |
| $E6$ | $1,cap1\|S/T,(pmtn),d_j\|*\|\sum C_j$ | 20 |
| $E7$ | $1,cap\infty\|T,d_j\|line\|C_{\max}$ | 1 |
| **remaining – general rides** | | **4 problems** |
| $E8$ | $1,cap1\|(pmtn)\|line\|C_{\max}$ | 2 |
| $E9$ | $1,cap1\|pmtn\|tree\|C_{\max}$ | 1 |
| $E10$ | $1,cap\infty\|\|line\|C_{\max}$ | 1 |
| **remaining – S/T rides** | | **105 problems** |
| $E11$ | $1,cap1\|S,(pmtn),(prec)\|*\|C_{\max}$ | 20 |
| $E12$ | $1,cap1\|S/T,(pmtn)\|*\|\sum *C_j$ | 40 |
| $E13$ | $1\|S,(pmtn)\|line\|C_{\max}$ | 2 |
| $E14$ | $1\|S,pmtn\|tree\|C_{\max}$ | 1 |
| $E15$ | $P,cap1\|S/T,pmtn\|*\|\sum C_j$ | 10 |
| $E16$ | $P/Q/R,cap1\|S/T\|*\|\sum C_j$ | 30 |
| $E17$ | $1,cap\infty\|T\|line\|\sum *C_j$ | 2 |
| **Open problems** | | **77 problems** |
| **s=t rides** | | **6 problems** |
| $O1$ | $Q\|s=t\|line\|\sum *C_j$ | 2 |
| $O2$ | $Q\|s=t,d_j\|line\|C_{\max}$ | 1 |
| $O3$ | $1/P/Q\|s=t,r_j\|line\|\sum C_j$ | 3 |
| **remaining – release times and deadlines** | | **13 problems** |
| $O4$ | $1,cap\infty\|(S/T),r_j\|line\|\sum C_j$ | 3 |
| $O5$ | $P/Q,cap\infty\|S,pmtn,r_j\|line\|\sum C_j$ | 2 |
| $O6$ | $P/Q,cap\infty\|T,pmtn,d_j\|line\|C_{\max}$ | 2 |
| $O7$ | $P/Q/R,cap1\|S/T,pmtn,d_j\|line\|C_{\max}$ | 6 |
| **remaining – general rides** | | **14 problems** |
| $O8$ | $1,(cap1)\|pmtn\|line\|\sum *C_j$ | 4 |
| $O9$ | $1\|pmtn\|line\|C_{\max}$ | 1 |
| $O10$ | $1,cap\infty\|\|line\|\sum *C_j$ | 2 |
| $O11$ | $P/Q,cap\infty\|(pmtn)\|line\|C_{\max}$ | 4 |
| $O12$ | $P/Q/R,cap1\|pmtn\|line\|\sum C_j$ | 3 |
| **remaining – S/T rides** | | **44 problems** |
| $O13$ | $1\|S/T,(pmtn)\|line\|\sum *C_j$ | 8 |
| $O14$ | $P/Q,cap\infty\|T,(pmtn)\|line\|\sum *C_j$ | 8 |
| $O15$ | $P/Q/R,(cap1/c)\|S,pmtn\|line\|C_{\max}$ | 8 |
| $O16$ | $Q/R,cap1\|S/T,pmtn\|*\|\sum C_j$ | 20 |

general rides. Most of them are also hard, only ten are easy, and six open. Perhaps the most interesting result obtained during our work on DaRClass is the hardness of the traveling repairman problem on a tree, $1|s = t|tree|\sum C_j$ (Sitters 2002), which settled a long-standing open problem.

Most of the other easy or open problems have rides with the same source or the same destination. Only four problems with general rides are known to be easy, and fifteen of them are open.

**Preemptions.** As in machine scheduling, allowing preemption makes a problem harder to classify. Among the open problems there are eleven where preemption is irrelevant (with rides of type $s = t$ or a server of type $1, cap\infty$), ten pairs in which both the preemptive version and the non-preemptive version are open, and 46 problems where only the preemptive version is open.

The makespan problems on trees $E9$ and $E14$ are, surprisingly, the only problems where the preemptive version is easy and the non-preemptive is hard.

For all open preemptive problems with release times or deadlines or general rides, the non-preemptive version is hard. The remaining problems, $O15$ and $O16$, are interesting in relation to their counterparts from machine scheduling.

The easiest problem of type $O15$, $P, cap1|S, pmtn|line|C_{\max}$, is easy if all the rides are on the same side of the origin, as preemptive scheduling results then apply. With rides on both sides, however, it may be impossible to balance the work among the servers. The non-preemptive version is hard.

The latency problems $O16$ are the only instances where a preemptive dial-a-ride problem may be hard while the non-preemptive version is easy. Their counterparts in scheduling are easy for uniform machines (Gonzalez 1977) and hard for unrelated ones (Sitters 2001), but neither result seems to be of much use here.

**Number, type, and capacity of servers.** Not surprisingly, parallel servers and capacities other than 1 tend to make the problems hard. Most of the easy problems involve unit capacity. At present only two problems with general capacity and four with infinite capacity are easy, and they all have a single server.

For parallel servers, there is never a difference between $Q$ and $R$ in our current classification, and there is never a difference between common capacity $c$ and general capacity (which could only matter for $P$ or $Q$).

The open problems $O1$ and $O2$ exhibit a striking difference between $P$ and $Q$. For identical servers the problems are trivial: it is optimal to use two servers, one for either direction. For a single server the problems are solved by dynamic programming. For uniform servers one can create complex instances where, for example, any number of fastest servers turns any given number of times.

Another problem with a different classification for $P$ and $Q$ is $O16$ discussed above. For identical servers it is easy, since preemptions do not help. We still have not found a problem, however, where the $Q$ variant is provably hard and the $P$ variant is easy.

Problem $O11$ without preemption is easy for a constant number of servers, as each server has to try only polynomially many reasonable tours.

**Objective.** As in machine scheduling, makespan and latency have little in common. There are 22 open problems with the makespan objective.

For the latency objective, it is interesting to examine the influence of weights. There are several classes of problems where adding weights to the objective causes hardness: $E6$, $E15$, and $E16$. There are twelve pairs of open problems where both the weighted and unweighted version have not been classified, and 31 problems where the unweighted version is open and the weighted one is hard; these involve a reduction from PARTITION and exponential weights. Weighted problems seem to be easier to classify: we have no case where the unweighted version is easy and the weighted one is open.

## Acknowledgments

We are grateful to the referees, whose comments helped us to improve the exposition. Jiri Sgall was partially supported by grants 201/01/1195 of GA CR and A1019901 of GAAV CR, and cooperative research grant KONTAKT-ME476/CCR-9988360-001 from the NSF and MŠMT CR. Willem de Paepe was employed by the Department of Technology Management of the Technische Universiteit Eindhoven, where this research was performed.

## References

Afrati, F., S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, N. Papakonstantinou. 1986. The complexity of the travelling repairman problem. *Theoretical Informatics and*

*Applications* **20** 79–87.

Ascheuer, N., S.O. Krumke, J. Rambau. 2000. Online dial-a-ride problems: minimizing the completion time, *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science **1770**. Springer, Berlin, Germany. 639–650.

Atallah, M.J., S.R. Kosaraju. 1988. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing* **17** 849–870.

Conway, R.W., W.L. Maxwell, L.W. Miller. 1967. *Theory of Scheduling*. Addison-Wesley, Reading, MA.

Desrochers, M., J.K. Lenstra, M.W.P. Savelsbergh. 1990. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research* **46** 322–332.

Feuerstein, E., L. Stougie. 2001. On-line single server dial-a-ride problems. *Theoretical Computer Science* **268** 91–105.

Frederickson, G.N., D.J. Guan. 1992. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing* **21** 1130–1152.

Frederickson G.N., D.J. Guan. 1993. Non-preemptive ensemble motion planning on a tree. *Journal of Algorithms* **15** 29–60.

Frederickson, G.N., M.S. Hecht, C.E. Kim. 1978. Approximation algorithms for some routing problems. *SIAM Journal on Computing* **7** 178–193.

Garey, M.R., D.S. Johnson. 1975. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing* **4** 397–411.

Garey, M.R., D.S. Johnson, G.L. Miller, C.H. Papadimitriou. 1980. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods* **1** 216–227.

Gonzalez, T. 1977. Optimal mean finish time preemptive schedules. Technical Report 220, Computer Science Department, The Pennsylvania State University, University Park, PA.

Graham, R.L., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5** 287–326.

Guan, D.J. 1988. Routing a vehicle of capacity greater than one. *Discrete Applied Mathe-*

*matics* **81** 41–57.

Jackson, J.R. 1955. Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles, CA.

Karp, R.M. 1972. Reducibility among combinatorial problems. R.E. Miller, J.W. Thatcher, eds. *Complexity of Computer Computations*, Plenum Press, New York. 85–103.

Kendall, D.G. 1953. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *Annals of Mathematical Statistics* **24** 338–354.

Lageweg, B.J., J.K. Lenstra, E.L. Lawler, A.H.G. Rinnooy Kan. 1982. Computer-aided complexity classification of combinatorial problems. *Communications of the ACM* **25** 817–822.

Lawler, E.L., J.M. Moore. 1969. A functional equation and its application to resource allocation and sequencing problems. *Management Science* **16** 77–84.

Lenstra, J.K., A.H.G. Rinnooy Kan. 1981. Complexity of vehicle routing and scheduling problems. *Networks* **11** 221–227.

Lenstra, J.K., A.H.G. Rinnooy Kan, P. Brucker. 1977. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* **1** 343–362.

Middendorf, M. 1994. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science* **125** 205–228.

Paepe, W.E. de 1998. *Computer-aided complexity classification of dial-a-ride problems.* M.Sc. Thesis, Department of Economics and Econometrics, University of Amsterdam, The Netherlands.

Sitters, R.A. 2001. Two NP-hardness results for preemptive minsum scheduling of unrelated parallel machines. K. Aardal, B. Gerards, eds. *Integer Programming and Combinatorial Optimization.* Lecture Notes in Computer Science **2081**. Springer, Berlin, Germany. 396–405.

Sitters, R.A. 2002. The minimum latency problem is NP-hard for weighted trees. W.J. Cook, A.S. Schulz, eds. *Integer Programming and Combinatorial Optimization.* Lecture Notes in Computer Science **2337**. Springer, Berlin, Germany. 230–239.

Smith, W.E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3** 59–66.

Tsitsiklis, J.N. 1992. Special cases of traveling salesman and repairman problems with time windows. *Networks* **22** 263–282.

*Accepted by W. David Kelton; received June 2002; revised January 2003; accepted April 2003.*