*May 4th, 2018*

**Solving for user equilibrium**

**5.1 Heuristic equilibration techniques**

The heuristic approaches to the user-equilibrium problem reviewed in this section include ***capacity constraint method*** and ***inremental assignment techniques.*** At the core of these methods lies the *network loading mechanism*. The process follows the route-choice criterion, which underlies any traffic assignment model. The network loading mechanism assigns each O-D flow to the *shortest travel time path* connecting this O-D pair, which is kwon as the "***all-or-nothing" assignment***.

**Capacity restraint**

The capacity restraint involves a repetitive all-or-nothing assignment in which the travel time resulting from the previous assignment are used in the current iteration. The algorithm can be summarized as follows:  
Step 0: *initialization*. Perform all-or-nothing assignment based on . Obtain a set of link flows . Set iteration counter .

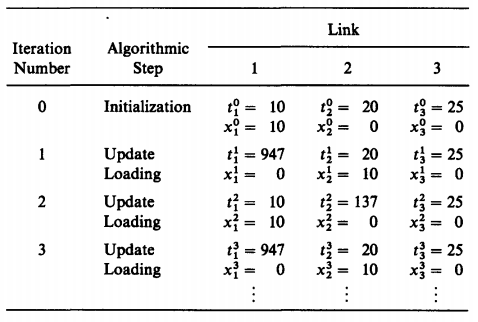
Step 1: update. Set .

Step 2: network loading. Assign all trips to the network using all-or-nothing based on travel times . This yields a set of link flows .

Step 3: convergence test. If , stop (the current set of link flows is the solution). Otherwise, set and go to step1.

Table 5.1.1 demonstrates an application of this procedure to the example network depicted in Figure 5.1.1. However, the algorithm does not converge, as the flow "flip-flops" between links 1 and 2, whereas link 3 does not get loaded at all.

Table 5.1.1 capacity restraint algorithm applied to the network in Figure 5.1.1



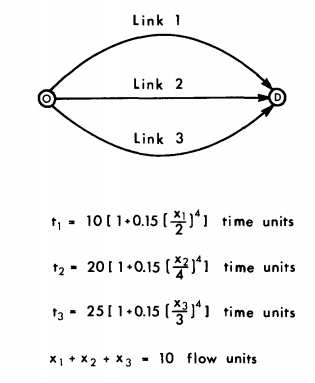


Figure 5.1.1 network example, with three links and one O-D pair.

To remedy this situation, the algorithm can be modified as follows. First, instead of using the travel time obtained in the previous iteration for the new loading, a combination of the last two travel time obtained is used. Second, the failure to converge is recognized explicitly and the algorithm is terminated after a given number of iterations, . The equilibrium flow pattern is then taken to be the average flow for each link over the last four iterations.

The steps of the modified capacity restraint algorithm are as follows (using weights of 0.75 and 0.25 for averaging process):

Step 0: *initialization*. Perform all-or-nothing assignment based on . Obtain a set of link flows . Set iteration counter .

Step 1: *update*. Set .

Step 2: *smoothing*. Set .

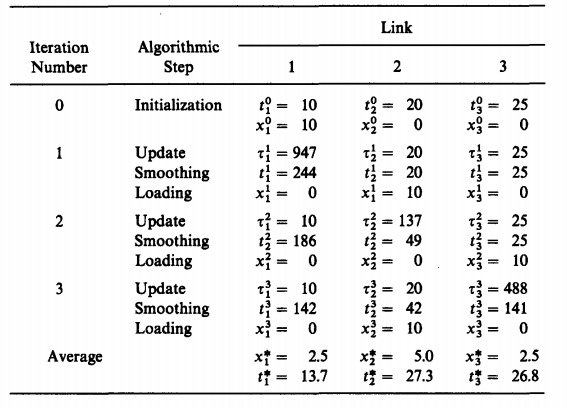
Step 3: *network loading*. Assign all trips to the network using all-or-nothing based on travel times . This yields a set of link flows .

Step 4: *Stopping rule*. If , go to step 5. Otherwise, set and go to step 1.

Step 5: *Averaging*. Set and stop.

An application of this algorithm to the network example of Figure 5.1.1 is demonstrated in Table 5.1.2. Note that it produces a solution that is not an equilibrium flow pattern since, even though all paths are used,  is substantially different from and .

Figure 5.1.2 modified restraint algorithm applied to the network in Figure 5.1.1



**Incremental assignment**

The incremental assignment method assigns a portion of the origin-destination matrix at each iteration. The steps are as follows ( denotes the flow on link  resulting from the assignment of the  increment of the O-D matrix onto the network ):

Step 0: preliminaries. Divide each origin-destination entry into equal portions (i.e. set ). Set and .

Step 1: update. Set .

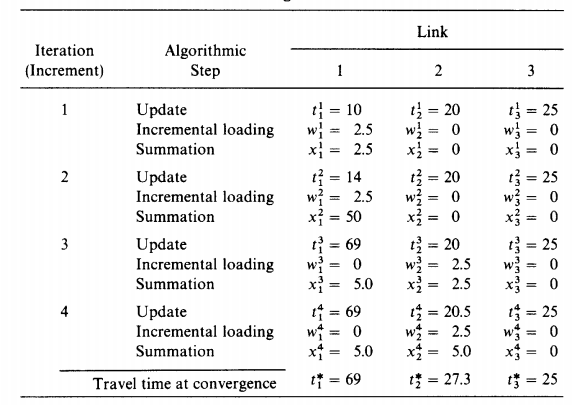
Step 2: inremental loading. Perform all-or-nothing assignment based on , but using only the trip rates for each O-D pair. This yields a flow pattern .

Step 3: flow summation. Set .

Step 4: stopping rule. If , stop (the current set of link flows is the solution); otherwise, set and go to step 1.

Table 5.1.3 demonstrates the application of this algorithm to the network example of Figure 5.1.1. However, the resulting flow pattern is not a equilibrium flow pattern. As the number of increments grows, the incremental assignment algorithm may generate a flow pattern closer to the user-equilibrium condition.

Table 5.1.3 incremental assignment algorithm applied to the network in figure 5.1.1



**5.2 Applying the convex combinations method**

The convex combinations method is especially suitable for solving the equivalent UE problem since the direction finding step is efficient relatively.

The UE program is given by

[5.1a]

Subject to

[5.1b]

[5.1c]

The definitional constraints

[5.1d]

Applying the convex combinations algorithm to minimization the UE program requires, at every iteration, a solution of the linear program

[5.2]

Over all feasible values of .

The gradient of with respect to the link flows at the iteration is the link travel time vector, since . The LP objective function at the iteration thus becomes

[5.3a]

Subject to

[5.3b]

[5.3c]

Where and .

The program [5.3] calls for minimizing the total travel time over a network with fixed travel times, . The total travel time spent in the network will be minimized by assigning all motorists to the shortest travel time path connecting their origin and destination. Consequently, the program in Eqs. [3.5] is known as the all-or-nothing program.

To see that the solution of program [5.3] does not involve more than an all-or-nothing assignment, the linearization step of the convex combinations method can be derived by taking the gradient of objective function of [5.1a] with respect to path flows (instead of link flow, as in [5.2]). The linearized program then becomes

[5.4a]

Subject to

[5.4b]

[5.4c]

Where is the travel time on path connecting and , at the iteration of the algorithm.

This program can be decomposed by O-D pair since the path travel time are fixed. The resulting subproblem for pair is given by

[5.5a]

Subject to

[5.5b]

[5.5c]

This program is obviously minimized by finding the path, ,with the smallest travel time among all paths connecting and , and assigning all the flow to it. In other words,

[5.6a]

And [5.6b]

In case two or more paths are tied for the minimum, any one of them can be chosen for flow assignment.

Once the path flow are found, the link flows can be calculated by using the incidence relationships, that is,



This solution defines the descent direction .

The line search for the optimal move size can be performed with any of the interval reduction methods, but the bisection method may be particularly applicable. The reason is that the derivative of the objective function with respect to is given by



Which can be easily calculated for any value of .

The stopping criterion for solving the UE program could be based on the values of the objective function. However, this function is merely a mathematical construct that lacks behavioral or economic meaning. A possible measure of the closeness of a particular solution to equilibrium is the similarity of successive O-D travel times. Letting denote the minimum path travel time between O-D pair at the iteration, the algorithm can terminate if, for example,

[5.8a]

Alternatively, a criterion that is based on the change in flows can be used. For example, the algorithm can terminate if

[5.8b]

The algorithm itself, when applied to the solution of the UE problem, can be summarized as follows:  
step 0: *initialization*. Perform all-or-nothing assignment based on . Obtain a set of link flows . Set iteration counter .

Step 1: update. Set .

Step 2: direction finding. Perform all-or-nothing assignment based on . This yields a set of flows .

Step 3: line search. Find that solves



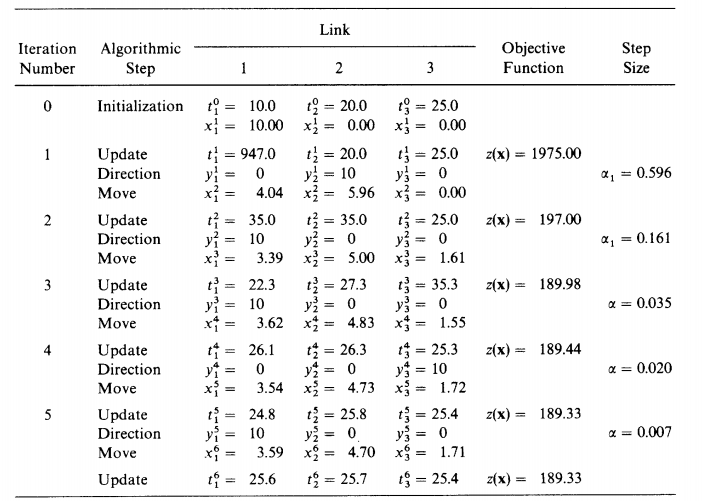
Step 4: move. Set .

Step 5: convergence test. If a convergence criterion is met, stop (the current solution , is the set of equilibrium flows); otherwise, set and go to step 1.

Note the similarity between this algorithm and the capacity constraint method. In fact, if the move size, , is fixed at  for all , the resulting algorithm is identical to the capacity constraint method.

To demonstrate the convergence of this algorithm to the equilibrium solution, it is applied to the three-link example depicted in Figure 5.1. It is evident that after five iterations the flows are close to equilibrium.

Table 5.2.1 convex combinations method applied to the network in Figure 5.1.1

  
The number of iterations required for convergence is significantly affected by the congestion level on the network, which is demonstrated in Figure 5.2.1. In actual applications, only four to six iterations are usually sufficient to find the equilibrium flow pattern over large urban networks.

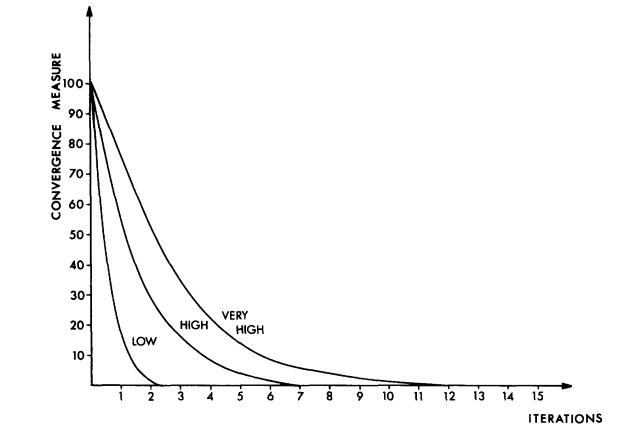


Figure 5.2.1 convergence rate of the convex combinations algorithm for various congestion levels.

**5.3 shortest paths over a network**

**Shortest-path algorithm**

The algorithm presented here is known in the operations research literature as the *label-correcting method* (标号修正法) *.* It finds the shortest path from a given origin node to all other nodes in the network. Accordingly, it has to be used for each origin in turn for a complete all-or-nothing assignment to be performed.

The information about network stored in the computer are shown as follows:  
(1) a list of links identified by their end nodes, for example .

(2) a travel time is associated with each link .

(3)for every node : 1)the current label of this node, , which is the distance from the root node to node  along the (current) shortest path. 2) its predecessor node, .

To help manage and keep track of the nodes, the algorithm uses an additional list called the sequence list, which includes all the nodes that have yet to be examined as well as the nodes requiring further examination.

The steps of the label-correcting method are as follows:  
step 0: *initialization.* Set all labels to infinity(which are arranged in a label list) and all predecessor nodes to zero(in the predecessor list). Place the origin node ,, on the sequence list with label .

Step 1: *update*. All nodes, , that can be reached from by traversing only a single link are tested in the examination process. If the minimum path to through is shorter than the previous path to , then is updated. In other words, if



Then the current shortest path from the root node to can be improved by going through node .

The label list is updated setting , the predecessor list is updated by setting . Also, the sequence list is updated by adding to it. Once all nodes (that can be reached from )are tested, the examination of node is complete and it is deleted from the sequence list.

Step 2: if the sequence list is empty, stop. Otherwise, return to step 1.

**Example**

Consider the simple network depicted in Figure 5.3.1. The contents of label list, the predecessor list and the sequence list for these iterations are given in Table 5.3.1.

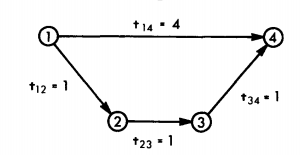
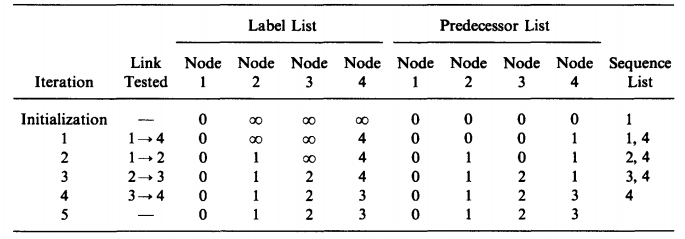
****

Figure 5.3.1 network example for the minimum-path algorithm

Table 5.3.1 contents of the label, predecessor and sequence list

****