CHAPTER 3

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

Let us begin by considering the time-independent version of this problem. In what follows, we assume standard terminology for graphs (see Bazaraa, Jarvis, and Sherali (1990), for example). In particular, if $G(N, A)$ is a graph having a node set N and an arc set A, then the forward star $FS(i)$ for any $i \in N$ is defined as $FS(i) = \{j : (i, j) \in A\}$, and the reverse star $RS(i)$ for any $i \in N$ is given by $RS(i) : \{j : (j, i) \in A\}$. Suppose that we are given a digraph $G(N, A)$ where N and A are the sets of nodes and arcs of G, respectively. Let $O \in N$ be the origin (or starting) node, and let $D \in N$ be the destination (or final) node. Without loss of generality, assume that the network has been preprocessed such that $RS(O) = \varnothing$ and $FS(D) = \varnothing$, where $FS(\cdot)$ and $RS(\cdot)$ respectively denote the forward and reverse stars of any node $(\cdot)$. Furthermore, assume that by successively scanning the sets $FS(\cdot)$ starting at O we find all the nodes that are reachable from S, and by successively scanning the sets $RS(\cdot)$ starting at D we find all the nodes that can reach D. Accordingly, we can then assume that N is comprised only of nodes in the intersection of these two sets, with A being the associated connecting arcs. Note that this reduction is for algorithmic convenience/efficiency, but not necessary for its operation/application.

In addition to the (constant) delays or travel times d_{pq} specified on the arcs $(p, q) \in A$, suppose further that each arc is also ascribed a label taken from some alphabet Σ, and that we are given a language $L(R)$ (or simply L) defined on a regular expression R. That is, the set L is

comprised of *words*, or sequences of alphabets, that constitute acceptable sequences of labels on any selected path $P \in \wp$ = {Paths in *G* from node *O* to node *D*}. Hence, if $l(P)$ denotes the word formed by the sequence of labels on the arcs in a path $P \in \wp$, the *Time-Independent Label-Constrained Shortest Path Problem* (*TILSP*) is to find a shortest path *P** from *O* to *D* from among all paths in the set $\wp \cap \{P : l(P) \in L\}$.

The following are definitions of certain key computer science terminology used in the TRANSIMS documentation and in Barrett et al. (1998). For the purpose of our discussion and development, the items 1, 2, and 3 below suffice.

1. **Alphabet å:** collection of symbols (such as letters).
2. **å*:** collection of all possible finite strings of alphabets.
3. **Language L Í å*:** collection of "words" or strings from Σ^* that are acceptable according to some criteria or rules.
4. **Deterministic Finite Automaton (DFA):** (Q, Σ, d, q_o, F), where

   Q = set of finite states for the system

   Σ = finite input alphabet set

   $q_o \in Q$ = initial state

   $F \subseteq Q$ = set of final possible states

   $d : Q \times \Sigma \rightarrow Q$: transition function that takes a (state, alphabet) combination, and accordingly, transforms to some other (perhaps the same) state.

   Note: Sometimes $d : Q \times \Sigma^* \rightarrow Q$ is used for more general purposes.

5. **String $x \in$ å* is Accepted by a Deterministic Finite Automaton M:**

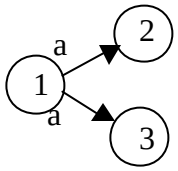
This happens if $d(q_o, x) = p$ for some $p \in F$, i.e., starting in the state q_o, under the operations implied by the string *x*, one will transition ultimately to a desired final state.

6. **Language Accepted by a Deterministic Finite Automaton M: L(M)**

$L(M) = \{x \in \sum^*: d(q_o, x) \in F\}$, i.e., this is the set of words x for which a transition from the initial state to some final state is possible.

7. **Regular Language L:** A language for which there is some deterministic finite automaton *M* for which $L \equiv L(M)$.

8. **Regular Expressions:** Class of regular languages, i.e., languages accepted by the collection of finite automata.

9. **Nondeterministic Finite Automaton (NFA):** One in which the transition function is a *point-to-set* map, i.e., given a state *q* and a label or alphabet *a*, $d(q, a)$ might be a set of possible states to which a transition could occur. For example, in 

we would have

$d(1, a) = \{2,3\}$. Hence, in an NFA, we have

$d : Q \times \sum \to 2^Q$, the power set of Q (set of all subsets of *Q*).

As before, we can generalize *d* to

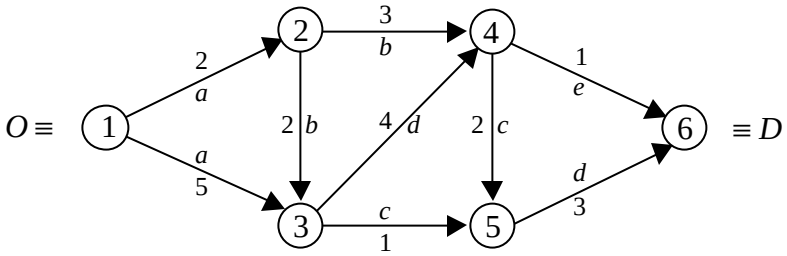
$d : 2^Q \times \sum^* \to 2^Q$, where $d(P, x) = \bigcup_{q \in P} d(q, x)$.

Note: Any NFA can be represented by an *equivalent* DFA by allowing the states of the corresponding DFA to be *sets* of the states defined for the NFA.

We will first focus on the basic concepts, and subsequently, we will prescribe an efficient algorithm in which the detailed steps and constructs described below are only *implicitly* conceptualized within the implementation. Toward this end, consider the following example.

Example 1.

Given a graph G having arc delays $d_{pq}\ \forall\ (p, q) \in A$ and labels as shown below, suppose that we are required to find a shortest path (not necessarily simple) from node $O = 1$ to node $D = 6$ subject to the constraint that the corresponding label sequence (word) should belong to the language L, where L is given by one of the following cases (each *case* here describes a different problem instance): (i) $L \equiv \{abcd\}$, or (ii) $L \equiv \{abcd, abde\}$, or (iii) $L \equiv \{abcd, abe\}$.



Step 1. Let node $O \equiv 1$ be the *state* at *Stage* 0, and define *states* at *Stage s* to be nodes reachable from node 1 in *s* steps. Hence, we have,

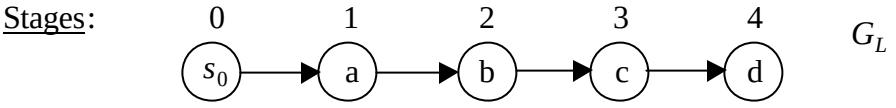
	Stages						
	0	1	2	3	4	5	6
States (nodes)	1	2	3	4	5	6	0
		3	4	5	6		
			5	6			

(**Note:** This need not be pre-generated; as mentioned above, this is only for conceptual purposes at this point, and the relevant information will be utilized implicitly during the solution process developed in Section 3.)

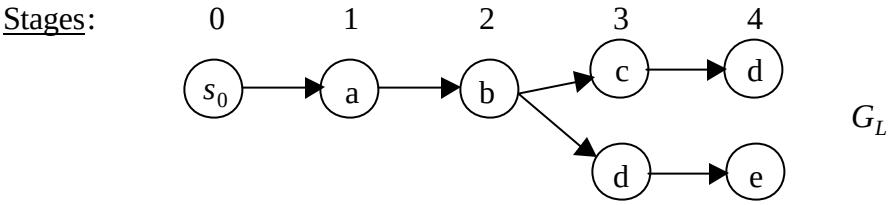
Step 2. Examining the admissible labels in L, construct a corresponding *transition graph* G_L as follows. Begin with a single node corresponding to a *dummy label* s_0 at Stage 0. Then, recursively for stages s = 1,2,..., given the graph up to Stage s–1, extend it to Stage s by performing the following constructions. Let the nodes of G_L at Stage s correspond to the possible distinct labels on the arcs via which we are permitted to reach a node/state in G at this Stage s. Next, if the language L permits an arrival via a label l' arc in G at Stage s–1 followed by a traversal of an arc having a label l to a node in G at Stage s, then introduce the arc (l',l) in G_L between the corresponding nodes l' and ℓ at stages s–1 and s, respectively, in G_L. Note that we assume that the language L contains only acceptable words, i.e., strings that define possible sequences by which we can reach node D from node O in that many steps. Furthermore, we assume that G_L exists such that all chains in G_L are admissible with respect to L, with any node label appearing at most once at any stage s (see Remark 1).

For the three cases of L specified above, we have the following corresponding transition graphs G_L

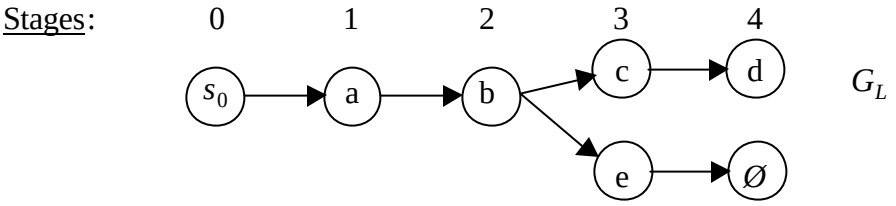
Case (i)



Case (ii)



Case (iii)



Note that as in Case (iii), if the permissible strings or words in L are of unequal lengths, we use a sequence of "empty" label symbols to extend the shorter words, so that all words in L become of the same length. Let the *number of stages S* to be considered equal the length of the maximum admissible string in L ($S = 4$ here).

Step 3. Using the graphs G and G_L (conceptually), construct a graph G^* having node (O, s_0) at Stage 0, and having the following nodes for each stage s, $s = 1,...,S$:

$\{(i,l)$: it is possible to come to node *i* at Stage *s* via an arc with label $l\}$

where we also have

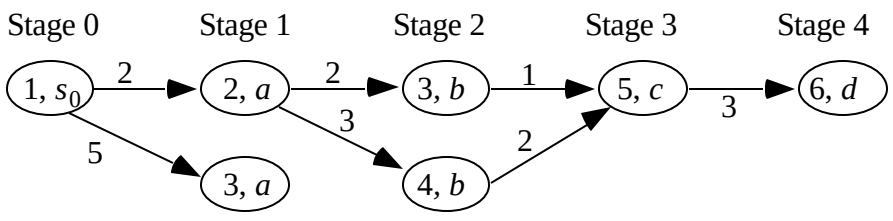
(a) for Stage *S*, only nodes (i,l) with *i* equal to the terminal node *D* are admissible, and

(b) the node $(i,\varnothing)$ is permitted only for *i* equal to the terminal node *D*. (In our example, node *D* = 6 is the terminal node.)

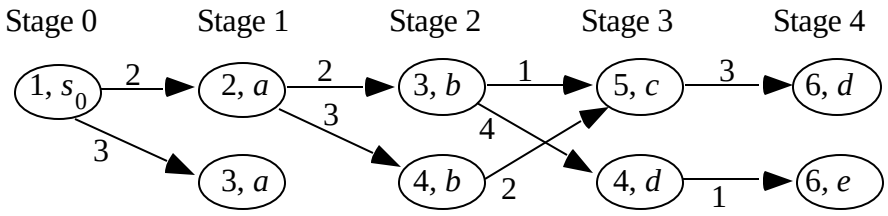
Next, progressing along the stages in order *s* = 1,...,*S*, for each stage $s \in \{1,...,S\}$, consider each node (i,l) belonging to this stage, and construct an arc from (i',l') at Stage *s*–1 to this node (i,l) having a cost $d_{i'i}$ if (a) we can come to node *i* from node i' in *G* with arc (i',i) in *G* having a label *l*, *and* (b) the arc (l',l) exists in G_L from Stage *s*–1 to Stage *s*. (Note that if $l \equiv \varnothing$, then we must have $i' \equiv i \equiv D$, and we take $d_{i'i} \equiv 0$.)

For the above three cases, the graph *G** can be constructed as follows.

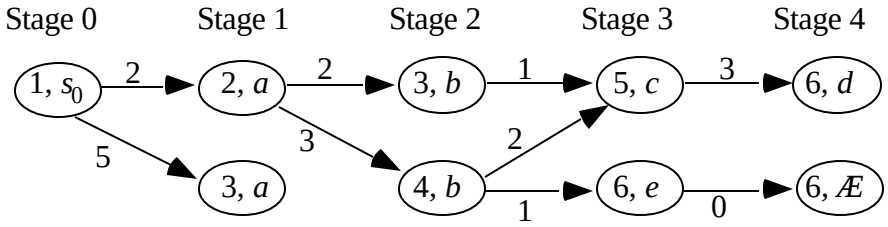
Case (i) *G*:*



Case (ii) *G:**

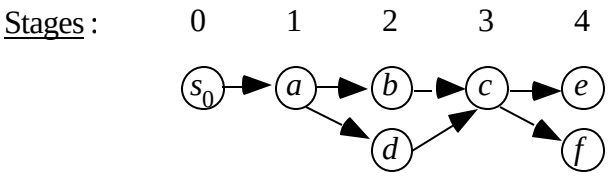


Case (iii) *G:**

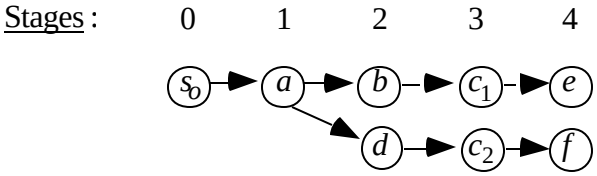


Step 4. Find the shortest path from node (0, s_0) to all the terminal nodes at Stage S using any standard SP algorithm. Pick the shortest of these paths and trace the corresponding path and the labels using a backward pass (using the DOWN or *predecessor labels* in the usual fashion).

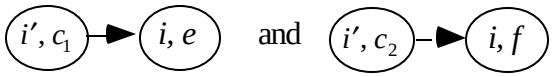
Remark 1. Note the importance of asserting the existence of the graph G_L without a duplication of nodes at each stage in G_L. For example, if L = {*abce*, *adcf*) for some graph G, then if we use the particular graph shown below as G_L, it would *also* imply the admissibility of the crossover strings *abcf* and *adce* in *L*.



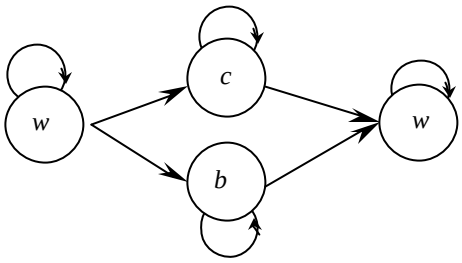
In order to avoid this inclusion of inadmissible strings, we would need to define the graph G_L as follows:



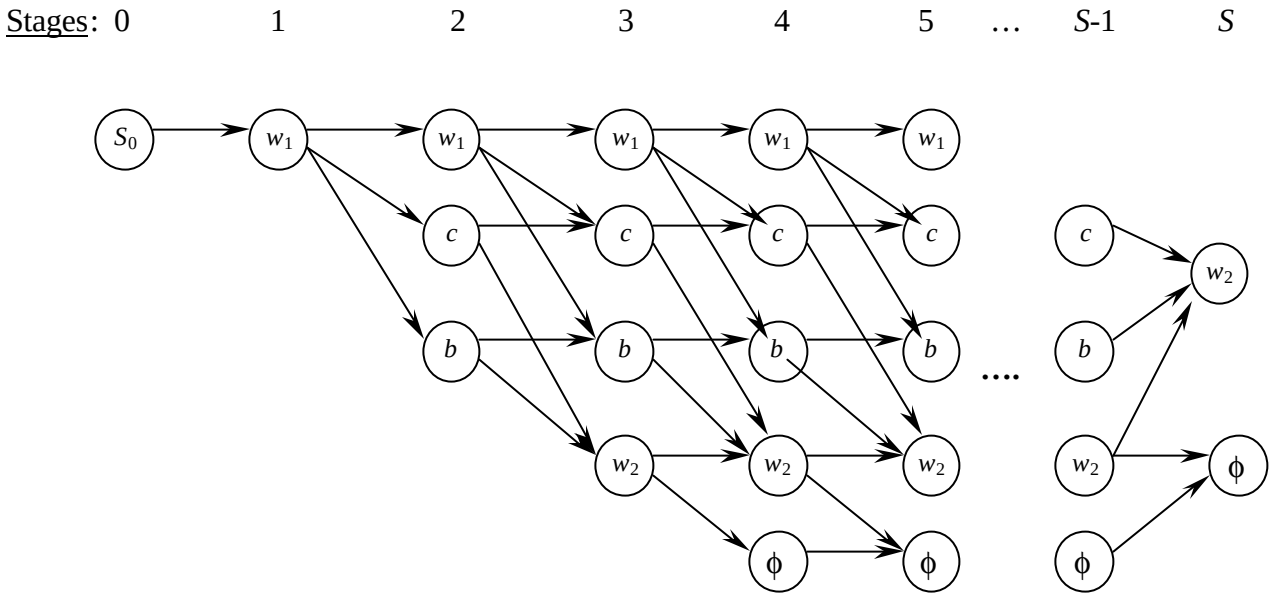
where c_1 and c_2 are duplicates of the label c at Stage 3. Then, the node and arc definition process for G^* above would need to be modified to accommodate nodes of the type (i,l) with $l = c_1$ or c_2 at Stage 3, along with the corresponding arc connections. For example, at Stage 4, we would consider connections of the following type.



Henceforth, we assume that the label and language definitions conform with the definition of a suitable graph G_L for which the paths in G_L correspond to admissible words in L, and vice versa. In this same vein, we might sometimes be given a label transition graph in the form specified below for a trip from a home location to an office, say, for an individual:



The interpretation of this graph is that the individual begins from home and first traverses a series of links having the w (walk) label. The self-loop arc from node w to itself connotes that any possible member of such consecutive walk-label links can be traversed. When a new label transition occurs, this happens along a link that denotes travel via a car (c) or a bus (b). Accordingly, several consecutive links having the respective car-label or the bus-label can be followed next, before finally transitioning again to the walk mode before reaching the office destination. In order to compose an admissible graph G_L in this context, we would need to label the first set of walk-labels that originate from the home as w_1, say, to differentiate these from the terminating walk-label transitions, which we denote by w_2. Furthermore, while the above transition graph itself does not bound the total number of stages or transitions, such an upper bound S would be typically evident in practice, as for example from the transportation network itself. Accordingly, the expanded view of G_L in this context would appear as follows.



Note that the required assumption on G_L is satisfied. Furthermore, we implicitly view the above structure of G_L within the framework of the foregoing transition graph itself. ☒

Figure 4 presents a flow-chart summarizing the essential steps of a rudimentary algorithm (including an efficient construction of G^*) for solving TILSP.

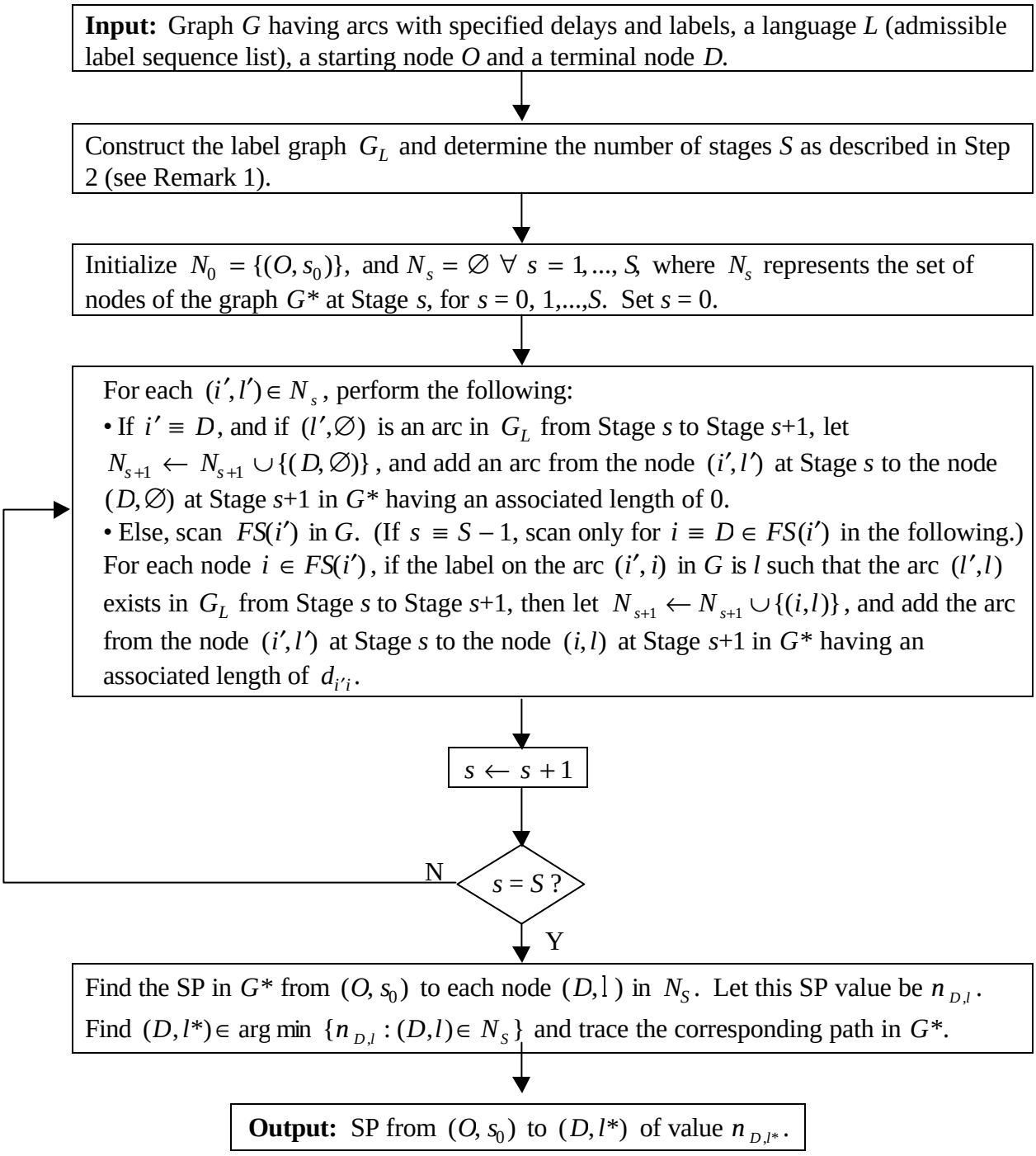
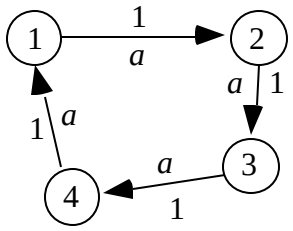
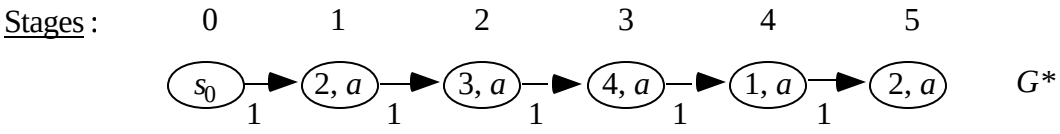


Figure 4: Flow-Chart for a Rudimentary Procedure to Solve the TILSP Problem.

Remark 2. As Barrett et al. point out, a label constrained SP can be *nonsimple*. (The problem of finding shortest *simple* label constrained paths, if they exist, is NP-hard.) For example, consider the graph G given below, with $L = \{(aaaaa)\}$, and with nodes 1 and 2 as the starting and terminal nodes O and D, respectively. The shortest label constrained path is $1 \to 2 \to 3 \to 4 \to 1 \to 2$, which is nonsimple.



This path, however, comes from the following shortest *simple* path in G^*.



Note how nodes can repeat at different stages (e.g. node (2, a) appears at stages 1 and 5).⊠