CHAPTER 4

TIME-DEPENDENT LABEL-CONSTRAINED SHORTEST PATH PROBLEM WITH APPLICATION TO TRANSIMS

In contrast with the former case of constant travel times, suppose now that the link delays are time-dependent functions. Specifically, for each $(p, q) \in A$, suppose that we have a time-dependent link delay function $d_{pq}(t)$ that specifies the travel time on link (*p*, *q*), given a starting time *t* at the corresponding tail node *t*. This function d_{pq} might be a general real valued function defined on a continuum of time over some horizon interval H $(d_{pq} : H \subseteq R \rightarrow R)$, or it might be some discretized approximation from some experimental or simulation output analysis, being defined as

$$d_{pq} : H \equiv \{0, D, 2D, ..., KD\} \rightarrow \{0, D, 2D, ..., K'D\}$$

for some discretized time duration D and integer K', and some suitably large integer K such that the characterization of the delay function beyond the time KD is not of practical interest.

In either case, let us define

T = some upper bound on the length (total delay) of any acceptable path in the solution to the underlying problem.

Note that we use this additional parameter T to control the degree of exploration of the network and to fathom or weed out the examination of paths that do not have delays below this threshold T. Also, it is well known that there exists a time-space static equivalent network representation for this problem. In this representation, each node is replicated as $\{i, t\}$ for all possible values of $t \in H$ when we could be at node i. Furthermore, for each $(i, j) \in A$, given a possible starting time $t < T$ at node i, where $t \in H$, this representation would construct an arc $\{(i, t), (j, t + d_{ij}(t))\}$ having a fixed delay of $d_{ij}(t)$, such that $t + d_{ij}(t) \leq T$, where note that $t + d_{ij}(t)$ then also belongs to H by our assumption.

Similar to the time-independent case, we also have labels associated with each arc taken from some alphabet set $\sum$, and we are given a language L defined on regular expressions that is comprised of a set of acceptable words (sequence of arc labels on acceptable paths). Suppose further that as before, we are given (or have constructed) the corresponding graph G_L. The *Time-Dependent Label-Constrained Shortest Path Problem* (TDLSP) then is to find a time-dependent shortest path P^* in G from O to D among all paths P from O to D (i.e., $P \in \wp$) for which $l(P) \in L$, where $l(P)$ is the sequence of alphabet labels on the path P.

The procedure we develop dynamically generates a reduced-size time-space network *implicitly*, using a minimal number of time-based node replications, while *simultaneously* finding the TDLSP from O to D. This is done within the framework of a dynamic programming routine that is similar in concept to the procedure of Figure 4. Figure 5 describes the proposed procedure. Here, time-expanded replicates (p, t, l) of each node $p \in N$ are automatically created only for specific, necessary values of times t, and labels l based on possible visitation times and label sequences.

In this process, we develop the node sets N_s for s = 0, 1,...,S as in Figure 4, but maintain an additional time component for each node, along with the predecessor or $DOWN_s(\cdot)$ list for each stage s in order to facilitate a back-tracing of the generated paths. Moreover, since we are not interested in pursuing paths having total delays that exceed T, and all delays are nonnegative, we trim off nodes for which the delay exceeds T.

Furthermore, as before, the nodes investigated for the final stage S correspond only to the terminal node D. Note that termination might occur prior to stage S either because no time-dynamic label constrained path is realizable that has a total delay less than or equal to the specified limit T, or within such a limit, the nodes at some earlier stage all correspond to the terminal node D.

Remark 3. Observe that when we perform the operation $N_{s+1} \leftarrow N_{s+1} \cup \{(i,t,l)\}$, if the node (i,t,l) already exists in N_{s+1}, we revise its $DOWN_{s+1}(\cdot)$ index as stated in the procedure; however, we could have left this index the same. Hence, only some alternative equally attractive choice of a partial path and label sequence is being maintained. ⊠

Remark 4. Note that the general procedure does not assume the *first-in-first-out* (FIFO) or *consistency* assumption whereby for each link $(p, q) \in A$, if we enter the link at an earlier time then we would also exit the link at a relatively earlier time (see Kaufman and Smith (1993)). However, when such an assumption holds true, we do not need to maintain duplicate nodes of the type (i, t_1, l) and (i, t_2, l) for $t_1 < t_2$ at any stage; the latter node is dominated by the former and can be dropped from consideration. Figure 5 states this modified rule in the main processing block. In particular, note that for the time-independent case, we have $d_{pq}(t) \equiv d_{pq} \ \forall \ t \in H, \quad \forall \ (p, q) \in A$. Hence, the consistency assumption holds true, and the procedure of Figure 5 can be applied under this revision. ☒

Observe that in an actual implementation of the procedure of Figure 5, we would number the distinct nodes (i, t, l) generated in the process consecutively as k = 1, 2, 3, …, maintaining a list that equates each such index k to the corresponding triplet-node indicator (i, t, l), along with its corresponding stage. Furthermore, we can maintain a one-dimensional array DOWN (·), where DOWN $(k_1) = k_2$ for a node $(k_1) = (i, t, l)$ at stage $s + 1$ and a node $(k_2) = (i', t', l')$ at stage s corresponds to $\text{DOWN}_{s+1}(i, t, l) = (i', t', l')$ in Figure 5.

It is also worthwhile to note that the routine of Figure 5 is an extension to the partition shortest path procedure PSP (see Glover et al. (1985 a, b)) that considers, time-dynamic travel times as well as label-sequence requirements. Sherali (1991) describes the relationship between PSP and a dynamic programming routine, where the states and decisions at any stage in the latter correspond to the set of possible nodes that can be arrived at (called NEXT in PSP), and the set of nodes from which this arrival occurs (called NOW in PSP), respectively. A similar relationship of the procedure of Figure5 to a dynamic programming routine, and consequently, to a PSP approach is evident in the present context. Some insights into this relationship are detailed below.

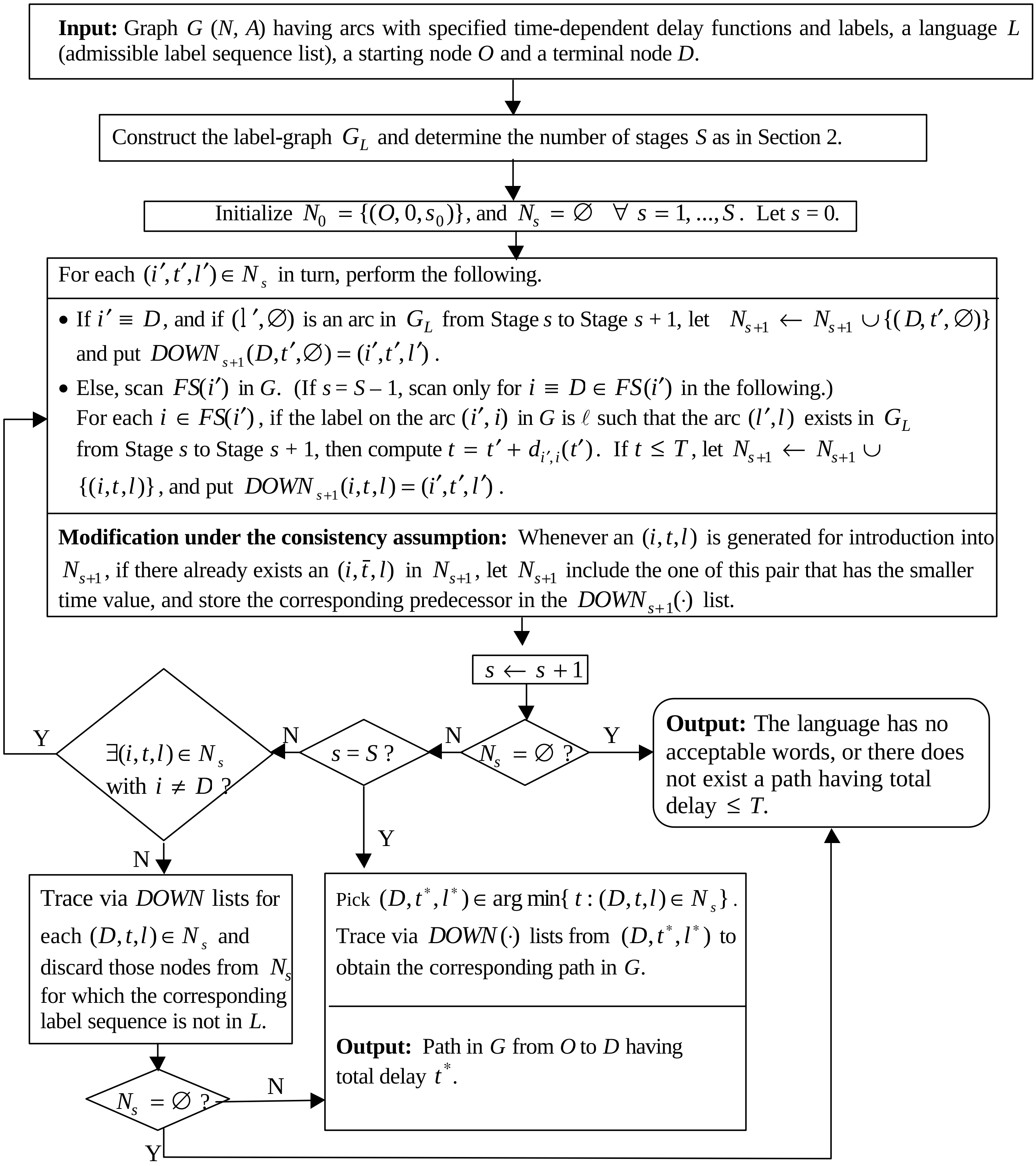
Relationship between the Partitioned Shortest Path (PSP), the Dynamic Programming (DP), and the Proposed Algorithms for Solving TDLSP

Sherali [1991] has established a theoretical equivalence between the PSP algorithm, and a dynamic programming (DP) approach. Consider the following definition of *stages, states,* and *decisions* for a forward-recursive dynamic programming formulation of the problem of determining shortest simple paths from a root node r to all the other nodes of a network (N, A), having node set $N = \{1, \ldots, n\}$ and arc set A. Let c_{ij} be arbitrary costs associated with the links $(i, j) \in A$. Let **stage** k represents a point in the DP algorithmic process when we are k steps away from the root node r. Hence, the DP algorithm can have at most n-1 stages. At any stage k, the **state** variable s_k is represented by a set of corresponding nodes reachable from r in k steps, and the immediate predecessors of this set of nodes constitute the set of possible **decisions** d_k.

At any stage k, suppose that we denote by NOW a list of current (decision) nodes, and we let NEXT represent a list of successor nodes of the nodes in NOW. Note that the nodes in NOW are reachable within k steps. The DP algorithm updates the k-step SP information for the nodes in the list NOW to the $(k+1)$-step SP information for the corresponding nodes in the list NEXT at each stage. At the end of each stage k, we put the current list NEXT of stage k equal to the list NOW of the following stage (i.e. $\text{NOW}_{k+1} \leftarrow \text{NEXT}_k$).

The difference between the PSP and DP algorithms is that at any stage k, the DP algorithm simultaneously considers all the nodes in NOW, and finds the corresponding list NEXT for this set of nodes, while, the PSP algorithm selects a node in NOW one at a time, and finds its successor nodes to form the list NEXT. The PSP algorithm adds any successor node when its label is updated to the list NEXT one at a time, **unless if the node is still present in NOW**. Hence the main difference between the PSP and the DP algorithms occurs when some node in the list NOW revises **a successor node, which is still in the current list NOW**. The DP algorithm will consider the revised labels/costs/total times of the node at the **subsequent** stage, while the PSP algorithm uses the revised labels continuously **within** the stage itself, and does not add such a node to the list NEXT. Hence, at the subsequent stage, the PSP algorithm will not contain this node again in the list NOW (unless its label gets revised after it has been processed in NOW), whereas in the DP algorithm, this node will still reappear in the subsequent list NOW. However, Sherali [1991] shows that there exists a sequence of selecting nodes in NOW for scanning at any iteration in the PSP algorithm such that for this sequence of selecting nodes in NOW, the PSP algorithm is precisely the foregoing DP algorithm.

Algorithm TDLSP proceeds similar to the DP interpretation of the PSP algorithm because of the need to keep a precise track of the number of steps in order to enforce the constraint on the label sequence. However, the algorithm TDLSP does not define the lists NOW and NEXT **explicitly**. It defines the node set N_s as a collection of reachable nodes (i, t, l) at stage s, where i denotes the nodes' number/name, t denotes the arrival time to the node i, and l denotes the label used to arrive at the node i. The set N_s is revised in the algorithmic process from stage s to stage s+1. Based on the above discussion, it is evident that the set N_s represents the list NOW, and the set N_{s+1} represents the list NEXT in accordance with their usage within the PSP and the DP algorithms.



Input: Graph G (N, A) having arcs with specified time-dependent delay functions and labels, a language L (admissible label sequence list), a starting node O and a terminal node D.

Construct the label-graph G_L and determine the number of stages S as in Section 2.

Initialize $N_0 = \{(O, 0, s_0)\}$, and $N_s = \varnothing \quad \forall\ s = 1, ..., S$. Let $s = 0$.

For each $(i', t', l') \in N_s$ in turn, perform the following.

- If $i' \equiv D$, and if $(l', \varnothing)$ is an arc in G_L from Stage s to Stage $s + 1$, let $N_{s+1} \leftarrow N_{s+1} \cup \{(D, t', \varnothing)\}$ and put $DOWN_{s+1}(D, t', \varnothing) = (i', t', l')$.
- Else, scan $FS(i')$ in G. (If $s = S - 1$, scan only for $i \equiv D \in FS(i')$ in the following.) For each $i \in FS(i')$, if the label on the arc (i', i) in G is ℓ such that the arc (l', l) exists in G_L from Stage s to Stage $s + 1$, then compute $t = t' + d_{i',i}(t')$. If $t \le T$, let $N_{s+1} \leftarrow N_{s+1} \cup \{(i, t, l)\}$, and put $DOWN_{s+1}(i, t, l) = (i', t', l')$.

Modification under the consistency assumption: Whenever an (i, t, l) is generated for introduction into N_{s+1}, if there already exists an $(i, \bar{t}, l)$ in N_{s+1}, let N_{s+1} include the one of this pair that has the smaller time value, and store the corresponding predecessor in the $DOWN_{s+1}(\cdot)$ list.

$s \leftarrow s + 1$

$N_s = \varnothing$?

Output: The language has no acceptable words, or there does not exist a path having total delay $\le T$.

$s = S$?

$\exists (i, t, l) \in N_s$ with $i \neq D$?

Trace via $DOWN$ lists for each $(D, t, l) \in N_s$ and discard those nodes from N_s for which the corresponding label sequence is not in L.

$N_s = \varnothing$?

Pick $(D, t^*, l^*) \in \arg\min\{\, t : (D, t, l) \in N_s \}$. Trace via $DOWN(\cdot)$ lists from (D, t^*, l^*) to obtain the corresponding path in G.

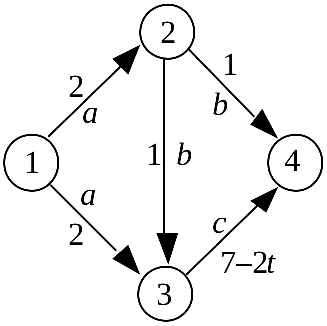
Output: Path in G from O to D having total delay t^*.

Figure 5. Flow-Chart for the Algorithm to Solve the TDLSP

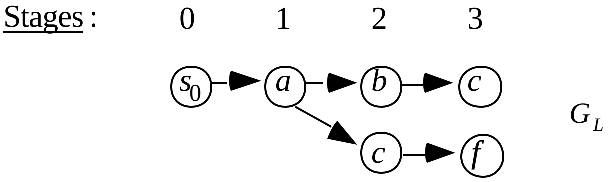
]

Example 2 (Inconsistent or Non-FIFO Delays)

Consider the following graph G with delay functions and arc labels as shown.



Note that all the links have time-independent delays, except for link (3, 4) for which $d_{34}(t) = 7 - 2t$. Hence, if we arrive at node 3 at time t = 2, it would put us at node 4 at time 2 + 3 = 5, while arriving at node 3 at a later time t = 3, would put us at node 4 at time 3 + 1 = 4, i.e., earlier than in the former case. Suppose also that the specified language is $L = \{abc, ac\}$, so that the non-label constrained shortest path $1 \rightarrow 2 \rightarrow 4$ is infeasible. Let us also assume that $T \geq 5$, and note that S = 3. The graph G_L is given as follows.



The algorithm of Figure 5 would proceed as follows, where the arrows depict the corresponding $DOWN(\cdot)$ relationships established at each stage.

Initialization: $N_0 = \{(1,0,s_0)\}$, where $O \equiv 1$, and $N_s = \emptyset$ for $s = 1,2,3$.

s = 0:

$$\begin{tikzcd}[row sep=small]
 & (2,2,a) & \\
N_0 = \{(1,0,s_0)\} \arrow[ur] \arrow[dr] & & = N_1 \\
 & (3,2,a) &
\end{tikzcd}$$

s = 1:

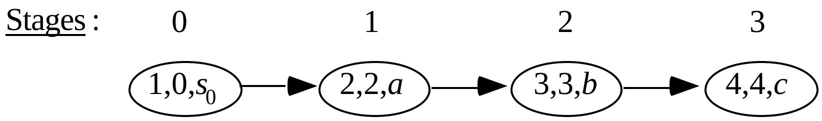
$$\begin{tikzcd}[row sep=small]
 & & (4,3,b) & \\
N_1 = & (2,2,a) \arrow[ur] \arrow[r] & (3,3,b) & = N_2 \\
 & (3,2,a) \arrow[r] & (4,5,c) &
\end{tikzcd}$$

s = 2:

$$\begin{tikzcd}[row sep=small]
 & (4,3,b) & & \\
N_2 = & (3,3,b) \arrow[r] & (4,4,c) & = N_3 \\
 & (4,5,c) \arrow[r] & (4,5,\emptyset) &
\end{tikzcd}$$

s = 3:

With $D \equiv 4$, we identify (D, t^*, l^*) as (4,4,*c*). Tracing via $DOWN(\cdot)$ produces the following path (in reverse order).

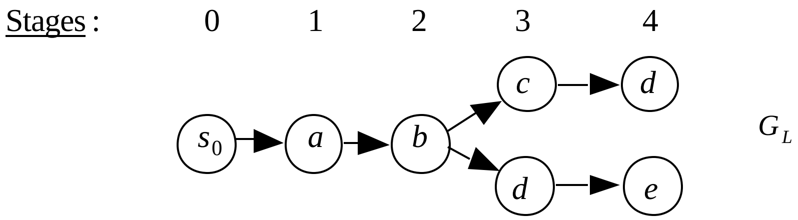


Hence, the path 1 → 2 → 3 → 4 having a label sequence *abc* and total delay of $t^* = 4$ solves the given TDLSP problem.

Remark 5. Note that if we had specified $T = 4$, then the node $(4, 5, c) \in N_2$ would have been suppressed at Stage $s = 1$, and so, N_3 would only have inherited (4,4,*c*) at Stage $s = 2$. Alternatively, with $T = 3$, N_2 would again have dropped (4,5,*c*) at $s = 1$, while we would have obtained $N_3 = \varnothing$ at $s = 2$. The procedure would have terminated with an infeasibility indication. Decreasing T further would have yielded an infeasibility indication at an earlier stage. ⊠

Example 3. (Consistent or FIFO Delays):

For the sake of illustrating the application of Figure 5 to solve a *time-independent* label constrained shortest path problem, consider Example 1 under the language specification of Case (ii), and assume that $T = 9$. The value of S is 4, and the graph G_L is shown below.



The procedure of Figure 5 would proceed as follows, noting the modification stated under the consistency assumption.

Initialization: $N_0 = \{(1, 0, s_0)\}$, where $O \equiv 1$, and $N_s = \varnothing$ for $s = 1,...,4$.

$s = 0$:

$$N_0 = \begin{tikzcd} \{(1, 0, s_0)\} \arrow[r] \arrow[dr] & (2, 2, a) \\ & (3, 5, a) \end{tikzcd} = N_1$$

$s = 1$:

$$N_1 = \begin{tikzcd} (2, 2, a) \arrow[r] \arrow[dr] & (3, 4, b) \\ (3, 5, a) & (4, 5, b) \end{tikzcd} = N_2$$

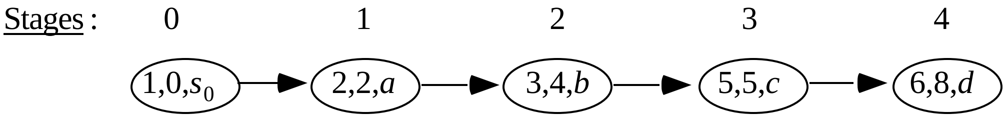
$\boxed{s = 2:}$

$$N_2 = \left\{\begin{tikzcd}[row sep=small] (3,4,b) \arrow[r] \arrow[dr] & (5,5,c) \\ & (4,8,d) \\ (4,5,b) \arrow[r] & (5,7,c) \leftarrow \text{eliminate} \end{tikzcd}\right\} = N_3$$

$\boxed{s = 3:}$

$$N_3 = \left\{\begin{tikzcd}[row sep=small] (5,5,c) \arrow[r] & (6,8,d) \\ (4,8,d) \arrow[r] & (6,9,e) \end{tikzcd}\right\} = N_4$$

$\boxed{s = 4:}$ Terminate with (D, t^*, l^*) = (6,8,*d*). Tracing backwards using the $DOWN(\cdot)$ list yields



This yields the optimal TDLSP $1 \to 2 \to 3 \to 5 \to 6$ in *G*, with the label sequence *abcd*, and having a total delay equal to $t^* = 8$.

Remark 6 (Choice of *T* and Curtailing Computations). If we know some feasible path to the TDLSP problem, then using its delay as the value of *T*, we can reduce the extent of computations performed. This motivates the use of a quick heuristic to derive an initial feasible solution. Furthermore, in large-scale applications, we could partition the graph into sections depending on the choice of *O* and *D*, and require the nodes within each section to be visited by a certain time. Alternatively, we can specify a threshold value T_s for each stage *s*, and maintain only those $(i, t, l) \in N_s$ for which $t \leq T_s \ \forall \ s = 1, ..., S$. By suitably choosing a progression of T_s-values, we can curtail the computational effort, although this might mean a loss in finding an exact optimum.

This feature will be further developed in this thesis, in concert with Euclidean Heuristic (see Sedgewick et al. [1986]) ⊠

We close this section by establishing the complexity of the procedure in Figure 5.

Proposition 1. Given a graph $G(N,A)$, and given the graph G_L corresponding to the language L, let

r = maximum admissible word length in L (longest simple path in G_L)

m = maximum number of nodes at any stage in G_L.

Then under the consistency assumption, the procedure of Figure 5 is of polynomial-time complexity $O(rm^3 \mid A \mid)$.

Proof. The number of stages enumerated is $O(r)$. For each stage s, any node i of G appears in $(i,t,l) \in N_s$ at most m times. Scanning the forward star of i and checking the at most m^2 arcs in G_L from stage s to stage $s + 1$ for each such repetition of i takes $O(\mid FS(i) \mid m^3)$ time. Summing this over all possible nodes of G appearing at stage s gives a total complexity of $O(m^3 \mid A \mid)$ per stage. Hence, the overall process is of complexity $O(rm^3 \mid A \mid)$, and this completes the proof. ⊠

Remark 7. In the case of non-FIFO link delays, if t is the maximum number of distinct values of times for which it is possible to visit any node within the interval $[0,T]$ (for integer valued delay functions, we can take $t \equiv T$), then the complexity of the algorithm of Figure 5 is pseudopolynomial of order $O(trm^3 \mid A \mid)$. ⊠

Time-Dynamic Chained Activity Route Planner

The key utility of the foregoing approach arises in the following context. Consider a certain traveler starting at a home location at a certain time (t = 0), and wishing to go to the office via label strings constituting a language of the type $L = \{(w...w, d...d, r...r, w...w), (w...w, d...d, b...b, w...w)\}$ where w, d, r, and b respectively represent walking, driving, rail (taking trains), and taking buses. The words can be of various lengths, so long as they are admissible with respect to the overall multi-modal transit network. This latter network can be conceptualized as a layered network as shown in Figure 6, where the starting node O is represented by the home location and the terminal node D is represented by the office location on the walking network, and where there exist various *process arcs* (shown dotted) between appropriate possible and desirable location connections from one layer to another.

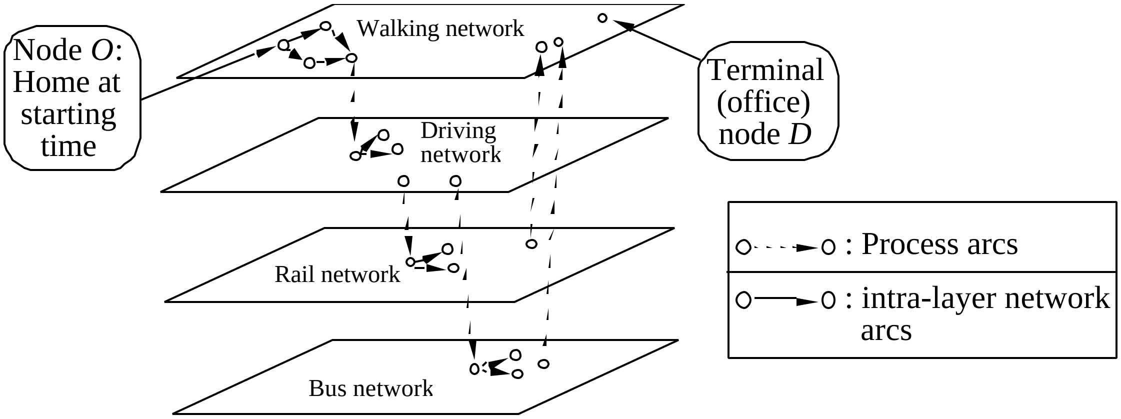
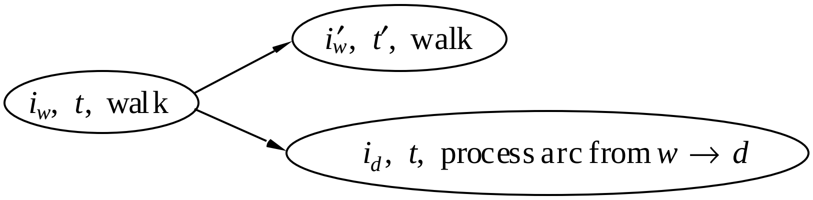
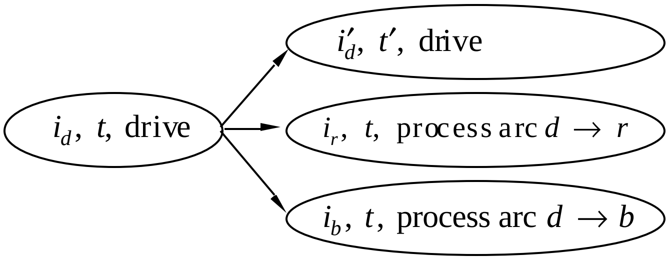

Node *O*: Home at starting time
Walking network
Terminal (office) node *D*
Driving network
Rail network
Bus network
: Process arcs
: intra-layer network arcs

Figure 6: Layered Multi-Modal Network.

We can now apply the algorithm of Figure 5 to this network under time-independent or time-dependent conditions, and in the latter case, under a non-FIFO or FIFO (consistency) assumption. For example, given a node (i_w, t, walk) at some stage s, representing that we have reached location i at time t while walking (in the walking network), we might have possible connecting arcs of the following type leading to states in stage s + 1, where i_w and i_d respectively represent location i in the walking and driving networks.



Similarly, in the drive network, we might have the following types of connections from one stage to the next.


i_d, t, drive
i'_d, t', drive
i_r, t, process arc $d \rightarrow r$
i_b, t, process arc $d \rightarrow b$

The key element here is an efficient implementation of the procedure of Figure 5, using suitable node-set reduction schemes as described in Remark 6.

Remark 8. Note that the algorithm discussed thus far assumes that the time delay functions $d_{ij}(t)$ are known for each link (i, j) in each of the layered networks. Such information might be gleaned experimentally or via a dynamic traffic assignment simulation process executed using some OD trip matrices (e.g., see Hobeika, Sherali, and Sivanandan [1994] or Hobeika and Sherali [1997]). These delay values for each link can be stored in look-up tables (e.g. as estimated travel times on an hourly or half-hourly basis) or be input as analytical statistically determined functions. We can also assume that the FIFO or consistency condition holds, which is true for most practical cases. ☒