Special Cases of Traveling Salesman and Repairman Problems with Time Windows*

John N. Tsitsiklis
Laboratory for Information and Decision Systems and the Operations Research Center
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Consider a complete directed graph in which each arc has a given length. There is a set of jobs, each job i located at some node of the graph, with an associated processing time h_i, and whose execution has to start within a prespecified time window $[r_i, d_i]$. We have a single server that can move on the arcs of the graph, at unit speed, and that has to execute all of the jobs within their respective time windows. We consider the following two problems: (a) minimize the time by which all jobs are executed (traveling salesman problem) and (b) minimize the sum of the waiting times of the jobs (traveling repairman problem). We focus on the following two special cases: (a) The jobs are located on a line and (b) the number of nodes of the graph is bounded by some integer constant B. Furthermore, we consider in detail the special cases where (a) all of the processing times are 0, (b) all of the release times r_i are 0, and (c) all of the deadlines d_i are infinite. For many of the resulting problem combinations, we settle their complexity either by establishing NP-completeness or by presenting polynomial (or pseudopolynomial) time algorithms. Finally, we derive algorithms for the case where, for any time t, the number of jobs that can be executed at that time is bounded.

I. INTRODUCTION

As is well known, the traveling salesman problem (TSP) is NP-complete even if we are restricted to grid-graphs [9]. Furthermore, introducing time constraints into the problem (such as time windows) can only make it harder [13]. For this reason, time-constrained variants of the TSP have been primarily studied from a pragmatic point of view, for the purpose of designing branch and bound algorithms with practically acceptable running time; see the surveys [3] and [15]. On the other hand, there is some hope of obtaining polynomial time algorithms for time-constrained variations of the TSP when we restrict it to

*Research supported by the C. S. Draper Laboratories under contract DL-H-404164, with matching funds provided by the National Science Foundation, under Grant ECS-8552419.

NETWORKS, Vol. 22 (1992) 263–282
© 1992 by John Wiley & Sons, Inc.
CCC 0028-3045/92/030263-20$04.00

special cases. In the first special case that we consider, the jobs† to be executed are placed on a straight-line segment. A few problems of this type have been studied in [12], where one particular variation was shown to be polynomially solvable; some other variations, however, were left open and we study them in Sections 2 and 3. In practice, the straight-line problem can arise in the case of a server visiting customers located along a highway or in the case of a ship visiting ports along a coastline.

In a variant of the TSP, instead of minimizing the total time it takes to execute all jobs, one tries to minimize the sum of their waiting times. This is known as the traveling repairman problem (TRP) and has been studied in [1]. While this problem is also NP-complete, in general, it was shown in [1] that some progress is possible for the straight-line case. We discuss this problem further in Section 4. Note that the TRP captures the waiting costs of a service system from the customers' point of view. As such, it can be used to model numerous types of service systems. Also, the cost function used in the TRP is the same as *flowtime* (also known as sum of completion times), which is a most commonly employed performance measure in scheduling theory.

Problem Definition and Notation

Let Z_0 be the set of nonnegative integers. Consider a complete directed graph G. To each arc (i, j), we associate a length $c(i, j) \in Z_0$. We assume that the arc lengths satisfy the triangle inequality and we use the convention $c(i, i) = 0$ for all i. There are n jobs $J_1, \ldots, J_n$, with job J_i located at a node x_i of G. To each job J_i, we associate a time-interval $[r_i, d_i]$, with $r_i \in Z_0$ and $d_i \in Z_0 \cup \{\infty\}$. We refer to r_i as a *release time* and to d_i as a *deadline*. The interval $[r_i, d_i]$ is called a *window* and $d_i - r_i$ is its *width*.

Each job has an associated *processing time* $h_i \in Z_0$. We have a single server that starts at time 0 at a certain node x^* of G. The server can move on the arcs of the graph at unit speed or stay in place. It must start the execution of each job J_i during the time interval $[r_i, d_i]$. Furthermore, if the execution of job J_i starts at time t_i, then the server cannot leave node x_i or start the execution of another job before time $t_i + h_i$.

Formally, an instance of the problem consists of G, x^*, the job locations $x_1, \ldots, x_n$, the lengths $c(i, j)$, and the nonnegative integers r_i, d_i, h_i, $i = 1, \ldots, n$. A feasible solution (also called a feasible schedule) for such an instance is a permutation π of $\{1, \ldots, n\}$, indicating that the jobs are executed in the order $J_{\pi(1)}, \ldots, J_{\pi(n)}$, and a set of nonnegative integer times t_i, $i = 1, \ldots, n$, indicating the time that the execution of each job J_i is started. Furthermore, we have the following feasibility constraints:

(a) $t_i \in [r_i, d_i]$, for all i;
(b) $c(x^*, x_{\pi(1)}) \leq t_{\pi(1)}$;
(c) $t_{\pi(i)} + h_{\pi(i)} + c(x_{\pi(i)}, x_{\pi(i+1)}) \leq t_{\pi(i+1)}$, for $i = 1, \ldots, n - 1$.

†We use the term "jobs" synonymously with the term "cities" that is often used in describing the TSP.

TABLE I. The complexity of special cases of Line-TSPTW (n is the number of jobs).

	Zero processing times	General processing times
No release times or deadlines	Trivial	Trivial
Release times only	$O(n^2)$ [12]	NP-complete (Theorem 3) ?
Deadlines only	$O(n^2)$ (Theorem 1)	?
General time windows	Strongly NP-complete (Theorem 2)	Strongly NP-complete [6]

We are interested in the following two problems:

TSPTW: Find a feasible solution for which $\max_i(t_i + h_i)$ is minimized.
TRPTW: Find a feasible solution that minimizes the total waiting time $\Sigma_{i=1}^{n} t_i$ or, equivalently, $\Sigma_{i=1}^{n} (t_i - r_i)$.

Summary of Results

In the first special case we consider, the job locations x_i are integers and $c(x_i, x_j) = |x_i - x_j|$. We refer to the resulting problems as Line-TSPTW and Line-TRPTW. Results for these two problems are presented in Sections 2–4. They are summarized in Tables I and II, together with earlier available results and references. Here, question marks indicate that there are still some open problems. For example, it is not known whether Line-TSPTW with release times only and general processing times is strongly NP-complete or pseudopolynomial.

In the next special case that we consider, we impose a bound B on the number of nodes of the graph and study the complexity as a function of the

TABLE II. The complexity of special cases of Line-TRPTW (n is the number of jobs).

	Zero processing times	General processing times
No release times or deadlines	$O(n^2)$ [1]	?
Release times only	?	Strongly NP-complete [10]
Deadlines only	NP-complete [1] pseudopolynomial	NP-complete [1] ?
General time windows	Strongly NP-complete (by Theorem 2)	Strongly NP-complete [6]

TABLE III. The complexity of TSPTW and TRPTW when the number of nodes is bounded by B; general processing times are allowed (the number of jobs is n).

	B-TSPTW	B-TRPTW
No release times or deadlines	$O(B^2 2^B + n)$ [8]	$O(B^2 n^B)$ (Theorem 7)
Release times only	$O(B^2 n^B)$ (Theorem 6)	Strongly NP-complete even if $B = 1$ [10]
Deadlines only	$O(B^2 n^B)$ [11]	?
General time windows	Strongly NP-complete even if $B = 1$ [6]	Strongly NP-complete even if $B = 1$ [6]

other problem parameters. We refer to the resulting problems as B-TSPTW and B-TRPTW. It turns out that if the processing times are zero for all jobs, then B-TSPTW and B-TRPTW can be solved by polynomial time algorithms, fairly similar to the algorithms of [11]. (Of course, the running time of these algorithms is exponential in B.) If we allow for different processing times, the picture is more varied, as seen in Table III. These results are proved in Sections 5–7.

We note that problems with a bound B on the number of nodes arise naturally in the context of manufacturing systems. For example, suppose that we have a single machine and that each node of the graph corresponds to a different job-type (or batch). We can then interpret the length of an arc as the "set-up" time spent by the machine before it can start processing jobs of a different type. In this context, it is natural to assume that the number of job-types is bounded by a small constant B, while allowing for a large number of jobs to be executed over a long time horizon. Scheduling problems incorporating set-up times when switching between job-types have been studied in [2] and [11]. The context for [2] was provided by a computer system that can run several different programs; set-up times here correspond to the time needed to load appropriate programs or compilers into the main memory.

Finally, in Section 8, we consider another special case that seems to arise often in practice. In particular, we assume that there exists an integer D such that the number of jobs J_i for which $t \in [r_i, d_i]$ is bounded by D for all t. Note that if the time windows of different jobs are not large, and if these time windows are spread fairly uniformly in time, then such an assumption is likely to hold with a reasonably small value of D. We refer to the resulting problems as TSPTW(D) and TRPTW(D). We establish that the natural dynamic programming algorithm has complexity $O(nD^2 2^D)$ for TSPTW(D) and $O(TD^2 2^D)$ for TRPTW(D), where T is an upper bound on the duration of an optimal schedule. This agrees with experimental results reported in [5] for some related problems. We finally show that TRPTW(D) is NP-complete even if $D = 2$, and, therefore, it is very unlikely that our pseudopolynomial time algorithm can be made polynomial.

II. THE LINE-TSPTW WITH ZERO PROCESSING TIMES

Throughout this and the next two sections, we focus on problems defined on the line. In particular, each job location x_i is an integer. Furthermore, in this section, we assume that the processing time h_i of each job is equal to 0. The problem is clearly trivial if $r_i = 0$ and $d_i = \infty$ for all i. It was shown in [12] that the problem can be solved in $O(n^2)$ time (via dynamic programming) for the special case where $d_i = \infty$ for all i, that is, when we only have release times. We now show that the same complexity is obtained for the special case where $r_i = 0$ but the deadlines are arbitrary.

Theorem 1. The special case of Line-TSPTW in which $h_i = r_i = 0$ for all i can be solved in $O(n^2)$ time.

Proof. Since the processing and release times are zero, it follows that each job is immediately executed the first time that the server visits its location. In particular, if the server has visited locations a and b, with $a \le b$, then all jobs whose location belongs to the interval $[a, b]$ have been executed.

Let us assume that the job locations have been ordered so that $x_1 \le x_2 \le \cdots \le x_n$. Let i^* be such that $x^* = x_{i^*}$. (The existence of such an i^* can be assumed without any loss of generality; for example, we can always insert an inconsequential job at location x^*.) We assume that $|x_i - x^*| \le d_i$ for all i; otherwise, the problem is infeasible. Let us fix some i, j with $1 \le i \le i^* \le j \le n$. Consider all schedules in which the server visits location x_i for the first time at time t and has executed all jobs in the interval $[x_i, x_j]$ within their respective deadlines. Let $V^-(i, j)$ be the smallest value of t for which this is possible, and let $V^-(i, j) = \infty$ if no such schedule exists. We define $V^+(i, j)$ similarly, except that we require that the server visits location x_j (instead of x_i) for the first time at time t. Note that according to the preceding verbal definition we have the convention $V^-(i^*, j) = \infty$ for $i^* < j$. (The reason is that the server starts at x_{i^*} and therefore the requirement that x_j be visited before the first visit of x_{i^*} is impossible.) Similarly, $V^+(i, i^*) = \infty$ for $i < i^*$. We then have the following recursion:

$$V^+(i^*, j) = x_j - x^*, \qquad j > i^*,$$

$$V^-(i, i^*) = x^* - x_i, \qquad i < i^*,$$

$$U^+(i, j) = \min\,[V^+(i, j-1) + x_j - x_{j-1},\ V^-(i, j-1) + x_j - x_i], \qquad i < i^* < j,$$

$$V^+(i, j) = \begin{cases} U^+(i, j), & \text{if } U^+(i, j) \le d_j, \\ \infty, & \text{otherwise}, \end{cases} \qquad i < i^* < j,$$

$$U^-(i, j) = \min\,[V^-(i+1, j) + x_{i+1} - x_i,\ V^+(i+1, j) + x_j - x_i], \qquad i < i^* < j,$$

$$V^-(i, j) = \begin{cases} U^-(i, j), & \text{if } U^-(i, j) \leq d_i, \\ \infty, & \text{otherwise}, \end{cases} \qquad i < i^* < j.$$

Using this recursion, we can compute $V^+(1, n)$ and $V^-(1, n)$ in $O(n^2)$ time. The minimum of these two numbers is the cost of an optimal solution, with a value of infinity indicating an infeasible instance. An optimal solution can be easily found by backtracking. ■

Thus, Line-TSPTW, with zero processing times, is polynomial when we have only release times or only deadlines. Interestingly enough, the problem becomes difficult, when both release times and deadlines are present, as we show next. This settles an open problem posed in [12] and [15].

In our NP-completeness proof, we use the well-known NP-completeness of the satisfiability problem 3SAT defined as follows: We are given n Boolean variables $v_1, \ldots, v_n$ and m clauses $C_1, \ldots, C_m$ in these variables, with three literals per clause.* The problem consists of deciding whether there exists a truth assignment to the variables such that all clauses are satisfied.

We will also need a modified version of 3SAT, which we call MSAT. Here different clauses correspond to different time stages, and the variables are allowed to change truth values from one stage to another; however, the only change allowed is from T (true) to F (false). Furthermore, when the truth value of a variable changes, we allow it to be undefined for (at most) one stage in between.

Formally, an instance of MSAT is defined as follows: We have nK variables $v_i(k)$, $i = 1, \ldots, n$, $k = 1, \ldots, K$, and K clauses $D_1, \ldots, D_K$, with three literals per clause. Furthermore, for each k, only the variables $v_i(k)$, $i = 1, \ldots, n$, or their negations, can appear in clause D_k. An *extended assignment* is one whereby each variable $v_i(k)$ is assigned a truth value in the set $\{T, F, *\}$. The problem consists of deciding whether there exists an extended assignment that satisfies the following constraints:

(a) If $k < K$ and $v_i(k) \neq T$, then $v_i(k+1) = F$.
(b) For every k, either there exists some i for which $v_i(k) = *$, or the truth assignment is such that the clause D_k is satisfied.

Notice that, by definition, an "undefined" variable $v_i(k) = *$ can take care of clause D_k even if the variable $v_i(k)$ does not appear in D_k. Furthermore, notice that there is no point in letting $v_i(1) = F$, for any i. The reason is that letting $v_i(1) = *$ is at least as good as $v_i(1) = F$, for the purpose of satisfying clause D_1, and imposes the same constraint $v_i(2) = F$. We thus add to the definition of MSAT the requirement $v_i(1) \neq F$ for every i. For the same reason, we also require that $v_i(K) \neq T$ for every i.

Lemma 1. MSAT is NP-complete.

*We only consider instances in which a variable can appear in a clause at most once, either unnegated or negated.

Proof. We will reduce 3SAT to MSAT. Given an instance $(v_1, \ldots, v_n, C_1, \ldots, C_m)$ of 3SAT, let $K = m(n+1)$. The clauses in the instance of MSAT are essentially the same as $C_1, \ldots, C_m$, but repeated $n+1$ times. Formally, if $k = i + ml$, where $i = 1, \ldots, m$, and $l = 0, \ldots, n$, and if $C_k = (a \text{ OR } b \text{ OR } c)$, then $D_k = [a(k) \text{ OR } b(k) \text{ OR } c(k)]$. Here, each one of a, b, c is one of the variables v_j or $\bar{v}_j$, $j = 1, \ldots, n$.

Suppose that we have a YES instance of 3SAT and let us fix a satisfying assignment. We then define an extended assignment for the instance of MSAT by letting $v_i(k) = v_i$ for all i, k. [Keeping in line with the discussion preceding the lemma, we need the following exceptions: If $v_i = F$, let $v_i(1) = *$; also, if $v_i = T$, let $v_i(K) = *$.] Clearly, this has all the desired properties and we have a YES instance of MSAT.

Conversely, suppose that we have a YES instance of MSAT, and let us fix an extended assignment to the variables $v_i(k)$ with the desired properties. We split the set $\{1, \ldots, K\}$ into $(n+1)$ *segments*, each segment consisting of m consecutive integers. Property (a) in the definition of MSAT easily implies that, for any fixed i and for each segment, the value of $v_i(k)$ is fixed to either T or F, with the possible exception of one segment. [The latter would be a segment on which $v_i(k)$ changes from T or $*$ to F.] By throwing away n segments (one segment for each i), we are left with a segment on which the value of $v_i(k)$ stays constant for all i. We then assign to v_i the value of $v_i(k)$ on that segment, for all i. Since each clause D_k is satisfied, and since each clause C_k is "represented" in each segment, it follows that the assignment to the v_i's satisfies all of the clauses in the instance of 3SAT. ∎

We now move to the proof of our main result. In this proof, we find it convenient to visualize an instance of the problem in terms of a two-dimensional diagram (see, e.g., Fig. 1), where the horizontal axis corresponds to time and the vertical axis corresponds to location in space. The time window associated to each job is represented by a horizontal segment connecting points (r_i, x_i) and (d_i, x_i). The path traversed by the server corresponds to a trajectory whose slope belongs to $\{-1, 0, 1\}$.

Theorem 2. Testing an instance of Line-TSPTW for feasibility is strongly NP-complete, even in the special case where the processing times h_i are zero.

Proof. The problem is clearly in NP. We will reduce MSAT to Line-TSPTW. Let there be given an instance of MSAT with variables $v_i(k)$, $i = 1, \ldots, n$, $k = 1, \ldots, K$, and clauses $D_1, \ldots, D_K$. We will construct an equivalent instance of Line-TSPTW. For easier visualization, we will actually construct the two-dimensional representation [in (x, t)-space] of the latter instance.

We first construct a convenient "gadget" $G_i(k)$ associated to each variable $v_i(k)$. Suppose that k is even and that neither $v_i(k)$ nor its negation appears in the clause D_k. Then, the gadget $G_i(k)$ is as shown in Figure 1(a). It consists of four jobs $V_i(k)$, $\overline{V}_i(k)$, $J_i(k)$, $J_i'(k)$. The windows for jobs $J_i(k)$ and $J_i'(k)$ have

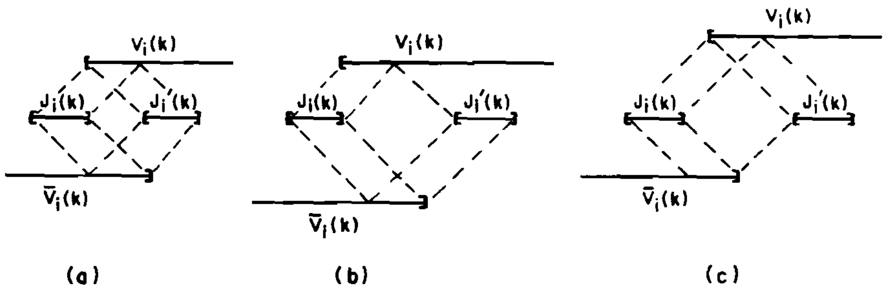

FIG. 1. The "gadget" $G_i(k)$ when k is even. (a), (b), and (c), correspond to the cases where $v_i(k)$ does not appear in D_k, or appears in D_k unnegated, or appears in D_k negated, respectively. Brackets are used to indicate release times and deadlines. The dashed lines have slope 1 or -1 and are shown only to indicate the relative positioning of the jobs and some paths that the server could possibly follow.

width 1. There is a release time for job $V_i(k)$ and a deadline for job $\bar{V}_i(k)$. For now, we leave the deadline of $V_i(k)$ and the release time of $\bar{V}_i(k)$ unspecified. They will be determined later when we connect the gadgets together.

Note that between the execution of jobs $J_i(k)$ and $J_i'(k)$ the server has enough time to execute job $V_i(k)$ (by moving northeast and then southeast) or job $\bar{V}_i(k)$ (by moving southeast and then northeast), but *not both*. We interpret the server's choice as an (extended) truth assignment to $v_i(k)$: Executing $V_i(k)$ or $\bar{V}_i(k)$ corresponds to $v_i(k) = T$ or $v_i(k) = F$, respectively; executing neither corresponds to $v_i(k) = *$. We say that the *delay* in executing $J_i(k)$ [or $J_i'(k)$] is 0 or 1 depending on whether $J_i(k)$ [or $J_i'(k)$] is executed at the beginning or the end of its time window. We say that there is a *delay reduction* at gadget $G_i(k)$ if the delay of $J_i(k)$ is 1 and the delay of $J_i'(k)$ is 0. It is clear from Figure 1(a) that a delay reduction is possible only if $v_i(k) = *$.

Suppose now that $v_i(k)$ appears unnegated in clause D_k. Then, the corresponding gadget $G_i(k)$ is as shown in Figure 1(b). The main difference from the previous case [cf. Fig. 1(a)] is that a delay reduction is possible not only if $v_i(k) = *$ but also if $v_i(k) = T$. Finally, if $v_i(k)$ appears negated in D_k, $G_i(k)$ is constructed in a symmetrical fashion [see Fig. 1(c)]. In this case, a delay reduction is possible if $v_i(k) = *$ or $v_i(k) = F$.

So far, we have discussed the case where k is even. If k is odd, the gadgets $G_i(k)$ are constructed by taking the gadgets of Figure 1 and turning them upside down.

The construction of the instance of Line-TSPTW is completed by indicating how to connect together the gadgets $G_i(k)$. This is done as follows (see Fig. 2 for an illustration):

(a) We have certain jobs $Q_0, Q_1, \ldots, Q_K$, with zero window width, which force the server to visit certain points in the (x, t) diagram.

(b) If k is even and $k \neq K$, Q_k and Q_{k+1} are located so that the server has to visit $J_1(k + 1)$ with unit delay and $J_n(k + 1)$ with zero delay.

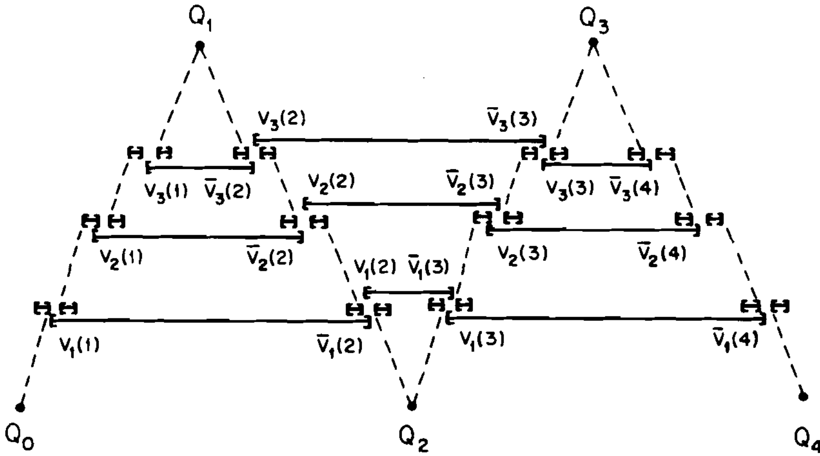

FIG. 2. Connecting the gadgets $G_i(k)$ to form an instance of Line-TSPTW. In this figure, $K = 4$ and $n = 3$. Dashed lines have slope 1 or -1 and are shown only to indicate the relative positioning of the jobs.

(c) If k is odd and $k \neq K$, Q_k and Q_{k+1} are located so that the server has to visit $J_n(k+1)$ with unit delay and $J_1(k+1)$ with zero delay.

(d) If the server executes $J_{i+1}(k)$ [respectively, $J_{i-1}(k)$] right after $J_i'(k)$, the delay of $J_{i+1}(k)$ [respectively, $J_{i-1}(k)$] is equal to the delay of $J_i'(k)$.

(e) We identify job $V_i(k)$ with job $\overline{V}_i(k+1)$, $k = 1, \ldots, K-1$. The release time (respectively, the deadline) of that job is determined by the structure of the gadget $G_i(k)$ [respectively, $G_i(k+1)$].

(f) The jobs $\overline{V}_i(1)$ and $V_i(K)$, $i = 1, \ldots, n$, are removed from the gadgets in which they should have appeared.

It is not hard to see that the gadgets can indeed be connected as described above. Furthermore, this can be done with the largest integer in the instance of Line-TSPTW being bounded by a polynomial function of n and K.

As a consequence of conditions (b), (c), and (d), the constructed instance has the following key property. For the server to execute all of the jobs $J_i(k)$ and $J_i'(k)$, $i = 1, \ldots, n$, as well as the jobs Q_{k-1} and Q_k, within their respective deadlines, it is necessary and sufficient that there be a delay reduction between the execution of Q_{k-1} and Q_k. This happens if and only if either:

(a) $v_i(k) = *$ for some i, in which case a delay reduction occurs by skipping both jobs $V_i(k)$ and $\overline{V}_i(k)$; or

(b) the clause D_k becomes true. For example, if $v_i(k)$ appears unnegated in D_k, setting $v_i(k) = T$, that is, executing $V_i(k)$, causes a delay reduction [see Fig. 1(b)].

We conclude that the requirement of executing the jobs $J_i(k)$, $J_i'(k)$ $i = 1, \ldots, n$, and Q_{k-1}, Q_k within their respective time windows is equivalent to satisfying clause D_k, in the sense required for the problem MSAT.

We now prove the equivalence of the constructed instance to the original instance of MSAT. Suppose that we have a YES instance of MSAT and consider a satisfying extended truth assignment. We construct a feasible schedule for Line-TSPTW. In this schedule, the jobs $Q_0, Q_1, \ldots, Q_K$ are executed in sequence. In between Q_{k-1} and Q_k, we execute $J_1(k), J'_1(k), \ldots, J_n(k), J'_n(k)$, if k is odd. (If k is even, the same jobs are executed but in a different order.) We execute job $V_i(1)$, between $J_i(1)$ and $J'_i(1)$, if and only if $v_i(1) = T$. For $1 < k < K$, we execute job $V_i(k)$ [respectively, job $\overline{V}_i(k)$] between $J_i(k)$ and $J'_i(k)$, if and only if $v_i(k) = T$ [respectively, $v_i(k) = F$]. Finally, we execute job $\overline{V}_i(K)$, between $J_i(K)$ and $J'_i(K)$, if and only if $v_i(K) = F$. For $k = 1, \ldots, K - 1$, if job $V_i(k)$ is not executed between $J_i(k)$ and $J'_i(k)$, then $v_i(k) \neq T$, which implies that $v_i(k + 1) = F$, and job $\overline{V}_i(k + 1)$ is executed between $J_i(k + 1)$ and $J'_i(k + 1)$. Since $V_i(k)$ and $\overline{V}_i(k + 1)$ are the same job, all jobs of the type $V_i(k)$ are executed within their deadlines. Furthermore, for each k, since clause D_k of MSAT is satisfied, there is a delay reduction in the path from Q_{k-1} to Q_k, and job Q_k can be executed at the required time.

Conversely, given a feasible solution of Line-TSPTW, we construct an extended truth assignment, by reversing the argument in the preceding paragraphs. In particular, if no job $V_i(k)$ or $\overline{V}_i(k)$ is executed between $J_i(k)$ and $J'_i(k)$, we set $v_i(k) = *$. Each clause D_k is satisfied since Q_k is executed in time. Furthermore, suppose that $k < K$ and $v_i(k) \neq T$. Then, $V_i(k)$ is not executed between $J_i(k)$ and $J'_i(k)$. Thus, $\overline{V}_i(k + 1)$ is executed between $J_i(k + 1)$ and $J'_i(k + 1)$, which implies that $v_i(k + 1) = F$, as desired.

Finally, since all of the data in the constructed instance are bounded by a polynomial in n and K, we have established strong NP-completeness of Line-TSPTW. ■

III. LINE-TSPTW WITH GENERAL PROCESSING TIMES

The special case in which $r_i = 0$ and $d_i = \infty$ for each i is trivial. If there are both release times and deadlines, the problem is strongly NP-complete even if all jobs were at the same location [6] (see also [7], p. 236); note that this is a pure scheduling problem. If we only have deadlines ($r_i = 0$), the problem is open. Finally, if we only have release times, our next result shows that the problem is NP-complete. However, we have not been able to establish strong NP-completeness or the existence of a pseudopolynomial algorithm.

Theorem 3. The special case of Line-TSPTW in which $d_i = \infty$ for all i is NP-complete.

Proof. We use the NP-completeness of the PARTITION problem defined as follows: An instance is described by m positive integers $z_1, \ldots, z_m$. Let $K = \sum_{i=1}^{m} z_i$. The problem is to determine whether there exists a set $S \subset \{1, 2, \ldots, m\}$ such that $\sum_{i \in S} z_i = K/2$.

Given an instance $(z_1, \ldots, z_m)$ of PARTITION, we will construct an instance of Line-TSPTW. The server starts at $x^* = 0$. There are m jobs $J_1, \ldots,$

J_m. Each job J_i is at location $x_i = iH$, has a release time $r_i = (2i - 1)H$, and a processing time equal to H, where $H = 2K + 1$. There is also a job J_{m+1} at location $(m + 1)H$ with zero processing time and release time r_{m+1} equal to $(2m + 1)H + 3K/2$. There is also a job J_{m+2}, at location 0, with zero processing time and with release time $r_{m+2} = r_{m+1} + K/2 + (m + 1)H = (3m + 2)H + 2K$. We finally introduce m jobs $J'_1, \ldots, J'_m$. Job J'_i is at location $x'_i = x_i - z_i = iH - z_i$ and has a processing time equal to z_i and a release time r'_i equal to $r_i + H = 2iH$. The question is whether there exists a feasible schedule for the server in which the execution of all jobs has been completed by time r_{m+2}.

Suppose that we have a YES instance of PARTITION and let $S \subset \{1, \ldots, m\}$ be such that $\Sigma_{i \in S}\, z_i = K/2$. Consider the following schedule: The server executes jobs $J_1, \ldots, J_{m+2}$ in this order. Furthermore, for each $i \in S$, the server executes J'_i between J_i and J_{i+1}. To execute J'_i, $i \in S$, the server travels z_i units backward, spends z_i time units for processing, and then retraces the z_i units that were traveled backward. Thus, executing each J'_i, $i \in S$, causes a delay of $3z_i$ time units. Therefore, the server will reach job J_{m+1} at time $(2m + 1)H + 3\, \Sigma_{i \in S}\, z_i = (2m + 1)H + 3K/2 = r_{m+1}$. Thus, J_{m+1} can be executed immediately. Then, on the way back (from J_{m+1} to J_{m+2}), the server pauses to execute the jobs J'_i for $i \notin S$. Hence, it will reach and execute job J_{m+2} at time $r_{m+1} + (m + 1)H + \Sigma_{i \notin S}\, z_i = r_{m+1} + (m + 1)H + K/2 = r_{m+2}$, and we have a YES instance of Line-TSPTW.

For the other direction, suppose that we have a YES instance of Line-TSPTW, and let use consider a schedule that reaches J_{m+2} at time r_{m+2}. We will show that this schedule must be of the form considered in the preceding paragraph. First, it is clear that J_{m+2} is the last job to be executed. Let S be the set of all i such that job J'_i is executed before J_{m+1}. The server leaves job J_{m+1} no earlier than time r_{m+1}, reaches J_{m+2} no later than time $r_{m+2} = r_{m+1} + (m + 1)H + K/2$, and has to travel a distance of $(m + 1)H$ in between. Thus, there are only $K/2$ time units available for processing on the way from J_{m+1} to J_{m+2}. Since $H = 2K + 1 > K/2$, it follows that all of the jobs $J_1, \ldots, J_m$ have been executed before J_{m+1}. Furthermore,

$$\sum_{i \notin S} z_i \leq K/2. \tag{1}$$

During the schedule, a distance of at least $2(m + 1)H$ has to be traveled (from 0 to job J_{m+1} and back), and a total of $K + mH$ time units have to be spent for processing. Since $r_{m+2} = (3m + 2)H + 2K$, it follows that there is a margin of only K time units that can be "wasted" by following a trajectory other than a straight line. If $i < j$ and job J_i were to be executed after job J_j, the server would have to travel at least $2H$ units more than the minimum required. Since $2H > K$, this is more than the available margin and we conclude that the schedule executes $J_1, \ldots, J_{m+1}$ in their natural order.

Suppose now that the schedule executes job J'_i before job J_i. Then, the execution of job J_i starts later than time $r'_i = 2iH$. From that point on, the server must travel to J_{m+1} and back to 0 [this needs $(m + 1 - i)H + (m + 1)H$ units of

travel time] and spend $(m + 1 - i)H$ time units for processing jobs $J_i, \ldots, J_m$. Thus, the server reaches J_{m+2} later than time $(2i + m + 1 - i + m + 1 + m + 1 - i)H = (3m + 3)H > (3m + 2)H + 2K = r_{m+2}$, contradicting the assumption that J_i' was executed before J_i.

Because of the preceding arguments, the set S determines completely the order in which the jobs are executed. In particular, the server "wastes" a total of $2 \sum_{i \in S} z_i$ time units for excess travel time, in order to execute each job J_i', $i \in S$, after the corresponding job J_i. On the other hand, the available margin for excess traveling is exactly K. Thus, $\sum_{i \in S} z_i \leq K/2$. This, together with Eq. (1), shows that $\sum_{i \in S} z_i = K/2$ and we have a YES instance of PARTITION. ■

IV. THE LINE-TRPTW

Our only new result for the Line-TRPTW is a corollary of Theorem 2. We note that the problem of testing an instance of Line-TRPTW for feasibility is identical to the problem of testing an instance of Line-TSPTW for feasibility. Thus, Theorem 2 implies that feasibility of Line-TRPTW is strongly NP-complete, even in the absence of processing times.

Let us also note that Line-TRPTW with deadlines only and general processing times is NP-complete but it is not known whether it is pseudopolynomial or strongly NP-complete.

V. BOUNDED NUMBER OF LOCATIONS AND ZERO PROCESSING TIMES

We now treturn to the more general problem in which the job locations correspond to the nodes of a complete directed graph G. We consider the problems B-TSPTW and B-TRPTW, which are the special cases of TSPTW and TRPTW in which the number of nodes of G is at most B. The problems considered in this and the next two sections resemble and have a partial overlap with those considered in [11]. The main principle behind our positive (polynomial time) results has been introduced in [11] and is the following: If for each location we can determine ahead of time the order in which the jobs are to be executed, then dynamic programming leads to a polynomial time algorithm. Our results and complexity estimates are farily similar to those in [11]. A difference is that [11] assumes the release times to be zero.

Our first result provides a polynomial time algorithm for the B-TSPTW.

Theorem 4. If the processing times are zero, then B-TSPTW can be solved in time $O(B^2 n^B)$.

Proof. We will use an algorithm of the dynamic programming type. Let us sort the jobs at a node i in order of increasing release times, and let J_{ik} be the kth such job. Let r_{ik} and d_{ik} be the release time and deadline of J_{ik}, respectively. Let K_i be the number of jobs at node i. Since processing times are zero, we can

assume that a job is executed the first time subsequent to its release time that the server visits its location. In particular, for each location, jobs are executed in order of increasing release times.

We will say that the server is in state $s = (i, n_1, n_2, \ldots, n_B)$ if the following hold:

(a) The server is currently executing the n_ith job at node i.
(b) For each node j, the server has executed exactly the first n_j jobs at that node, and each such job has been executed by its respective deadline.

Note that we have $O(Bn^B)$ states. For each state s, let $V(s)$ be the earliest time at which the server could reach state s, with $V(s) = \infty$ if it is impossible for the server to reach state s. We have $V(x^*, 0, \ldots, 0) = 0$ and $V(i, 0, \ldots, 0) = c(x^*, i)$ for every $i \neq x^*$. Furthermore, $V(i, n_1, \ldots, n_B)$ is equal to

$$U(i, n_1, \ldots, n_B) =$$

$$\max \{r_{i,n_i}, \min_{j=1,\ldots,B} [V(j, n_1, \ldots, n_{i-1}, n_i - 1, n_{i+1}, \ldots, n_B) + c(j, i)]\}$$

if $U(i, n_1, \ldots, n_B) \leq d_{i,n_i}$, and $V(i, n_1, \ldots, n_B) = \infty$ if $U(i, n_1, \ldots, n_B) > d_{i,n_i}$.

The optimal cost of the B-TSPTW problem is given by $\min_i V(i, K_1, \ldots, K_B)$ and an optimal feasible solution can be found by backtracking. Since only $O(B)$ arithmetic operations are needed in order to evaluate each $V(s)$, the complexity estimate follows. ■

In the complexity estimate of Theorem 4, we have ignored an $O(n \log n)$ term corresponding to the preprocessing required in order to sort the jobs at each location. This term is small compared to $O(B^2n^B)$, as long as $B > 1$. This comment also applies to all other complexity estimates in Sections 5–7.

For the B-TRPTW problem, the situation is slightly more complex. Here, a dynamic programming algorithm has to keep track of both time and waiting time, which leads to a pseudopolynomial time algorithm. We show that a polynomial time algorithm is, in fact, possible, if some extra care is exercised. The argument is again similar to that in [11].

Let T be any easily computable upper bound on the duration of an optimal schedule. For example, T could be the value of the largest deadline. Alternatively, if the deadlines are infinite, T could be taken as the largest release time plus a bound on the duration of any tour that visits all nodes of the graph.

Theorem 5. If the processing times are zero, then B-TRPTW can be solved in time $O(B^2n^BT)$, where T is any upper bound on the duration of an optimal schedule and n is the number of jobs. Alternatively, it can be solved in time $O(B^2n^{B^2+1})$.

Proof. We order the jobs at each node and define $s = (i, n_1, \ldots, n_B)$ as in the proof of Theorem 4. We say that the server is at state (t, s) if at time t the server is at location i and has executed exactly the first n_j jobs at node j without violating their deadlines; furthermore, the n_ith job at node i is executed at time t. Let $W(t, s)$ be the minimum possible sum of the already incurred waiting times if the server is at state (t, s); we let $W(t, s) = \infty$ if this is not feasible. The problem of computing $W(t, s)$ for all t and s is a standard dynamic programming problem. We have $O(TBn^B)$ possible states and from each state there are $O(B)$ possible next states. The cost of a transition from a state $(t, j, n_1, \ldots, n_B)$ to a state $(t', i, n_1, \ldots, n_{i-1}, n_i + 1, n_{i+1}, \ldots, n_B)$ is given by $(n - \Sigma_{k=1}^{B} n_k)(t' - t)$. The dynamic programming algorithm solves such a problem in time $O(B^2 n^B T)$; this proves the first part of the result.

Suppose that the server is at state $(t, j, n_1, \ldots, n_B)$ and that its next state is $(t', i, n_1, \ldots, n_{i-1}, n_i + 1, n_{i+1}, \ldots, n_B)$. We claim that if the server is following an optimal schedule, then t' must be equal to $\tau = \max\{r_{i,n_i+1}, t + c(j, i)\}$. Indeed, the problem constraints yield $t' \geq \tau$. Also, if $t' > \tau$, then the server could execute the $(n_i + 1)$st job at location i at time τ, and wait at that location until time t'. The net effect would be a reduction of the waiting time, thus contradicting optimality. We can therefore impose the additional constraint that the only allowed transitions are of the form $(t, s) \rightarrow (t', s')$ where $t' = \max\{r_{i,n_i+1}, t + c(j, i)\}$ for some i, j. We will show shortly that this limits the number of reachable states.

Let ℓ be the length of a path in the graph G, with at most n arcs. Let L be the set of all such ℓ. (We allow paths with repeated nodes. We also include the empty path, so that $0 \in L$.) Let $\mathcal{T} = \{t + \tau \mid t \in L \text{ and } \tau \in \{0, r_1, \ldots, r_n\}\}$. Given that the schedule starts at time 0, and given the nature of the allowed transitions (cf. the preceding paragraph), an easy inductive argument shows that any sequence of transitions leads us to a state of the form (t, s) with $t \in \mathcal{T}$. So, the dynamic programming recursion only needs to be carried out over such states.

Note that each element of L is of the form $\Sigma_{(i,j)} \alpha_{ij} c(i, j)$, where each α_{ij} is an integer bounded by n. Since there are only $B(B - 1)$ values of (i, j) to be considered, we conclude that L has $O(n^{B(B-1)})$ elements. It follows that $\mathcal{T}$ has $O(n^{B^2-B+1})$ elements. Accordingly, the number of states (t, s) to be considered by the dynamic programming algorithm is $O(Bn^B n^{B^2-B+1})$, leading to a total complexity of $O(B^2 n^{B^2+1})$. ∎

VI. THE TSPTW WITH A BOUNDED NUMBER OF LOCATIONS

We now study the B-TSPTW problem for the general case where processing times are arbitrary.

In the special case where there are neither release times nor deadlines, the problem is equivalent to the standard TSP and can be solved in time $O(B^2 2^B + n)$ [8]. (The additive factor of n in the complexity estimate is due to the need to compute the sum of the processing times.)

Next, we consider the case where we only have release times.

Theorem 6. The special case of B-TSPTW in which we only have release times (all deadlines are infinite) can be solved in time $O(B^2n^B)$.

Proof. A simple interchange argument shows that, for each location, the jobs at that location can be executed in order of increasing release times. Once we sort the jobs in order of increasing release times, the argument is identical to the proof of Theorem 4. The state $s = (i, n_1, \ldots, n_B)$ indicates the server's location and the number of jobs that have already been executed at each location. A similar dynamic programming algorithm applies, provided that we properly incorporate the processing times in the dynamic programming equation. ■

For the case where we only have deadlines, the problem can be solved in time $O(B^2n^B)$ by a simple modification of Theorem 3 of [11]. The reason is that the jobs in each location can be executed in order of increasing value of $d_i + h_i$. (This was first proved in [14] for the case $B = 1$.) Finally, if we have both release times and deadlines, the problem is strongly NP-complete even for $B = 1$, as remarked in the beginning of Section 3 [6].

VII. THE TRPTW WITH A BOUNDED NUMBER OF LOCATIONS

Testing for feasibility in the case where we have both release times and deadlines is equally hard as for the B-TSPTW and is therefore strongly NP-complete, even if $B = 1$. For the case where we only have deadlines, the problem is open for $B \geq 2$; it is polynomially solvable if $B = 1$ [14]. In the case where we only have release times, the problem is strongly NP-complete, even for $B = 1$ [10]. The last case to be considered is the subject of our next theorem.

Theorem 7. The special case of B-TRPTW in which there are neither release times nor deadlines can be solved in time $O(B^2n^B)$.

Proof. A simple interchange argument shows that, for each location, the jobs at that location should be executed in order of increasing processing times. We can they say that the server is at state $s = (i, n_1, \ldots, n_B)$ if it is at location i and has already executed, for each location j, the n_j "shortest" jobs located at j. Let $W(s)$ be the least possible total waiting time that has to be incurred before the server can get to state s. There is a total of $O(Bn^B)$ states. From each state s, the server can move to B other states and the dynamic programming recursion for computing each $W(s)$ needs $O(B)$ operations. ■

VIII. BOUNDED NUMBER OF ACTIVE JOBS

We say that a job i is *active* at time t if $t \in [r_i, d_i]$. Let $A(t)$ be the set of all active jobs at time t. In this section, we restrict to instances in which the cardinality of $A(t)$ is bounded by D for all t, where D is some integer constant. We refer to the resulting problems as TSPTW(D) and TRPTW(D).

Theorem 8. TSPTW(D) can be solved in time $O(nD^22^D)$.

Proof. For any $S \subset \{1, \ldots, n\}$ and $i \in \{1, \ldots, n\}$, we say that the server is at state (i, S) at time t if:

(a) We have $i \in S$ and the execution of job J_i starts at time t.
(b) For each $j \in S$, the execution of job J_j was started at some time t_j satisfying $t_j \le t$ and $t_j \le d_j$.
(c) For each $j \notin S$, we have $d_j \ge t$ and the execution of job J_j has not yet started.

We say that (i, S) is *reachable* if it is feasible for the server to get to that state at some time t; let $F(i, S)$ be the minimum possible such time. If (i, S) is not reachable, let $F(i, S) = \infty$. Thus, the optimal cost for our problem is equal to $\min_i F(i, \{1, \ldots, n\})$. Furthermore, $F(i, \{i\})$ is easily computed for each i.

Suppose that (i, S) is reachable and let $S' = S \cup \{j\}$. Then, state (j, S') can be reached via (i, S), at time $F(i, S) + h_i + c(i, j)$, provided that no deadlines are violated, that is, if $F(i, S) + h_i + c(i, j) \le \min_{k \notin S} d_k$. If the latter condition holds, we associate to state (j, S') a label with the value of $F(i, S) + h_i + c(i, j)$; we say that a label has been *propagated* from (i, S) to (j, S') and that (j, S') can be reached from (i, S). The value of $F(j, S')$ is given by $\max\{r_j, U(j, S)\}$, where $U(j, S)$ is the minimum of all labels associated with (j, S). We have $F(j, S') = \infty$ if there are no labels associated with (j, S).

The above discussion defines an algorithm based on label propagation. (It is just a particular implementation of forward dynamic programming.) The key property is that we do not have to do any work for any state (i, S) that does not have any associated labels. Thus, the complexity of the algorithm is proportional to the number of reachable states, times a bound on the number of labels propagated from any given state.

We first bound the number of reachable states. Suppose that (i, S) is reachable. Let $t = F(i, S)$. Let $Q = \{j \mid d_j < t\}$ and $R = \{j \mid r_j > t\}$. Then, $Q \subset S$ and $S \cap R = \emptyset$. Thus, $S = Q \cup S'$, where $S' \subset A(t)$. Note that if $A(t)$ is known, then Q is uniquely determined. (In particular, $Q = \{j \mid d_j < \min_{k \in A(t)} d_k\}$.) If follows that S is uniquely determined by specifying $A(t)$ and then specifying a subset S' of $A(t)$. We observe that, as t varies, the set $A(t)$ only changes $O(n)$ times. Thus, there are $O(n)$ choices for $A(t)$ and, having fixed $A(t)$, there are $O(2^D)$ choices for S'. We conclude that there are $O(D2^D)$ choices for S and the total number of reachable states is $O(nD2^D)$.

Suppose that state (i, S) is reachable and suppose that states $(i_1, S \cup \{i_1\})$, $\ldots, (i_m, S \cup \{i_m\})$ can all be reached from (i, S). Thus, we can reach state $(i_k, S \cup \{i_k\})$ immediately after state (i, S), without violating any deadlines, which implies that $d_{i_\ell} \ge r_{i_k}$, for $k, \ell = 1, \ldots, m$. Thus, $\max_k r_{i_k} \le \min_k d_{i_k}$ and there is a time at which all jobs $i_1, \ldots, i_m$ are active. It follows that $m \le D$ and at most D labels are propagated from each state. Using this, the desired complexity estimate follows. ■

In the proof of Theorem 8, we have implicitly assumed that given a state (i, S), we can determine the $O(D)$ states that can be reached immediately after $(i,$

S), in $O(D)$ time. This can be easily achieved provided that some preprocessing is performed on the problem data. For example, the $O(n)$ possible choices of $A(t)$ should be determined during a preprocessing phase. The complexity of this phase depends on the data structures employed and the representation of the input. If the jobs are available sorted according to their deadlines, preprocessing could take only $O(n)$ time. We omit the details but note that the preceding comments apply to our next result as well.

Theorem 9. TRPTW(D) can be solved in time $O(TD^2 2^D)$, where T is an upper bound on the time horizon of the problem.

Proof. Let the state (i, S) have the same meaning as in the proof of Theorem 8. For any state (i, S), and time t, let $W(i, S, t)$ be the minimum possible total waiting time accumulated, subject to the constraint that the server is at state (i, S) at time t. Let $W(i, S, t) = \infty$ if (i, S) is unreachable. We compute $W(i, S, t)$ by propagating labels with the values of $W(i, S, t)$ (forward dynamic programming). As argued in the previous proof, there are $O(D2^D)$ choices for (i, S). There are also T choices for t. Finally, from any (i, S), we can reach directly only $O(D)$ other states; thus, whenever $W(i, S, t) < \infty$, $O(D)$ labels are propagated, leading to the desired complexity estimate. ■

The algorithms developed in this section are quite similar to the algorithms in [4] and [5], as far as the use of forward dynamic programming and label propagation is concerned (though the problems considered in these references are different). Although no complexity estimates were provided in these references, the experimental results they have reported are qualitatively consistent with Theorems 8 and 9. In particular, [5] reports that "solution times increase linearly with problem size." Indeed, this is what is predicted from Theorems 8 and 9 if we assume that T is proportional to n and that D is held constant.

As in [4] and [5], the practical performance of the algorithms can be speeded up substantially by introducing certain additional tests that quickly identify unreachable states and save the effort of trying to propagate labels to them.

Note that the algorithm of Theorem 9 is pseudopolynomial. The following result shows that there is little hope for improvement.

Theorem 10. TRPTW(D) is NP-complete even in the special case where $D = 2$ and the processing times are zero.

Proof. We start from the 0-1 KNAPSACK problem that is known to be NP-complete. An instance of this problem consists of nonnegative integers $z_1, \ldots, z_n, \tau_1, \ldots, \tau_n$, and K. The problem consists of finding a set $S \subset \{1, \ldots, n\}$ that maximizes $\Sigma_{i \in S} z_i$ subject to the constraint $\Sigma_{i \in S} \tau_i \leq K$. We will now construct an equivalent instance of TRPTW(2), with zero processing times.

Consider a directed graph with nodes $u_1, \ldots, u_{n+1}, v_1, \ldots, v_n, w_1, \ldots, w_n$, connected as shown in Figure 3. The label next to each arc denotes its length. Here $\Delta_i = (2nK + 1)z_i$ and $Q = \max\{K, \max_i \tau_i\} + 1$. Note that the

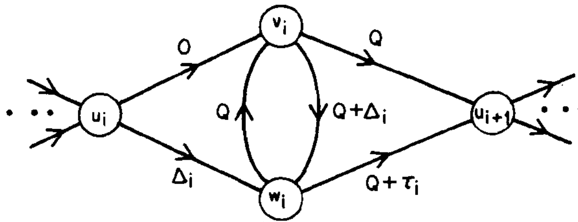

FIG. 3. A section of the graph constructed in the proof of Theorem 10.

triangle inequality is satisfied. We can turn this graph into a complete one by letting $c(i, j)$ be the length of a shortest path from node i to node j. [Let $c(i, j)$ be a very large number, if no such path exists.] There is a job W_i at node w_i and a job V_i at node v_i, for each i. The jobs W_i and V_i have a common release time $r_i = 2(i - 1)Q + \Sigma_{j=1}^{i-1} \Delta_j$ and a common deadline $d_i = r_i + Q + \Delta_i + K$. Since $Q > K$, it is easily checked that $d_i < r_{i+1}$. Finally, there is a job U_{n+1} at node u_{n+1} whose release time r_{n+1} and deadline d_{n+1} are both equal to $2nQ + K + \Sigma_{j=1}^{n} \Delta_j$. Again, it is easily checked that $r_{n+1} > d_n$. Therefore, we can only have two active jobs at a time, as required for an instance of TRPTW(2).

It is clear that in order for the server to get from node u_i to node u_{i+1}, it has two options.

(a) Go to w_i, then to v_i, then to u_{i+1}. This takes time $\Delta_i + 2Q$; we call this the *fast* path.
(b) Go to v_i, then to w_i, then to u_{i+1}. This takes time $\Delta_i + 2Q + \tau_i$; we call this the *slow* path.

For any schedule, let $S \subset \{1, \ldots, n\}$ be the set of all i such that the server takes the slow path from u_i to u_{i+1}. Note that the deadline for job U_{n+1} imposes the constraint

$$\sum_{i\in S} \tau_i \leq K. \tag{2}$$

Note that r_i $(i \neq n + 1)$ is the shortest path length from u_1 to u_i and therefore does not impose any constraint on the schedule. Furthermore, the constraint of Eq. (2) implies that the server is not allowed to get to node u_i any later than time $r_i + K$. It follows that the release time and deadline constraints are inconsequential and will be ignored in subsequent discussion.

Given a schedule, let

$$D_i = \sum_{j\in S, j<i} \tau_j. \tag{3}$$

Then, the server reaches node u_i at time $r_i + D_i$. If it takes the slow path to u_{i+1}, then the sum of waiting time suffered by V_i and W_i is equal to $2(r_i + D_i) +$

$Q + \Delta_i$. If it takes the fast path, this sum is equal to $2(r_i + D_i) + Q + 2\Delta_i$. By comparing the two expressions, we see that our instance of TRPTW(2) is equivalent to minimizing $2 \sum_{i=1}^{n} D_i - \sum_{i \in S} \Delta_i$ with respect to S, subject to the constraints (2) and (3). Given that each Δ_i is a multiple of $2nK + 1$, whereas $2 \sum_{i=1}^{n} D_i$ has to be bounded by $2nK$ [cf. Eqs. (2) and (3)], an equivalent problem is to maximize $\sum_{i \in S} \Delta_i$ subject to Eq. (1). Given that each Δ_i is proportional to z_i, this is equivalent to the original instance of the 0-1 KNAPSACK problem. ■

IX. CONCLUSION

We have studied the complexity of several variants and special cases of the traveling salesman and repairman problems in the presence of time windows and processing times. Although our study has settled the complexity of many of these variants, there is still a number of open problems. In particular, we have not determined the complexity of those problems in Tables I to III that have a question mark.

In another interesting variant, we could have assumed that the width of the time windows is bounded by an integer W. If we assume that each job has at least unit processing time, it is easily seen that either the problem is infeasible or the number of active jobs is bounded by $2W$. Therefore, by the results of Section 8, this special case of TSPTW (respectively, TRPTW) can be solved in polynomial (respectively, pseudopolynomial) time.

Discussions with Harry Psaraftis and Francois Soumis are gratefully acknowledged.

REFERENCES

[1] F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakonstantinou, The complexity of the traveling repairman problem. *Theor. Info. Appl.* **20**(1) (1986) 79–87.

[2] J. Bruno and P. Downey, Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM J. Comput.* **7**(4), 1978 (393–404).

[3] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, and F. Soumis, Vehicle routing with time windows: Optimization and approximation. *Vehicle Routing: Methods and Studies* (B. L. Golden and A. A. Assad, Eds.). Elsevier, Amsterdam (1988) 65–84.

[4] J. Desrosiers, P. Pelletier, and F. Soumis, Plus court chemin avec contraintes d'horaires. *R.A.I.R.O. Recherche Operationelle* **17**(4) (1983) 357–377.

[5] J. Desrosiers, Y. Dumas, and F. Soumis, A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *Am. J. Math. Management Sci.* **6**(3–4) (1986) 301–325.

[6] M. R. Garey and D. S. Johnson, Two-processor scheduling with start times and deadlines. *SIAM J. Comput.* **6** (1977) 416–426.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco (1979).

[8] M. Held and R. M. Karp, A dynamic programming approach to sequencing problems. *J. SIAM* **10**(1) (1962) 196–210.

[9] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, and D. B. Shmoys, Ed., *The Traveling Salesman Problem*. Wiley, New York (1985).

[10] J. K. Lenstra, A. H. G. Rinnoy Kan, and B. Brucker, Complexity of machine scheduling problems. *Ann. Discrete Math.* **1** (1977) 343–362.
[11] C. L. Monma and C. N. Potts, On the complexity of scheduling with batch setup times. *Operations Res.* **37** (1989) 798–804.
[12] H. N. Psaraftis, M. M. Solomon, T. L. Magnanti, and T.-U. Kim, Routing and scheduling on a shoreline with release times. *Management Sci.* **36**(2) (1990) 212–223.
[13] M. W. P. Savelsbergh, Local search in routing problems with time windows. *Ann. Operations Res.* **4** (1985/6) 285–305.
[14] W. E. Smith, Various optimizers for single-stage production. *Naval Res. Logistics Q.* **3** (1956) 59–66.
[15] M. Solomon and J. Desrosiers, Time window constrained routing and scheduling problems. *Transportation Sci.* **22**(1) (1988) 1–13.

Received September 1990
Accepted November 1991