Chapter 30:

Crime Network Assignment

Ned Levine
Ned Levine & Associates
Houston, TX

Table of Contents

Table of Contents (continued)

Chapter 30:
Crime Network Assignment

In this chapter, the fourth, and last, component of the crime travel demand model will be described. *Network assignment* involves the assigning of predicted trips to particular routes. The predicted trips are those that are either predicted from the trip distribution stage or from the mode split stage. In the former case, all trips from each origin zone to each destination zone are assigned to a particular travel route, usually on the assumption that they all travel with the same mode of travel (usually walking, biking or driving). In the latter case, the predicted trips from each origin-destination zone pair by specific travel modes are assigned to a particular route which is mode specific. Thus, bus trips are assigned to bus routes; train trips are assigned to train routes; driving trips are assigned to a road network; walking trips are assigned to a more limited road network; and biking trips are assigned to a mixture of roads and bike paths. In other words, the assignment of travel modes is specific to a particular network.

Once the trips are assigned to routes, several statistics can be calculated. First, the predicted path from an origin zone to a destination zone can be displayed. This can very useful for police who could increase their patrol on high crime routes. Second, the entire *trip load* on road segments can be calculated. Since many crime trips pass over the same network segments (e.g., freeways, major arterial roads), the total number of predicted trips on individual segments can be enumerated. The result is a map of the most heavily traversed segments in the network. Again, this can be very useful for police.

Thus, the network assignment completes the four stage modeling process of the crime travel demand framework. To summarize, in the first stage - trip generation, separate models of the number of crimes originating in each zone and the number of crimes ending in each zone are developed. In the second stage - trip distribution, the predicted number of crimes originating in each zone are allocated to each destination zone; the result is a prediction of the number of trips that occur between each origin-destination zone pair. In the third stage - mode split, each predicted origin-destination trip pair is separated (split) into distinct travel modes (e.g., walking, biking, driving, bus, train) with the result being a mode-specific origin-destination zone pair. Finally, the fourth stage - network assignment, assigns these trips to specific routes.

Theoretical Background

To understand the background, we need to look, first, at the nature of networks and second, at types of routing algorithms.

Networks

The most fundamental element of assignment is, of course, a network. The network can be a road network, a bus network (e.g., bus routes with stops), a train network (e.g., train lines with stations), or even a bicycle network (e.g., a mixture of roads and bicycle paths). Other kinds of networks can also be considered, for example telecommunication lines or even trade routes. We will concentrate on urban transportation networks, however.

The mathematical properties of networks are known as *graph theory* (Sedgewick, 2002). A network (or graph) is a set of nodes (or vertices) and a set of segments (or edges) that connect pairs of nodes. If there are V nodes (vertices), then there are V^2 pairs of nodes, including the distance from a node to itself. A graph with V nodes has, *at most*, $V(V-1)/2$ segments (edges); if multiple segments share nodes, then there will be even fewer.

Figure 30.1 illustrates a simple network. Travel occurs along the segments through the connecting nodes. A path is a sequence of nodes in which each successive node is connected to its predecessor in the path. hus, in the figure, there cannot be direct travel between node A and node C, but must go through an intermediate node (e.g., through B or through a path from D to E to C).

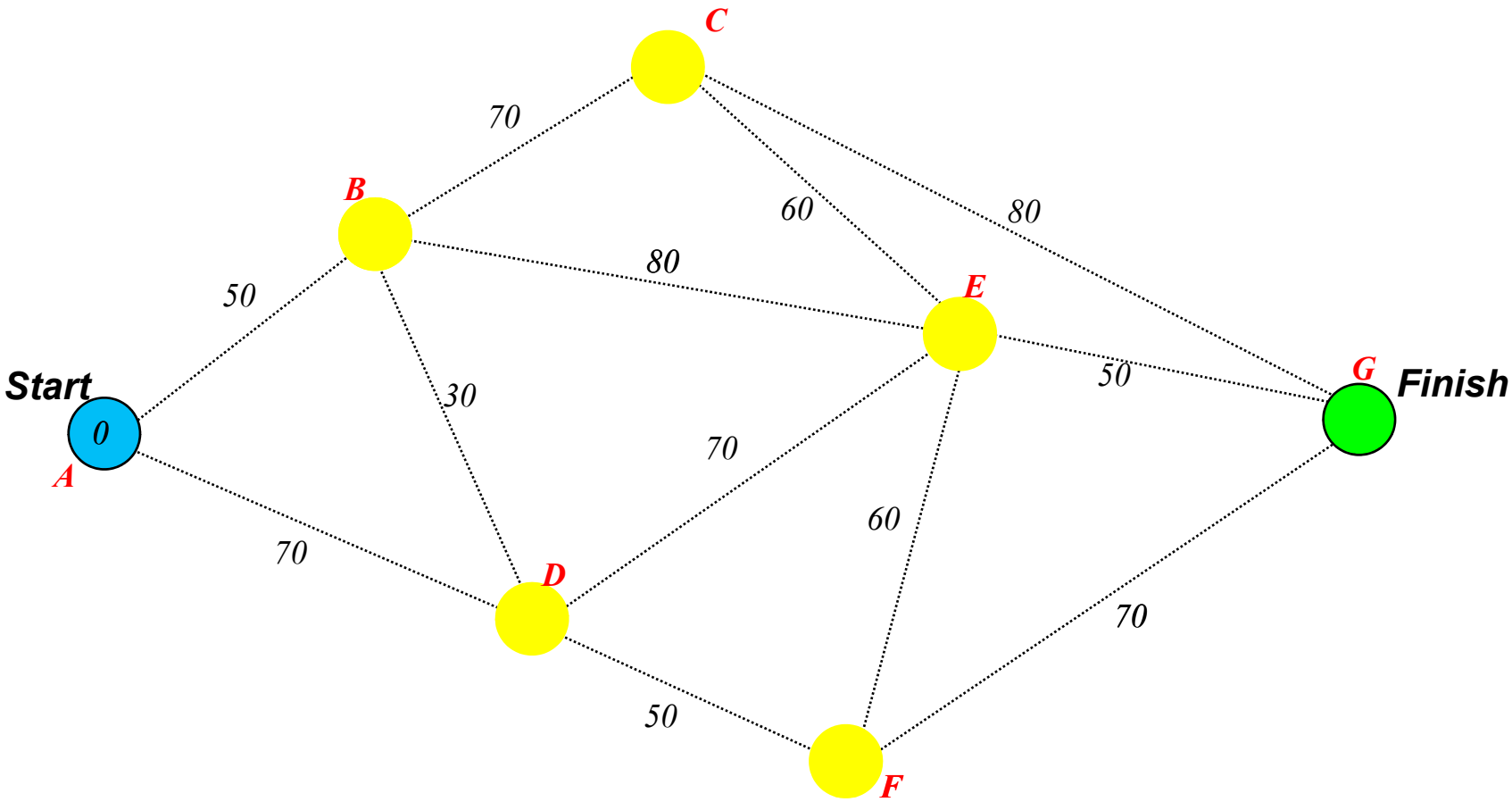
Impedance of a Network

There are several properties of a network that are important for travel modeling. First, the *length* of a segment is proportional to its impedance (see Chapters 28 and 29). The simplest kind of impedance is distance in which each unit length of the network corresponds to some unit of distance in the real world (e.g., one inch = 1 miles; one centimeter = 5 kilometers). This is analogous to the *scale* used in mapping systems. More complex types of impedance involve travel time, speed, or even generalized cost (a collection of several cost elements). Thus, to use the example in Figure 30.1, node A is connected to nodes B and D. The path from A to B is 50 units long; similar lengths are found for the other segments in the example. This could represent distance (e.g., 50 miles), travel time (e.g., 50 minutes), or generalized cost (e.g., $50).[1] To a graph, the units are irrelevant. As long as the user is explicit about these and consistent, path calculations will work properly.

[1] Speed could be used, but it is inversely proportional to impedance (i.e., the higher the speed, the less the impedance). Most shortest path algorithms treat the weight as proportional. However, speed can be converted into travel time by dividing distance by speed. To use the example, if the length is 1 mile long and the speed is 50 miles per hour, then the travel time is 1/50 hours (or 1.2 minutes).

Figure 30.1:

A Simple Network



Bi-directional and Single Directional Networks

Bi-directional networks

Second, typical transportation networks are either *bi-directional* or *single directional*. In a bi-directional network, travel can occur in either direction. Again, using Figure 30.1, if the network is bi-directional, then travel can occur from A to B or from B to A. A well known example of a bi-directional network is the TIGER system of the U.S. Census Bureau (2011). This is a representation of all major urban lines, including streets, railroad lines, census geography boundaries, jurisdictional boundaries, Congressional boundaries, and other features. It is used to map out census areas for the purpose of collecting the decennial Census. Virtually the entire United States is now mapped in the TIGER system. Depending on how carefully each jurisdiction updates the database for new roads and changes in existing roads, the TIGER system can be a very accurate spatial representation of the an urban road system. It is a widely available system and is often the first network that most police departments use when they create a crime mapping system. Figure 30.2 shows a TIGER network for Baltimore County and the City of Baltimore. There are 49,015 road segments in the TIGER map shown in the figure.

Problems with the TIGER system for travel modeling

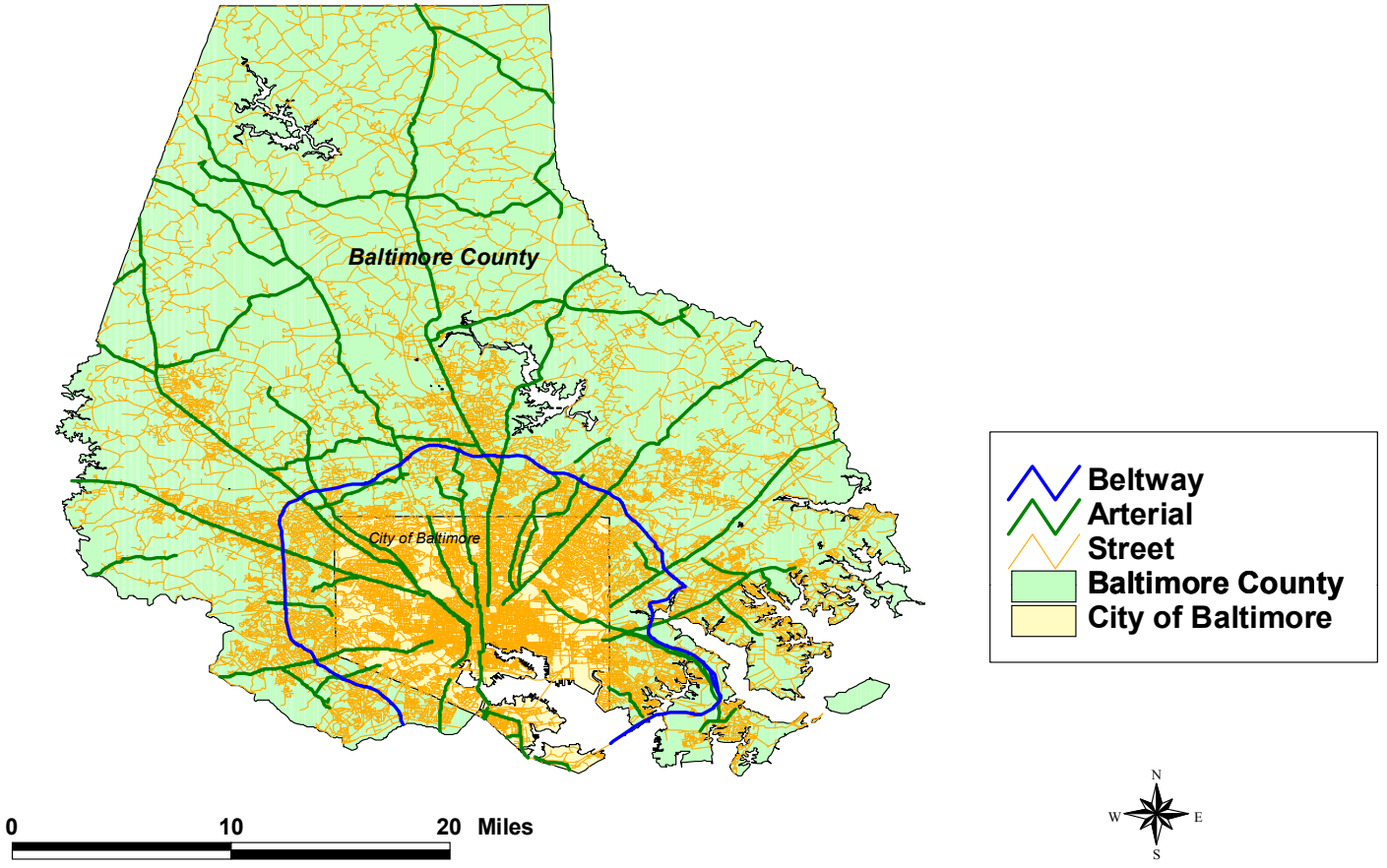
On the other hand, for travel modeling, there are substantial problems with bi-directional networks and with TIGER in particular. A major problem is that *connectivity* is often not tested. Since the aim of the TIGER system is to represent a metropolitan area for the purpose of collecting the Census, connectivity is not guaranteed since it is irrelevant for that purpose. It is not clear that all roads are properly represented, a feature that could substantially alter a shortest path algorithm. For example, in Figure 30.1, if the segment from A to B was not connected, then travel from A to C would have to take a circuitous path from A to D to E to C. Having an accurate and edited network is critical for modeling travel behavior. With a large number of segments in a TIGER system, it is often not clear where in a file connectivity is not properly linked.

Another problem is that TIGER is typically less accurate with respect to rail lines and has virtually no information about bus routes, which are local in nature. Depending on how diligent the local government is in updating the database, the representation may not be as accurate as possible (though, in general, it is getting better over time).

Another major deficiency of the TIGER system is the lack of information about travel time or travel cost. Travel along a TIGER network is defined by distance, which does not change by time of day. It does not have cost information either, which makes it less flexible for examining alternative routes as a function of additional cost factors (e.g., an analysis of travel

Figure 30.2:

TIGER Street Network
49,015 Road Segments



through an area with high surveillance compared to travel through an area with low security presence even if travel through the first area is shorter in time than through the second area). The TIGER system does have information about functional class of road (interstate, state highway, collector road) and it is possible to assign *a priori* speeds to the different segments based on these classes (e.g., 35 miles per hour for Interstate highways, 25 miles per hour for principal arterial roads). But, because the network is bi-directional, it is impossible to assign speeds for travel in opposite directions; in reality, there are usually differences in travel speeds in opposite directions (e.g., travel into the central business district in the morning might be at 15 miles per hour whereas travel in the opposite direction might be at 35 miles per hour).

Another major problem with TIGER and with a bi-directional network in general is in the representation of one-way streets. The TIGER system does not provide this information. Consequently, in using a TIGER file for modeling travel, a shortest path could easily travel up a one-way street in the wrong direction. To make the system work properly, there needs to be an additional field in the database that identifies a segment as one-way.

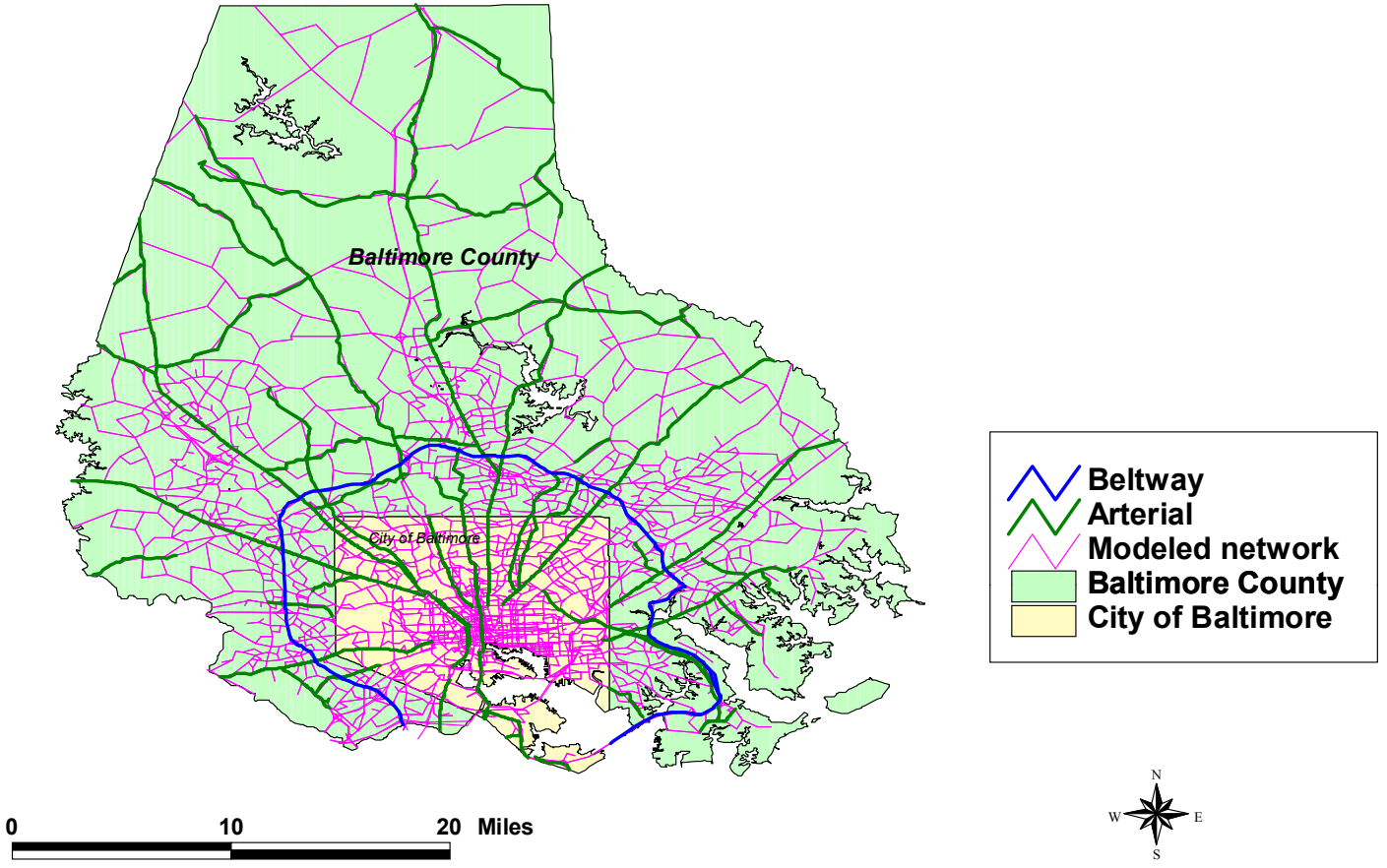
Single directional networks

A single directional (or uni-directional) network, on the other hand, allows travel in only one direction. This has the advantage of keeping travel consistently defined. Two-way travel is represented by two segments, one in each direction (e.g., one for travel from A to B and one for travel from B to A). One-way streets can be characterized by only one of the paired directions. Most transportation modeling networks are single directional since an accurate representation of travel is critical. Travel times, speeds or costs can be assigned to the different directions of travel between two nodes and can be further assigned to different times of the day (e.g., 20 miles per hour in the morning peak period, 15 miles per hour in the afternoon peak period, 30 miles per hour in the off-peak daytime period, and 45 miles per hour at nighttime).

An example of a single directional network is that used for travel demand modeling by most Metropolitan Planning Organizations (MPO). These are used to model travel over an entire metropolitan area (regional travel) and are generally updated regularly; connectivity is continuously tested and errors are few in number. The travel modeling network is usually a 'skeleton' network, covering all the major roads - freeways, principal arterial roads, minor arterial roads, and some collector roads. They usually do not include much information about local or neighborhood streets since these are not very relevant for regional travel modeling. Figure 30.3 shows a modeling network used by the Baltimore Metropolitan Council for their travel demand model. There are only 11,045 road segments in the file, less than one fourth the size of the corresponding TIGER network. Considering that each segment in a single direction, effectively only about 5,000-6,000 actual roads are being represented in the file.

Figure 30.3:

Modeled Street Network
11,045 Road Segments



Most importantly, modeling networks usually include information about travel time or travel speed (which can be converted to travel time by dividing distance by speed) and are usually broken down into different time periods. Thus, it becomes possible to analyze travel at different times of the day to account for the major congestion effects that occur at the peak travel periods, particularly the afternoon peak. Some modeling networks also include information on travel costs, which include parking, toll roads, and other costs that impact a trip. As mentioned in Chapter 26, any analyst wishing to develop a crime travel demand model should contact the local MPO about obtaining a copy of the modeling network used.

> Hint: A single directional network can also be treated as bi-directional. In this case, all the trips on that roadway will generally be assigned to only one of the paired segments (for a two-way pair). For the network load output, particularly, this can be useful for showing the total number of trips on a road segment, independent of direction. Otherwise, if defined as a single directional network, the loads in each direction will be displayed separately.

Problems with modeling networks

Modeling networks also have their problems. The biggest one is that they do not include all roads, but only the more important regional ones. This can lead to unrealistic paths being modeled in a neighborhood (e.g., entering or leaving a neighborhood from a centroid, rather than from a real street; taking circuitous routes to travel a short distance in space when, in fact, there are connecting local roads that actually exist but are not included in the file). However, neighborhood roads can usually be added to the network to provide more detail at the neighborhood level and to correct modeling errors. It is a tedious process, but a police department could slowly update such a system over time and improve its accuracy. Care must be taken in doing this, however, to ensure that connectivity is correctly portrayed.

Another problem, which may or may not be critical, is that the representation of roads in a modeling network is spatially simplified. Road segments are straight lines, rather than having curvature. In the TIGER system, the basic record of a segment is a straight line connecting two nodes, but also includes up to 10 intermediate 'shape grammar' nodes that define curvature (integrated with spatially more accurate information from the U.S. Geological Survey). Thus, a modeling network looks a little 'unreal' at a neighborhood level since there are only straight lines. But additional segments can be added to the file to improve local connectivity as well as familiarity.

Transportation Networks

The third property of a network for travel modeling is the type of network. Road networks were mentioned above. But there are also transit networks (e.g., bus routes, train routes) and even bicycle networks (e.g., bike paths). If a trip distribution matrix of trips from origins to destinations is analyzed by travel mode, then it is critical to have a mode-specific network. Using TIGER or a simple modeling network will imply that all trips occur by the existing road system. For transit trips (bus and rail) particularly, but also for biking trips and possibly walking trips, features that are specific to the travel mode must be included. Bus routes will use the existing road system, but they do not use all roads, typically only the major arterial roads. Train systems rarely use the existing road system, but usually have dedicated tracks. There are exceptions. Some light rail systems do run on arterial roads. Other rail systems will run on an arterial road, but with a grade separation. Depending on how the MPO conceptualizes this, there may be separate lines for the rail or not.

Thus, it is very important to check and edit all networks that are used. For transit networks, in particular, the lines need to be connected and thoroughly tested. Figure 30.4 repeats the Baltimore bus network map from Chapter 29 (figure 29.13). Each of the lines on the map represent bus routes; there can be (and usually are) more than one bus route running on any one line. Typically, these are drawn as separate line objects and are overlaid on each other. This particular network does not have information about bus stops. Consequently, a shortest path algorithm will choose the end nodes of segments to allow a trip to "enter" or "leave". Thus, it is possible that a bus trip would start at a location where there is not a bus stop. However, given that buses in Baltimore and elsewhere stop very frequently (every two or three blocks on average), the amount of error introduced is quite small.

With trains, however, it is absolutely critical that station locations be used to define the rail lines; people cannot enter or leave a train between stations. Figure 30.5 shows each of the four intra-urban rail lines with the station locations. Later in this chapter, there will be a discussion of a utility for creating rail lines from station locations. But a critical point is that each of the end points of the rail segments be associated rail stations. In the figure, each of the four rail lines is shown in separate color. For modeling in *CrimeStat*, however, the individual lines need to be merged into a single file in order for the shortest path routine to be able to move between rail lines (i.e., if there are separate line objects for each line, the routine will not know how to move from one line to another). Figure 30.6 shows the full rail line network.

Shortest Path Algorithms

Once a network has been created, edited and thoroughly tested for accurate connectivity, it can be used for a shortest path analysis. In a *shortest path* for a single trip (from an origin

Figure 30.4:

Baltimore Bus Network

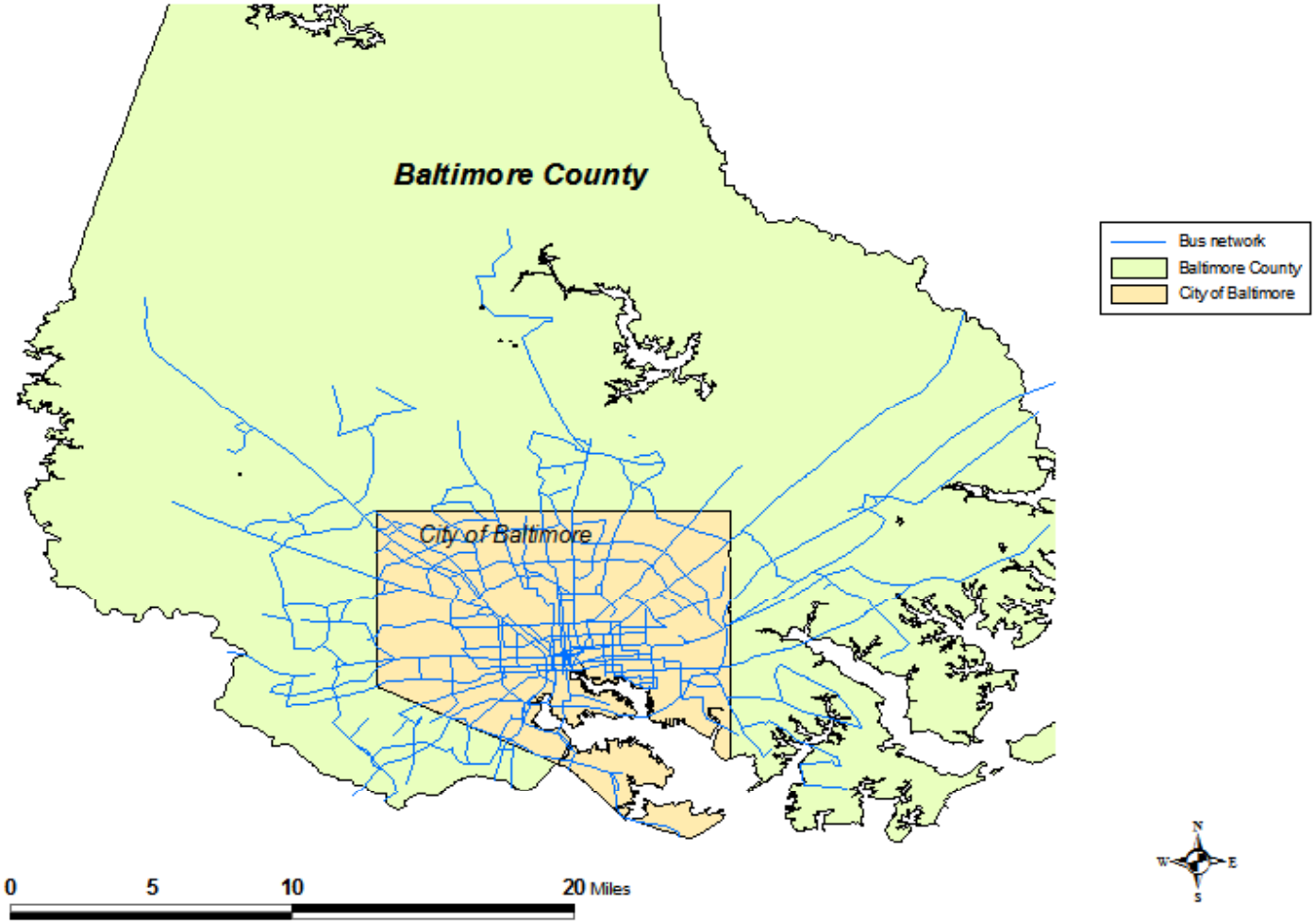


Figure 30.5:

Baltimore Intra-urban Rail Network: 2004
Intra-urban Lines and Rail Stations

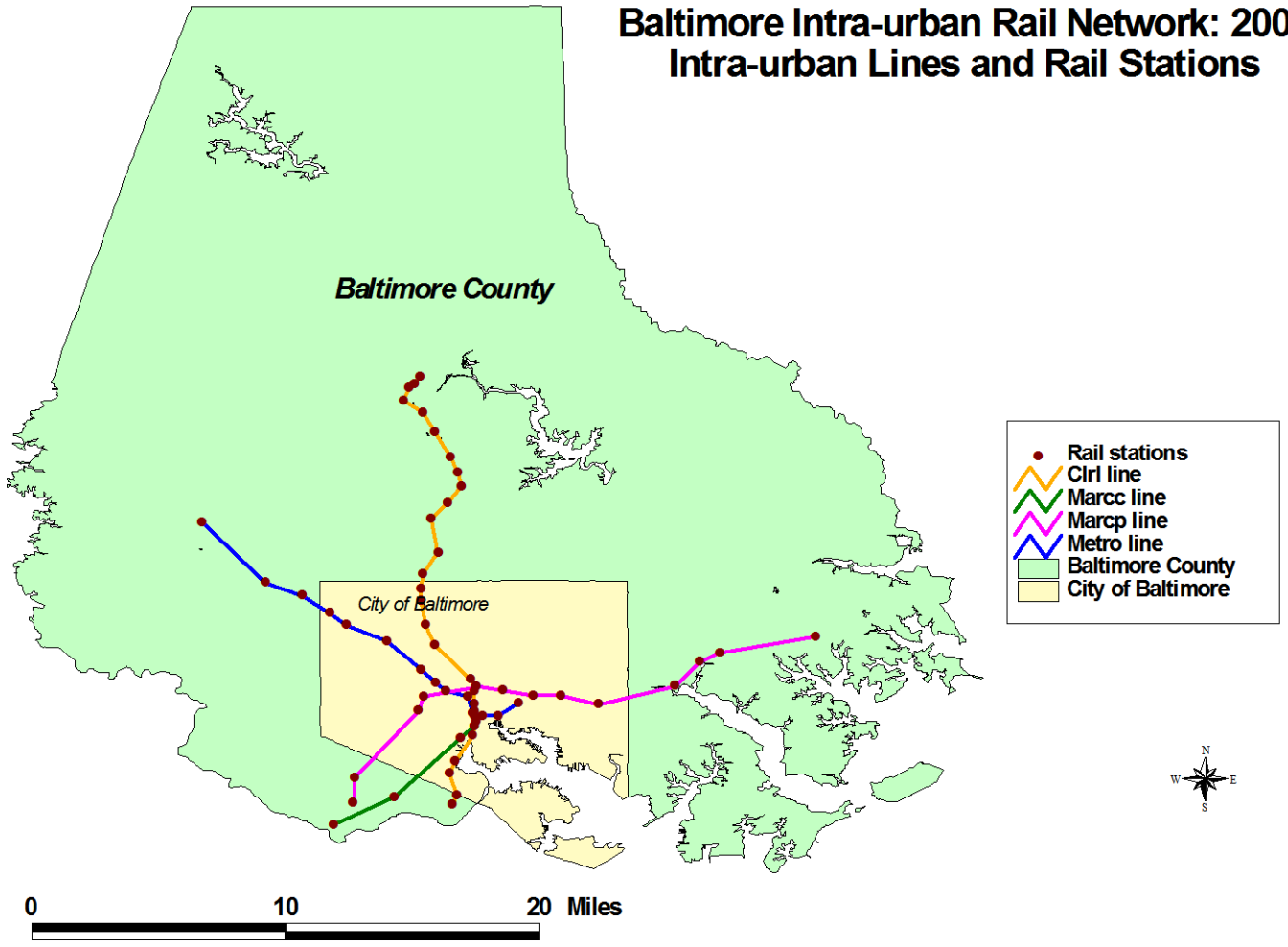
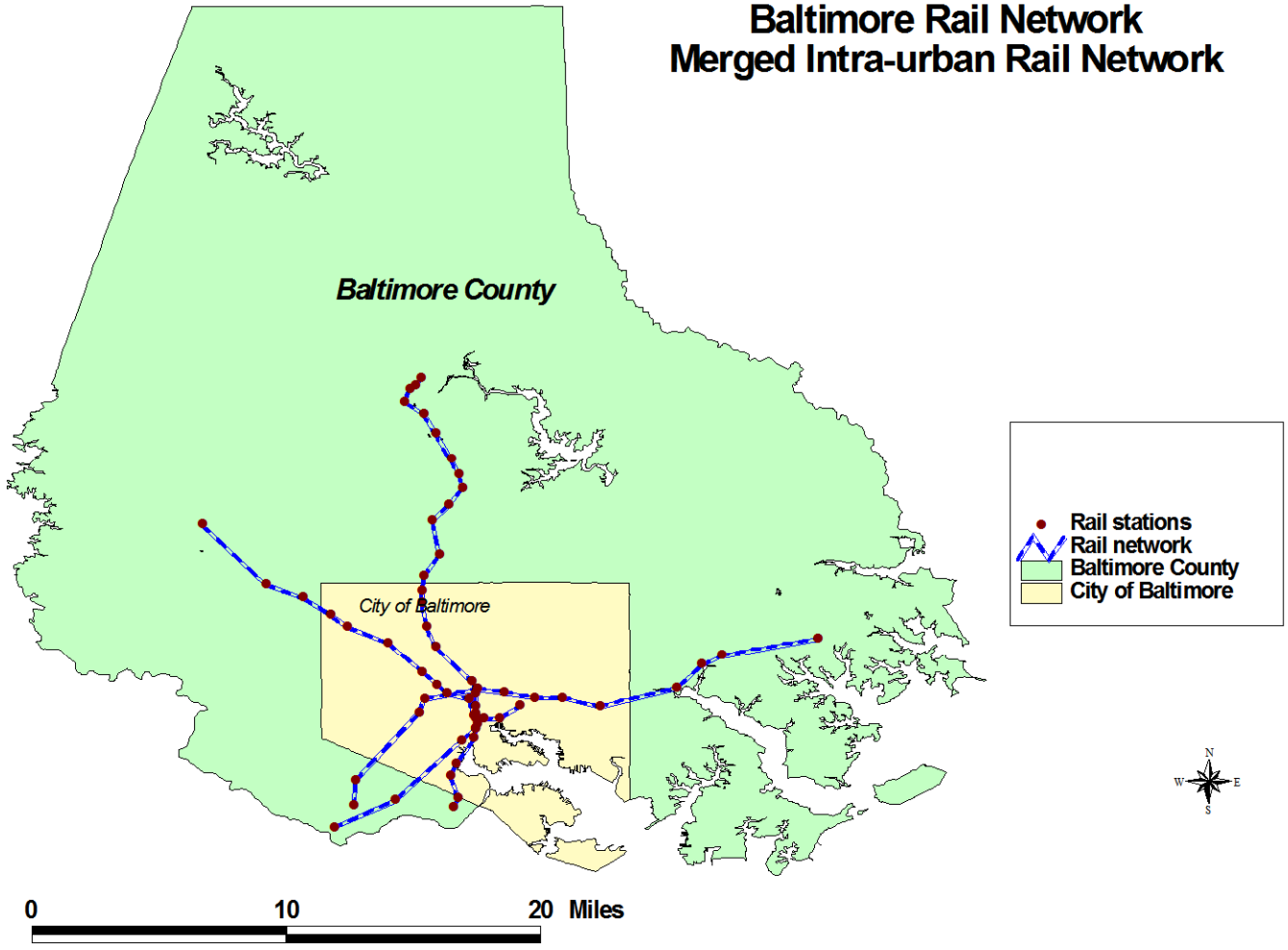


Figure 30.6:

Baltimore Rail Network
Merged Intra-urban Rail Network



zone to a destination zone), the route with the lowest overall impedance is selected. As mentioned, impedance can be defined in terms of distance, travel time, speed, or generalized cost.

There are a number of shortest path algorithms that have been developed (Sedgewick, 2002). They differ in terms of whether they are breadth-first (i.e., search all possibilities) or depth-first (i.e., go straight to the target) algorithms and whether they examine a one-to-many relationship (i.e., from a single origin node to many nodes) or a many-to-many relationship (All pairs; from each node to every other node).

The algorithm that is most commonly used for shortest path analysis of moderate-sized data sets (up to a million cases) is called *A**, which is pronounced "A-star" (Nilsson, 1980; Stout, 2000; Rabin 2000a, 2000b; Sedgewick, 2002). It is a one-to-many algorithm but is an improvement over another commonly-used algorithm called *Dijkstra* (Dijkstra, 1959). Therefore, I will start first by describing the Dijkstra algorithm before explaining the A* algorithm.

Dijkstra Algorithm

The Dijkstra algorithm is a one-to-many search strategy in which a shortest path from a single node to all other nodes is calculated. The routine is a breadth-first algorithm in that it searches all possible paths, but builds the path one segment at a time. Starting from an origin location (node), it identifies the node that is nearest to it **and** which has not already been identified on the shortest path. After each node has been identified to be on the shortest path, it is removed from the search possibilities. The algorithm proceeds until the shortest path to all nodes has been determined. In terms of a matrix of origin nodes (on the vertical) and destination nodes (on the horizontal - see figure 28.1 in Chapter 28), the search algorithm estimates the shortest path for any one row (i.e., from a particular origin to all destinations).

The algorithm can also be structured to find the shortest path between a particular origin node and a particular destination node. In this case, it will quit once the destination node has been identified on the shortest path. The algorithm can also be structured to find the shortest path from each origin node to each destination node. It does this one path at a time (e.g., it finds the shortest path from node A to all other nodes; then it finds the shortest path from node B to all other nodes; and so forth).

The network in Figure 30.1 will be used as an example. Figure 30.7 presents the network in terms of a particular origin node (A = Start) and a particular destination node (G = Finish). In the first step (not shown), the algorithm finds the node that is closest to A that has not already been put on the shortest path. In this case, it is to itself (i.e., A to A is the shortest path at this point). It thus removes A from the list of possible nodes and puts it in a shortest path

Figure 30.7:

Example of Dijkstra Algorithm

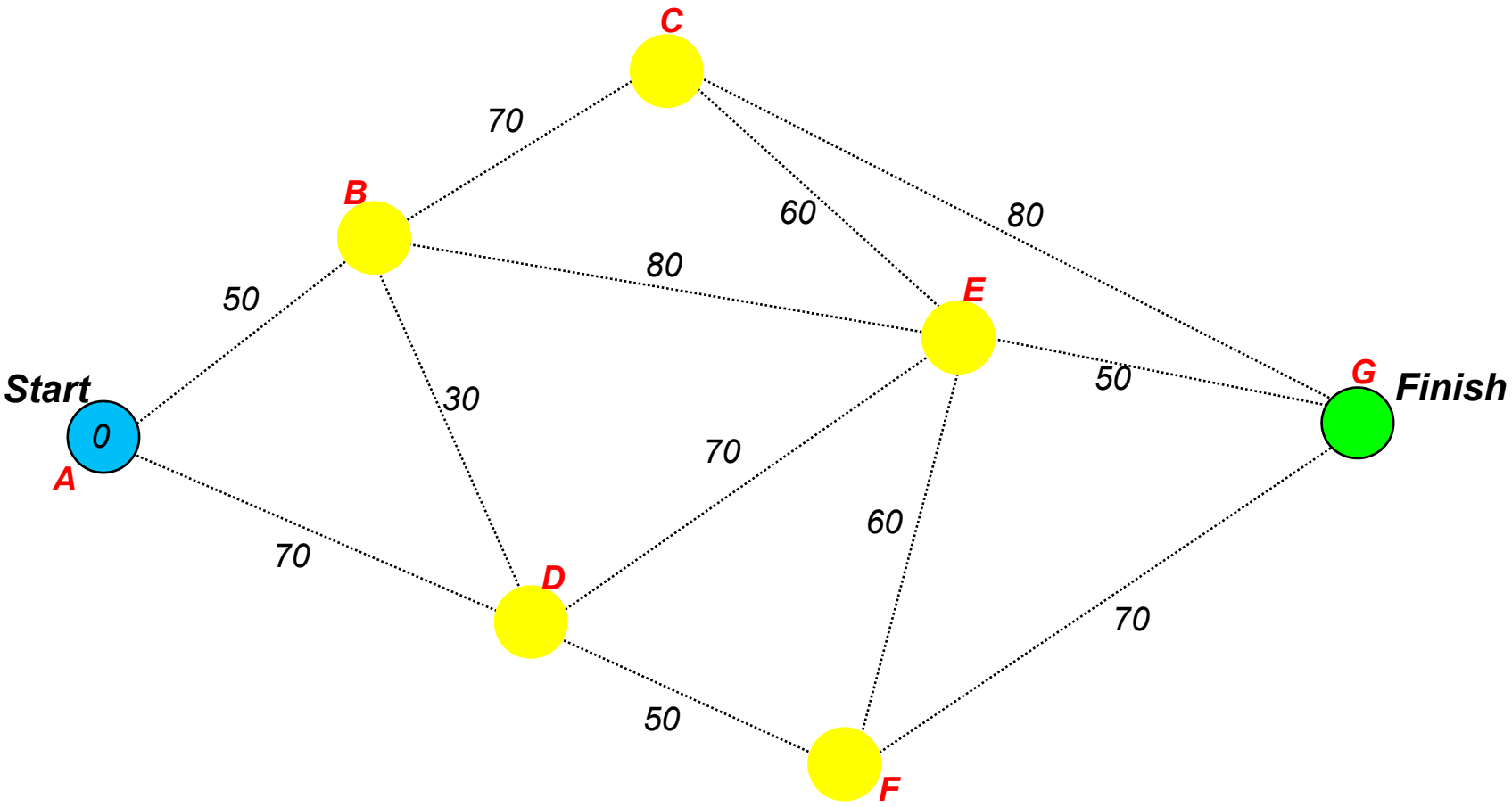
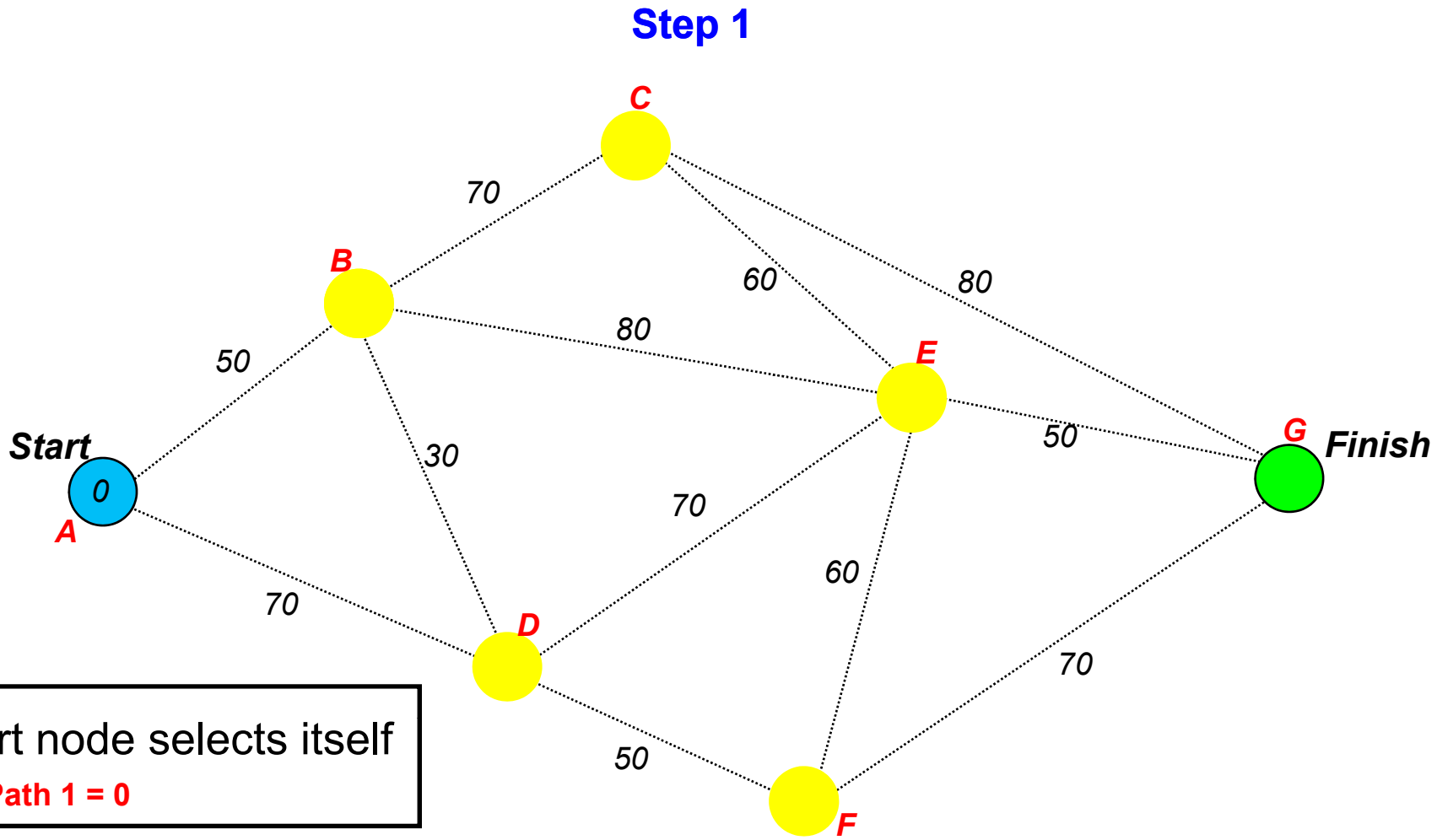


Figure 30.8:

Example of Dijkstra Algorithm



node list. Next, the routine finds the node that is closest to A that has not already been put on the shortest path list. This will be node B, which is 50 units distance from A (Figure 30.8). Thus, the shortest path now goes from A to B. Subsequently, node B is removed from the list of possible new nodes and is put on the shortest path list.

In step 2, the routine finds the node that is closest to one of the existing nodes on the shortest path list but which has not already been put on that list. This will be node D, which is 70 units from A (Figure 30.9). That is, if A and B have already been put on the shortest path list, then only two nodes are connected to these - C and D. The distance from A to C is 120 (50 + 70) while the distance from A to D is 70. Thus, the routine selects node D next. Subsequently, node D is removed from the list of possible new nodes and is put on the shortest path list.

In step 3 (Figure 30.10), the routine determines the node that is closest to A and which has not yet been put on the shortest path. There are two possibilities - C and F; both are 120 units distance from A. In the case of a tie, the routine 'flips a coin' and chooses one, in this case node F. Subsequently, node F is removed from the list of possible new nodes and is put on the shortest path list.

In step 4 (Figure 30.11), the routine adds node C to the shortest path. Note that had the 'coin flip' in step 3 chosen node C instead of F, in this stage node F would have been selected; thus, the routine produces the same solution, just in a different order. Both nodes C and F are 120 units distance from node A. Node C is now removed the list of possible new nodes and is put on the shortest path list.

In step 5 (Figure 30.12), the routine adds node E to the shortest path list because the distance to E through B is shorter than any other route that has not yet been determined (130 units from A). Notice that the path to E through C or D would have been longer than through B (180 and 140 units respectively).

Finally, in step 6 (Figure 30.13), the routine goes to the finish, node G. The path through B and E is shorter than by any other path to G (180 total units). Thus, the Dijkstra algorithm has searched every node in the network and determined a shortest path from node A to each of them (Figure 30.14). Even though we are only interested in the path from A to G, the algorithm solves all shortest paths from A to all nodes.

A* Algorithm

The biggest problem with the Dijkstra algorithm is that it searches the path to every single node. If the purpose were to find the shortest path from a single node to all other nodes,

Figure 30.9:

Example of Dijkstra Algorithm

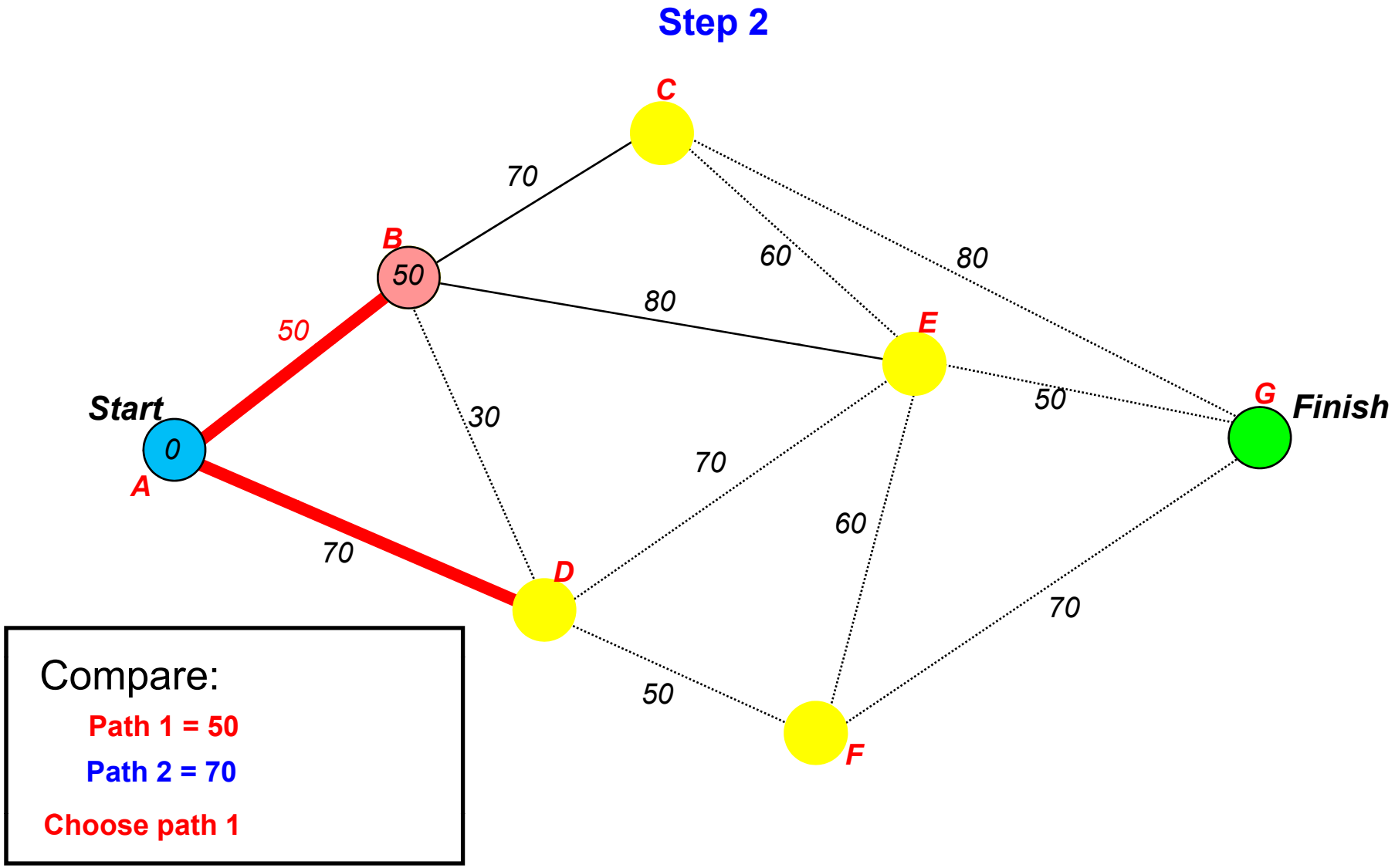


Figure 30.10:

Example of Dijkstra Algorithm

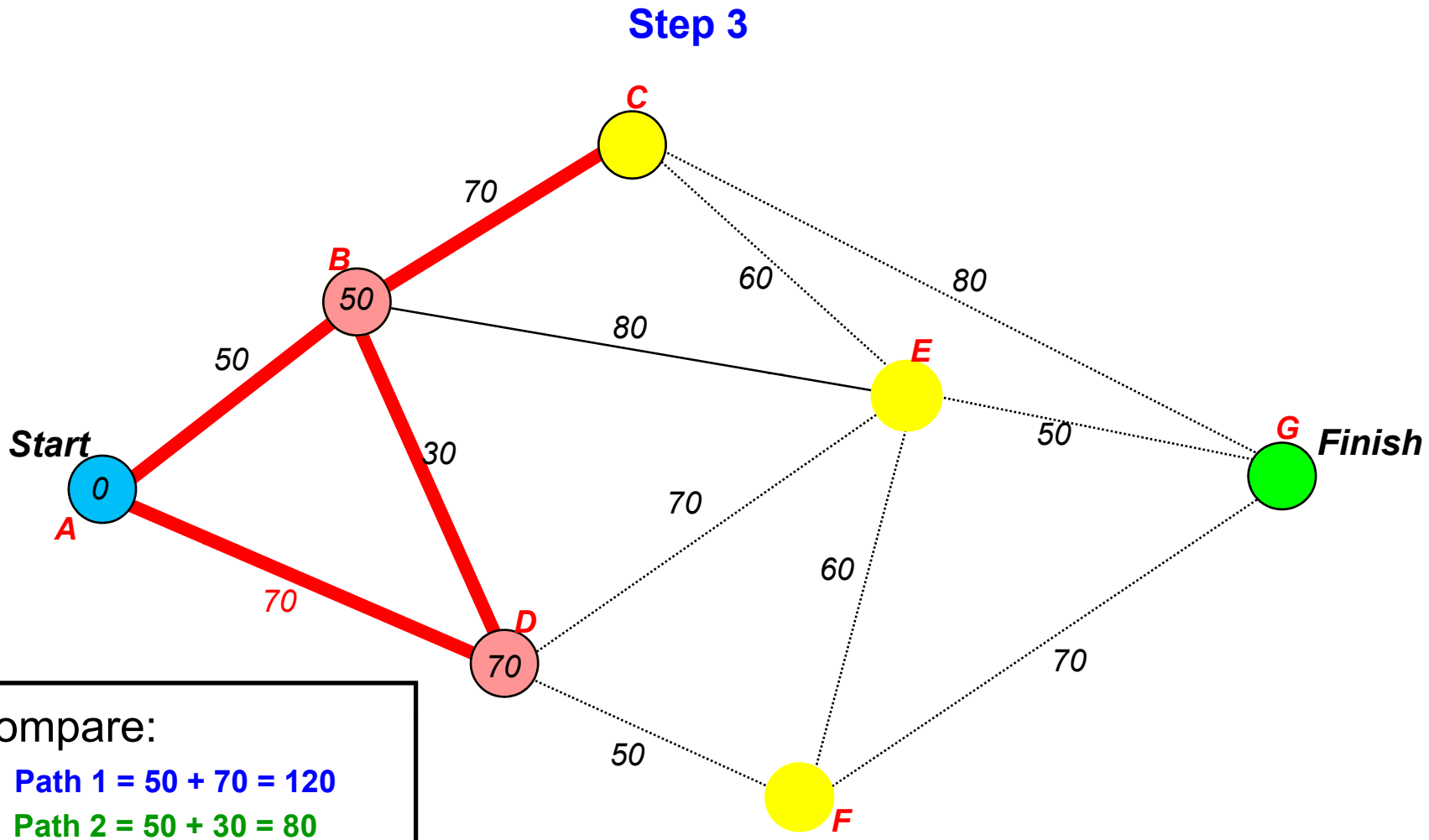


Figure 30.11:

Example of Dijkstra Algorithm

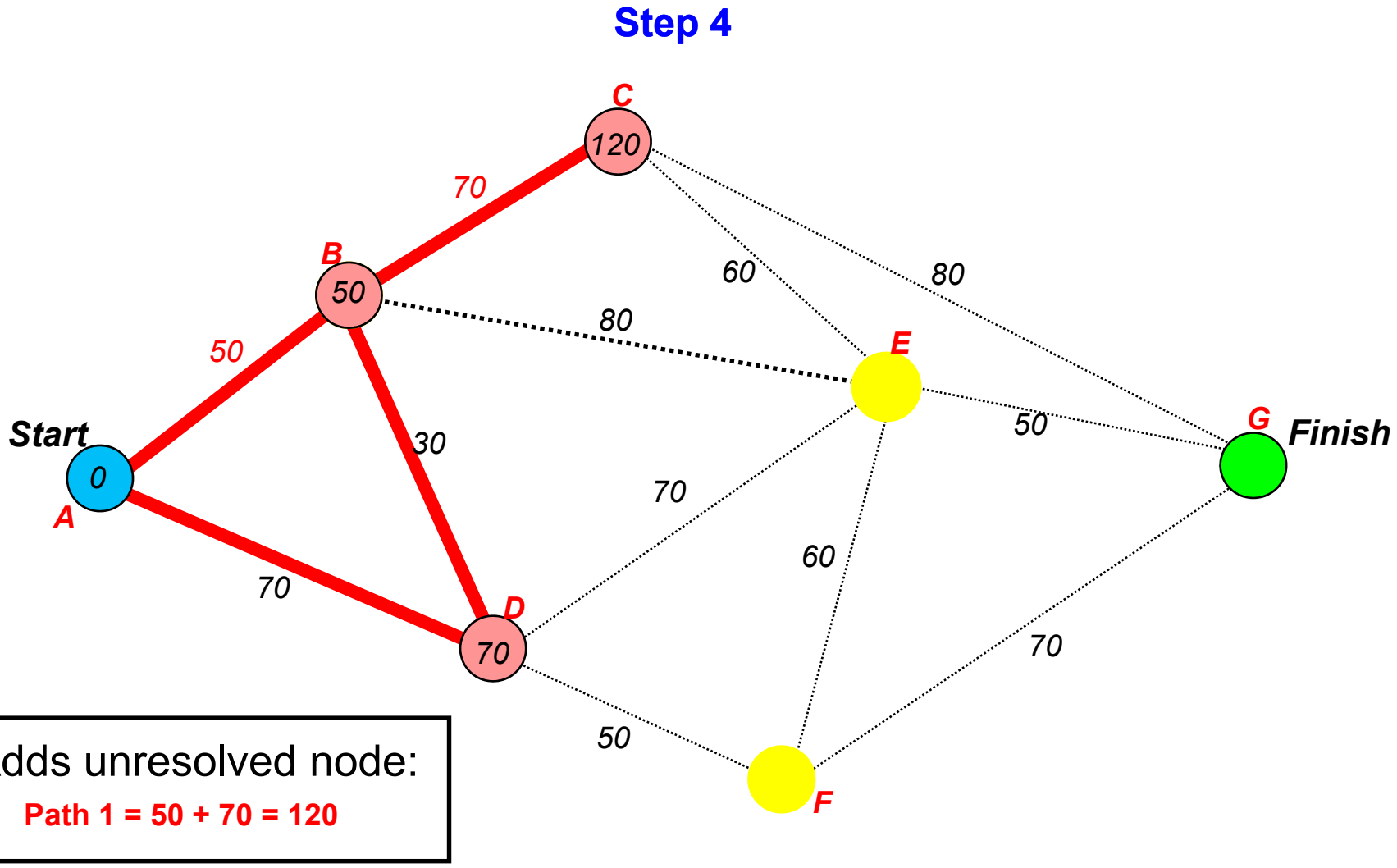


Figure 30.12:

Example of Dijkstra Algorithm

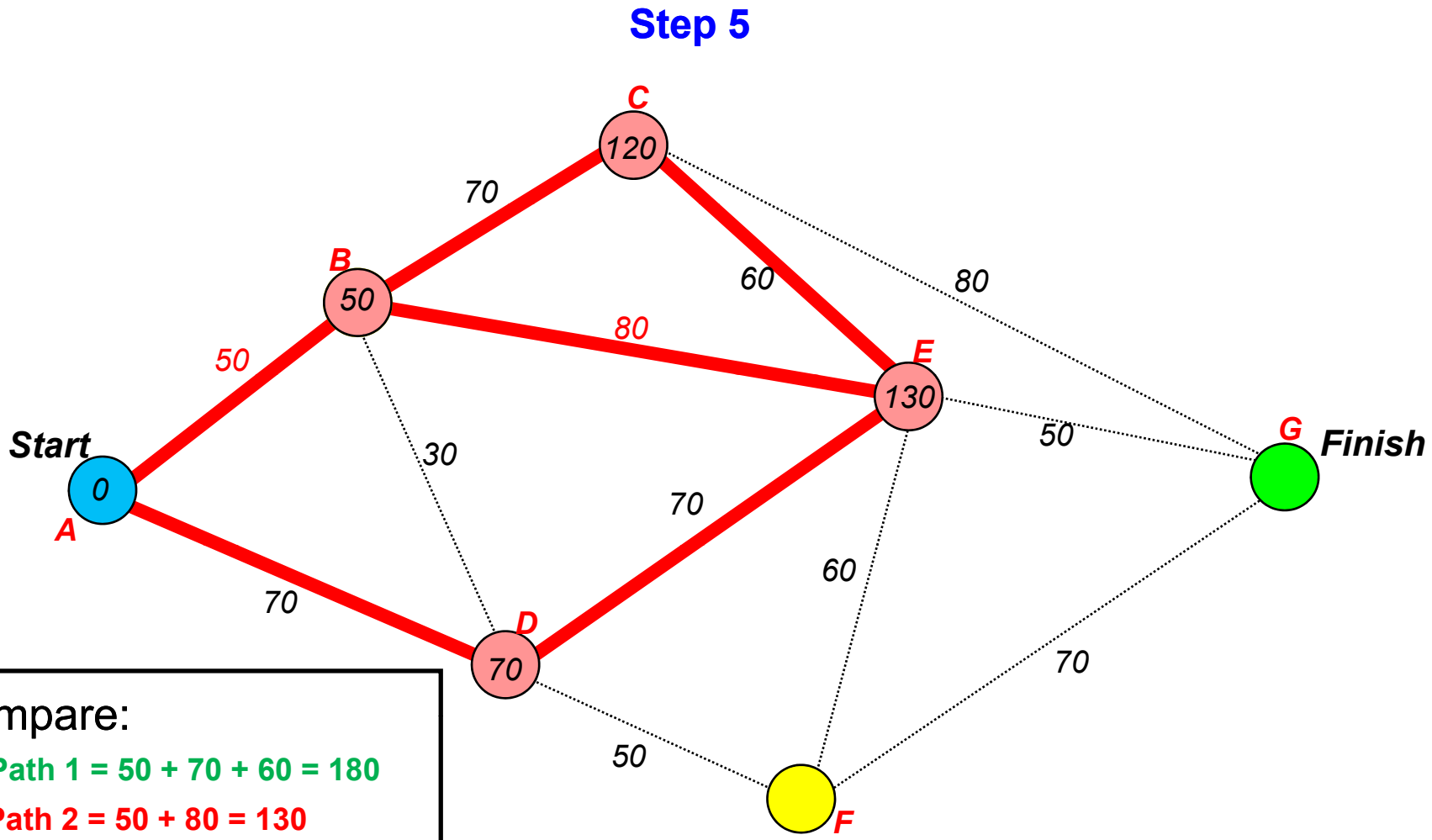


Figure 30.13:

Example of Dijkstra Algorithm

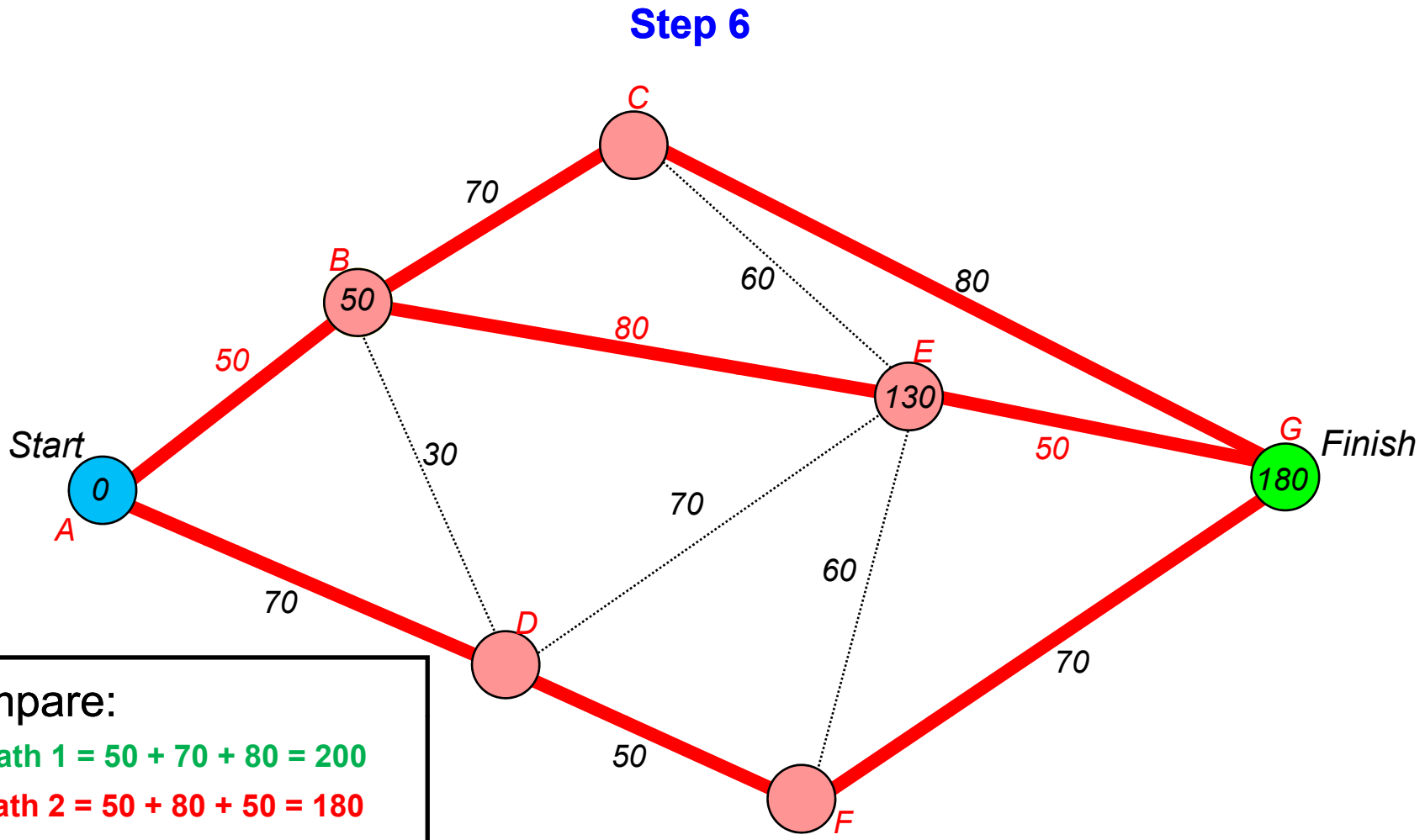
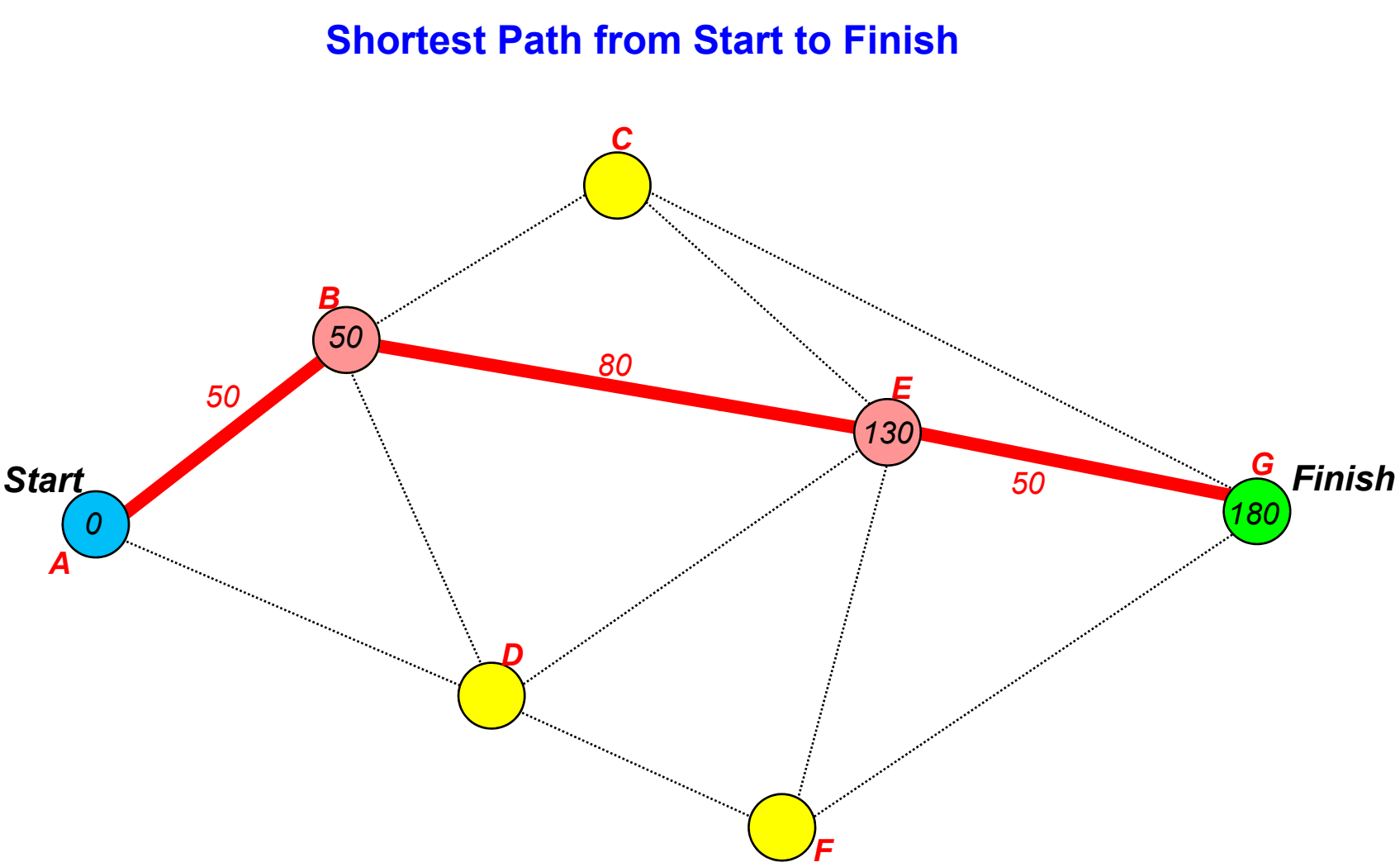


Figure 30.14:

Example of Dijkstra Algorithm



then this would produce the best solution. However, with an origin-destination matrix, we really want to know the distance between a pair of nodes (one origin and one destination). Consequently, the Dikjstra algorithm is very, very slow compared to what we need. It would be a lot quicker if we could find the distance from each origin-destination pair but quit the algorithm as soon as that distance has been determined.

This is where the A* algorithm comes in. A* was developed within the artificial intelligence research area as a means for developing a *heuristic* rule for solving a problem (Nilsson, 1980). In this case, the heuristic rule is the remaining distance from a solved node to the final destination. That is, at every step in the Dijkstra routine, an estimate is made of the remaining distance from each possible choice to the final destination. The node that is chosen for the shortest path is that which has the least total *combined* distance from the previously determined node to the final goal. Thus, for any step, if D_{i1} is the distance to a node, i, which has not already been put on the shortest path and D_{i2} is an estimate of the distance from that node to the final destination, the estimated total distance for that node is:

$$d_i = d_{i1} + d_{i2} \tag{30.1}$$

Of all the nodes that could be chosen, the node, i, which has the shortest total distance is selected next for the shortest path. There are two caveats to this statement. First, the node, i, cannot have already been selected for the shortest path; this is just re-stating the rules by which we search for nodes which have not yet been put on the shortest path list. Second, the estimate of the remaining distance to the final destination must be less than or equal to the actual distance to the final destination. In other words, the estimated distance, D_{i2}, cannot be an overestimate (Nilsson, 1980). However, the closer the estimated distance is to the real distance, the more efficient will be the search.

How then do we determine a reasonable estimate for D_{i2}? The answer is a straight line from the possible node to the final destination since the shortest distance between two points is a straight line (or, on a sphere, a Great Circle distance since the shortest distance between two points is an arc). If we simply calculate the straight-line (or straight arc if spherical distance is being used) from the node that we are exploring to the final node, then the heuristic will work.

Figure 30.15 displays the example network again. Like the Dijkstra algorithm, the routine first finds a node closest to A, which is itself. Next, it finds a node that has the least total distance from A to the final destination, G (Figure 30.16). There are two possibilities, go through B or go through D. The distance from A to B is 50 and the remaining distance from B to G is 130. Thus, the total distance through B would be 180. On the other hand, the distance from A to D is 70 and the remaining distance from D to G is 120. Thus, the total distance through D would be 190. Since 180 is smaller than 190, we choose node B.

Figure 30.15:

A* Modifies the Dijkstra Algorithm

Adding an Estimate of the Remaining Distance to the Dijkstra Distance

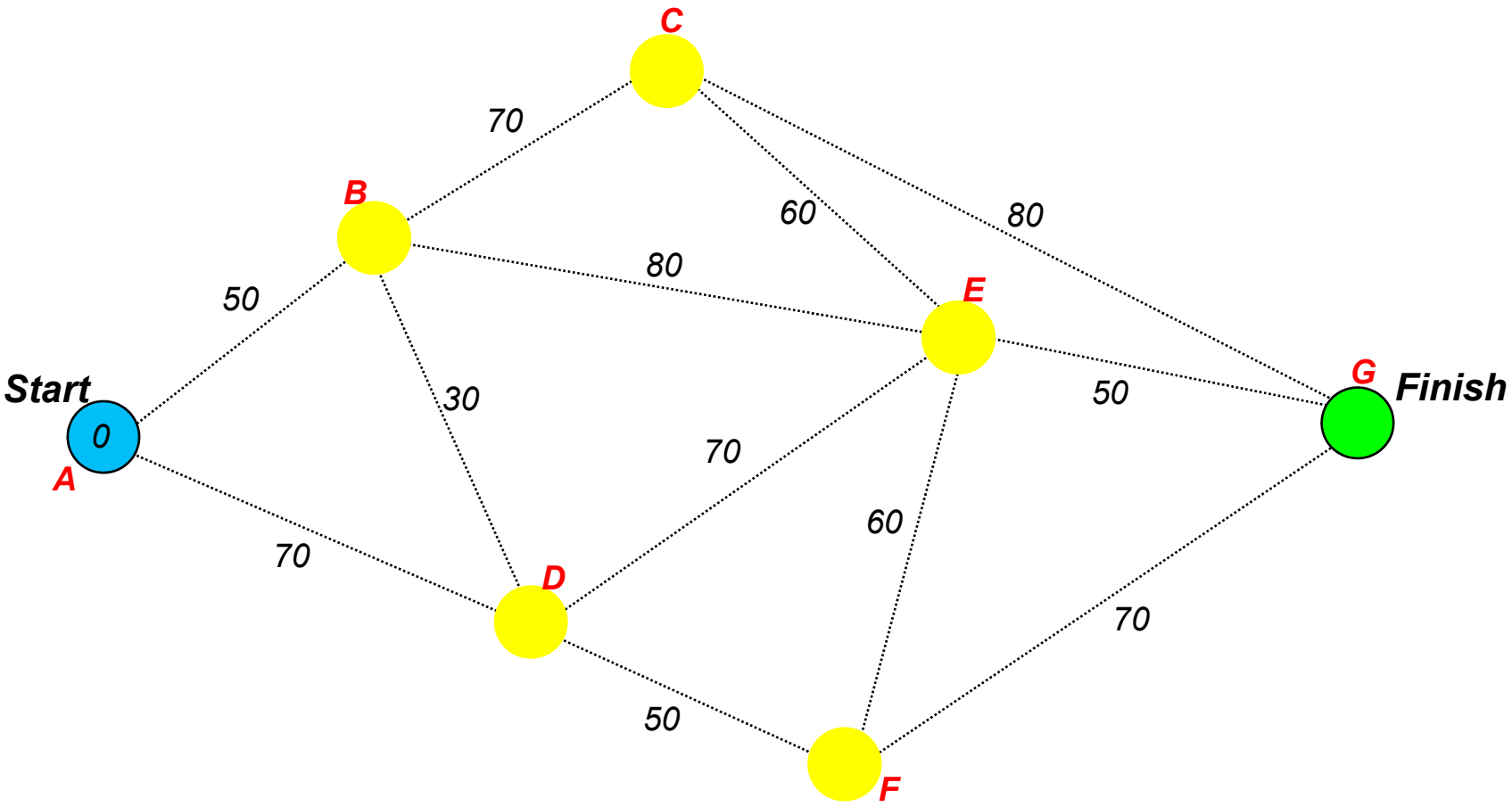
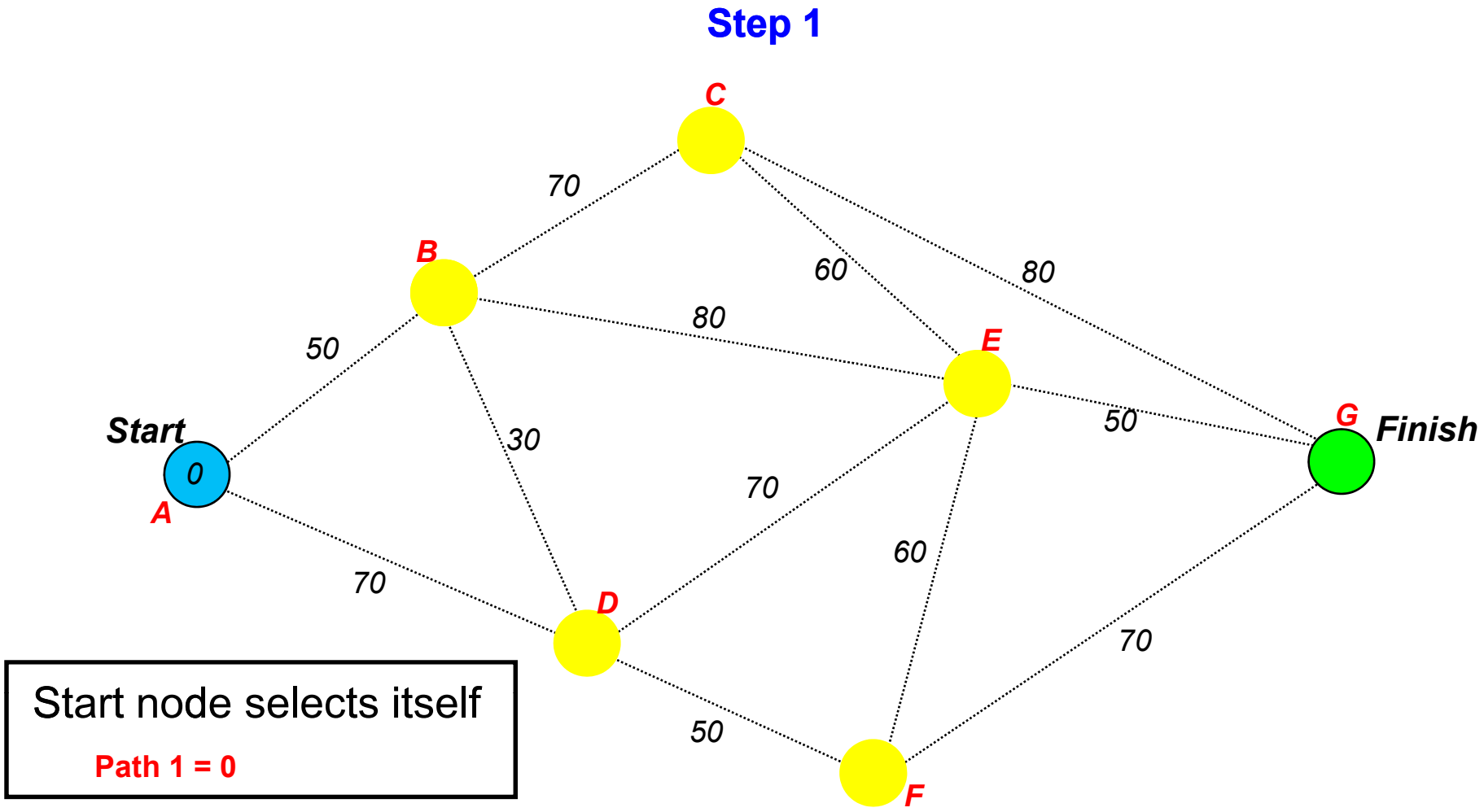


Figure 30.16:

A* Algorithm



In step 2 (Figure 30.17), three possibilities are explored for reaching G from A - through B and E; through B and C; and through D. The total distance through B and E is 180 (50 + 80 + 50) while the total distance through B and C is 200 (50 + 70 + 80) and through D is 190 (70 + 120). Thus, the routine chooses through B and E.

In step 3 (Figure 30.18), it is but a short path from E to the final destination G. The total distance through B and E to G is 180 while the total distance through B and C is 200 and through D is 190. Thus, the A* algorithm has determined a shortest path in three steps, rather than the 6 it took the Dijkstra algorithm (Figure 30.19).

In general, if V is the number of nodes in the network, the Dijkstra algorithm requires V^2 searches whereas the A* algorithm requires only V searches (Sedgewick, 2002). As can be seen, this is much more efficient than having to search every single possible node, which is what Dijkstra requires.

Applying A to multiple origins*

As with the Dijkstra algorithm, A* can be applied to multiple origins. It does it one origin-destination combination at a time. If an origin-destination matrix is represented by the origins as rows and the destinations as columns, then the A* algorithm takes each origin-destination combination and finds the shortest path. Since it does not search all possible nodes (only those in which the total distance to the destination is shortest), it cannot determine in one step the distance from an origin to all possible destinations. However, it is so quick as an algorithm that it can be applied to each cell of the origin-destination matrix and still come out much faster than a Dijkstra search.

Weighting of Segments

As mentioned above, the units of the network can be any type of impedance - distance, travel time, or cost. These can be thought of as *weights* applied to a segment. The A* algorithm does not really care what are the units of the segments as long as they are consistent and proportional to cost. The algorithm will determine the path with the shortest total cost (or total weight).

Thus, this algorithm can be applied to a trip distribution or mode split matrix of origin-destination pairs. It will determine the shortest path from each origin zone to each destination zone and can do this in the measurement units that are selected for weighting.

The advantages for travel demand modeling are enormous. It means that if the weighting variable is travel time, then the algorithm will find the shortest time path through the

Figure 30.17:

A* Algorithm

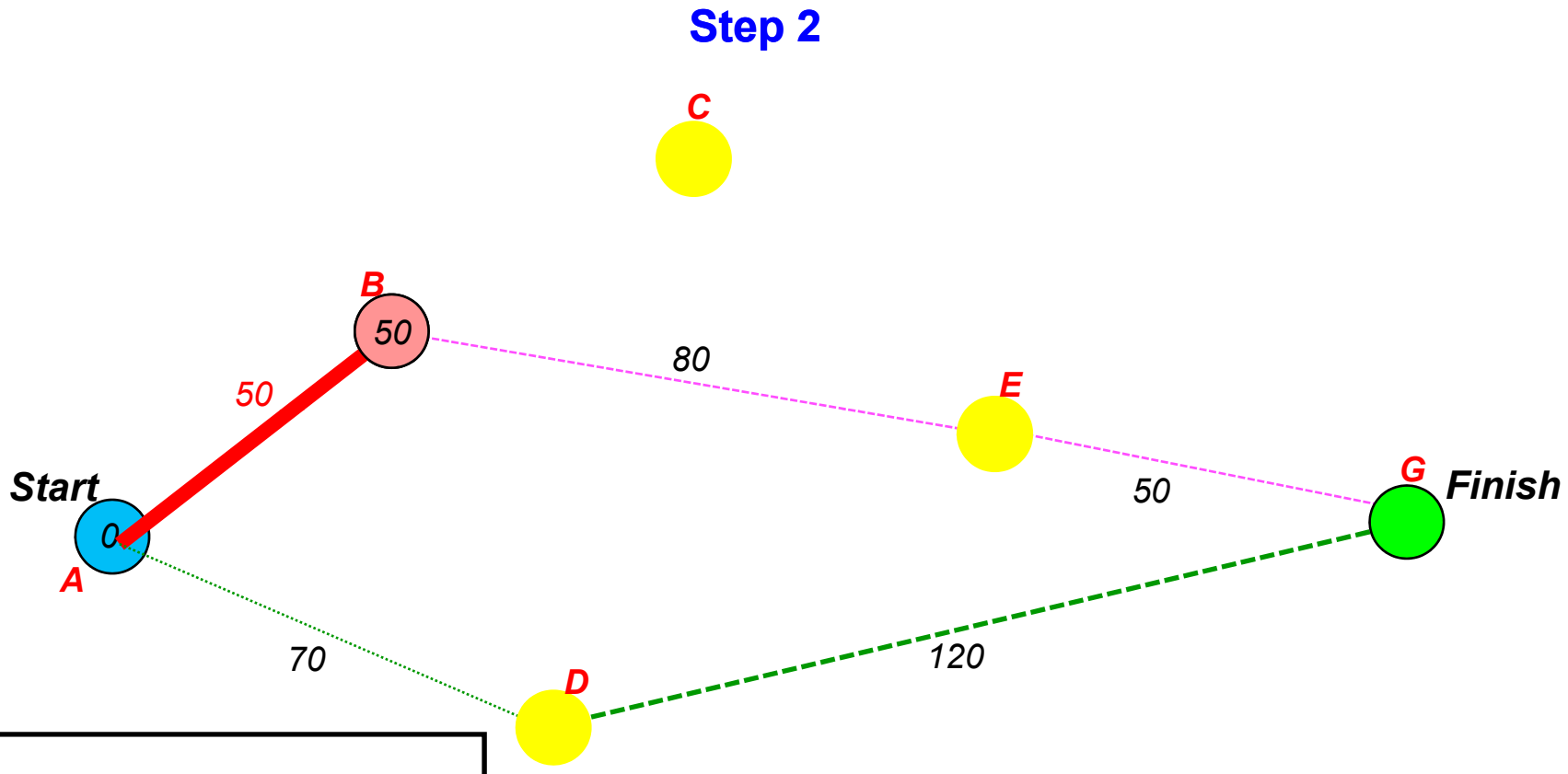
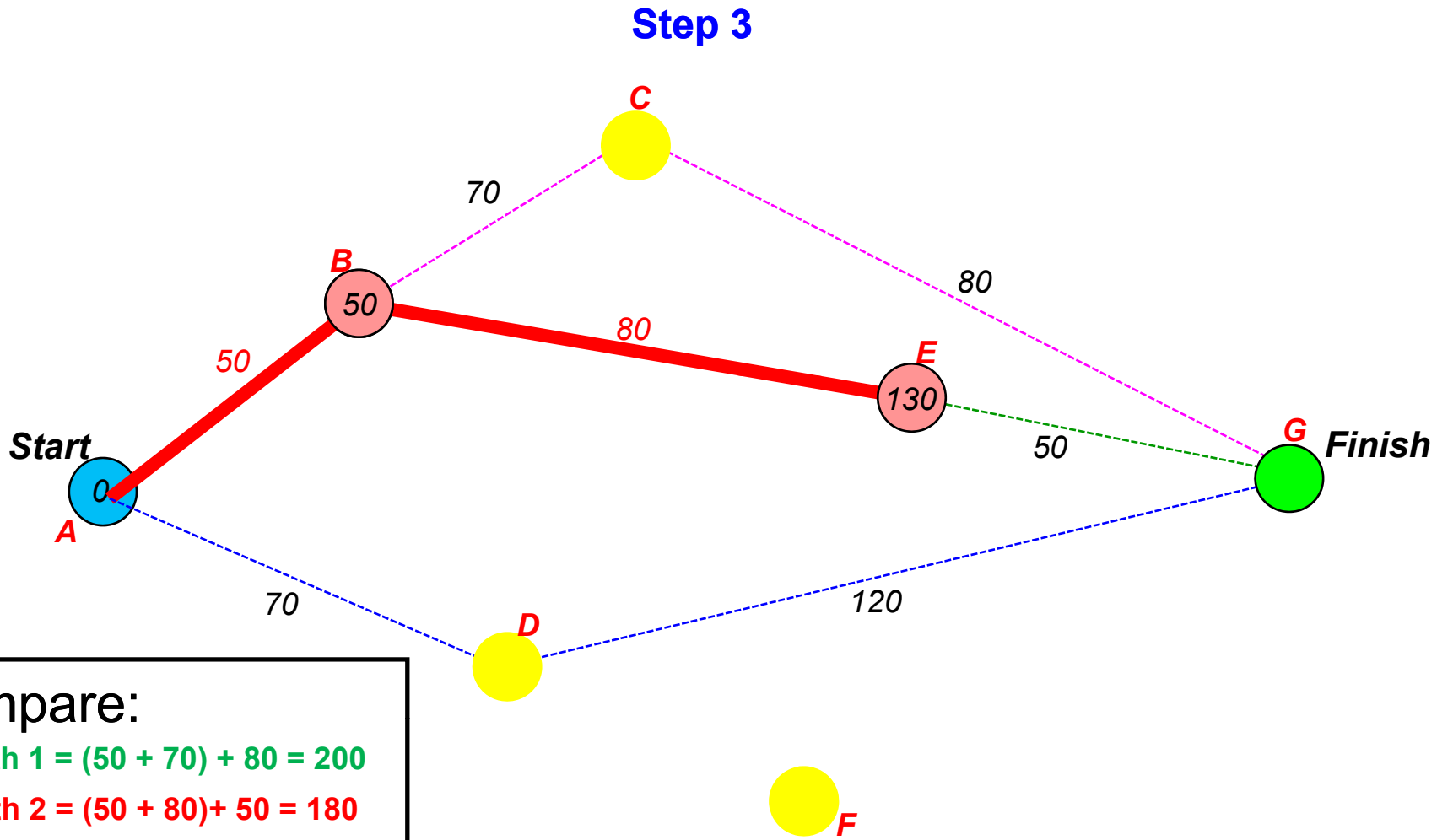


Figure 30.18:

A* Algorithm



Compare:

Path 1 = (50 + 70) + 80 = 200

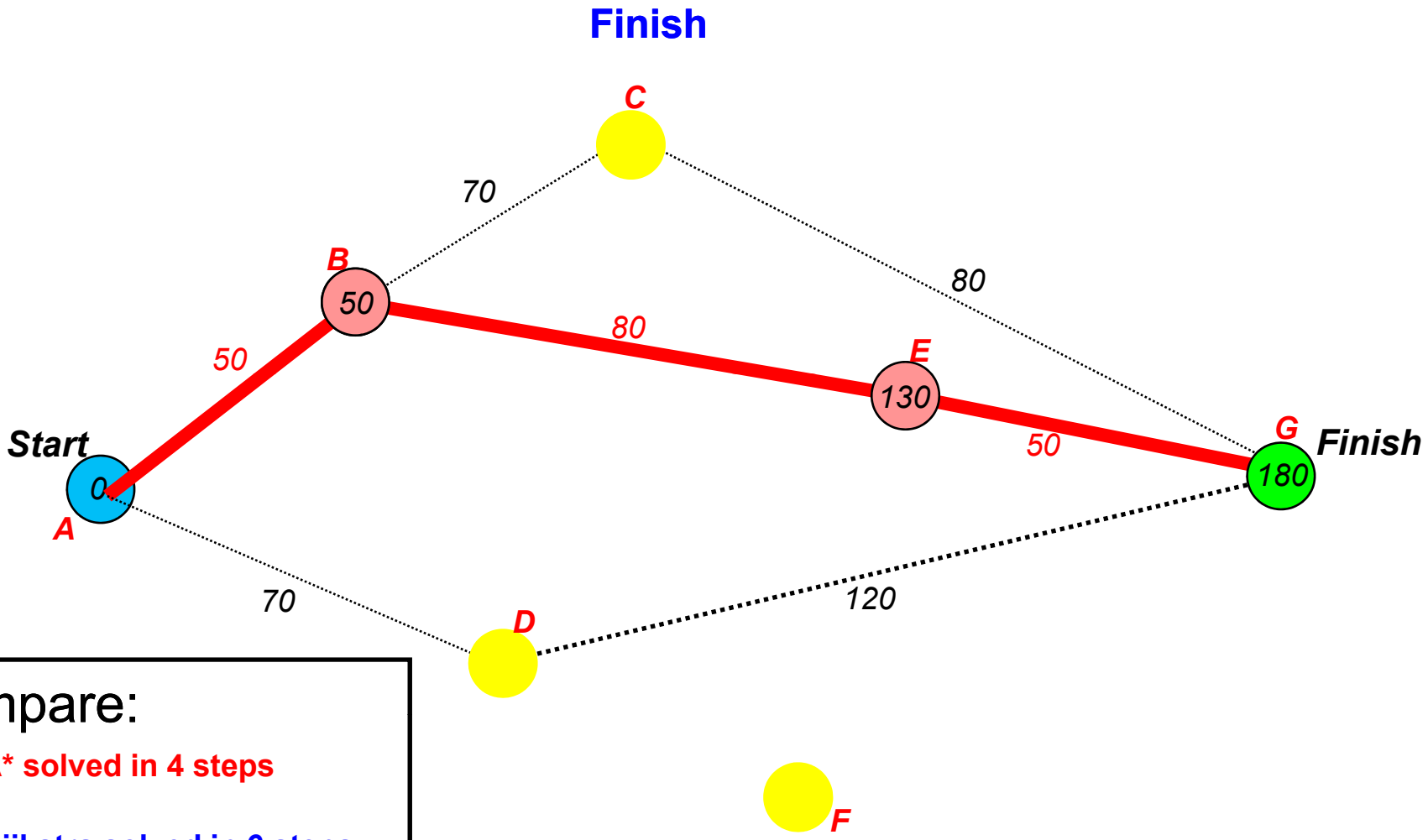
Path 2 = (50 + 80)+ 50 = 180

Path 3 = (70) + 120 = 190

Choose path 2

Figure 30.19:

A* Algorithm



Compare:

A* solved in 4 steps

Dijkstra solved in 6 steps

network for each origin-destination pair. If the weighting variable is generalized cost, then the algorithm will find the shortest cost path through the network. Finally, if the weighting variable is speed, then this must be converted into an impedance weight by dividing the distance of the segment by the speed to yield travel time. In short, the A* algorithm is an amazing one and allows the building of a routing algorithm.[2]

Routing Algorithms

In applying a shortest path analysis to a network assignment, several assumptions have to be made. As mentioned earlier, network assignment involves assigning trips to particular routes. Given a network of segments (e.g., road segments, train segments), a *routing algorithm* allocates the predicted number of trips to one or more routes. In other words, the network assignment is done through a routing algorithm. What makes this complex is that there are a number of different routing algorithms of which a shortest path is only one. Most of them are based on the assumption of travel cost relative to network capacity (Ortuzar & Willumsen, 2001).

The simplest type of routing algorithm is an *All or None* assignment. For each origin-destination pair (either for all trips or trips by specific travel mode), the algorithm calculates the shortest path through the network and assigns *all* trips to that path. This is the most rational model in that the cost of travel (whether measured by distance, travel time, or some cost measure) is minimized.

A second routing algorithm is a *stochastic path* in which each route has a certain probability of being selected. Multiple paths can be selected, but with a probability inversely proportional to their cost. The shortest path will be selected most often; the second shortest path next most often; the third shortest path third most often; and so forth. This type of algorithm attempts to capture the variability in travel behavior that can come from traveler's perceptions or incomplete information about the choice of path.

A third routing algorithm is a *congested assignment* in which there is feedback from the capacity of the network to the choice of route. In the classic case, as travel volumes increase on network segments, the capacity of the segment to absorb traffic is approached. The higher the ratio of the volume-to-capacity (V/C), the slower traffic becomes on the segment. In other words, the cost of travel increases. Eventually, if the volume keeps increasing, the speed slows so much as to eventually reduce the amount of traffic that can enter the segment (ITE, 2010). In theory, if there is so much traffic volume relative to the capacity, traffic comes to a complete

[2] For larger databases greater than, say, 1 million records, however, A* is too slow. An algorithm that is appropriate for very large databases can be found in Shekhar and Chawla (2003).

halt (gridlock). However, in practice this does not happen as drivers take other routes. Consequently, with high V/C ratios, other routes become more desirable and some traffic spills over on to those segments. his type of model is frequently used in metropolitan travel demand models for transportation since congestion is a major factor in most urban areas.

There are advantages and disadvantages to each of these approaches. The "All or none" assignment is the closest to a rational choice model; the route with the lowest total cost is chosen. On the other hand, this algorithm will continue to assign trips to a route even if the road segment becomes extremely congested, which is not realistic. A stochastic model has the advantage of accounting for variability. If individual-level data could be obtained that measured individual choices and perceptions of routes, then it is possible that a realistic proportional split among routes could be detected. More often, however, such information is lacking and a variation on the mode split model is used to proportion the trips among the different possible routes (see Ortuzar & Willumsen, 2001, Chapter 10 for more information).

The "Congested assignment" algorithm can be seen as a more realistic variation on the "All or none" in that the costs of travel change as the network capacity is reached. Most transportation models use that type of model because it is a more realistic representation.

Lack of Information about Crime Trips

The problem with crime trips, however, is that the number of trips is liable to represent only a very small proportion of the total trips on any segment of a network. Thus, there is not liable to be any feedback from the capacity limits of segments to crime trips *per se*. Any feedback is liable to apply to all trips, of which the crime trips are a sub-set. It might be possible to link a crime trip route choice algorithm to a general congested assignment in order to approximate this situation, but the amount of information that would be necessary to be obtained and the complexity of the modeling algorithm would probably not produce much tangible benefits beyond what a simple model predicts.

Further, there could be feedback from surveillance and other policing practices that might increase the cost to an offender of traveling along a particular route. However, without any detailed information about perceived costs of particular routes, it is difficult to postulate any type of model for choosing alternatives. This would be a very valuable area of research in understanding the travel behavior of offenders. At the end of this chapter, there is a brief discussion of an article that modeled the likely escape routes taken by bank robbers in Baltimore County, MD (Levine, 2007).

But, short of that information, an "All or none" assignment routine is probably the easiest to implement for allocating the predicted crime trips to routes.

The CrimeStat Network Assignment Module

The *CrimeStat* network assignment routine implements an "All or none" assignment based on the A* shortest path algorithm. Figure 30.20 shows the setup page for network assignment. On the page, there is a network assignment routine and there are some network utilities.

Network Used

The first input that needs to be made is which network is to be used. The choices are the network specified on the Measurement parameters page (the default) or an alternative network.

Network on measurement parameters page

Check the 'Network on Measurement parameters page' box to use that network. All the parameters will have been defined for that setup (see Measurement parameters page).

Alternative network

If an alternative network is to be used, it must be defined. Check the 'Alternative network' box and click on the 'Parameters' button. Figure 30.21 shows the dialogue box for the alternative network.

> Note: if a network is also used on the Measurement Parameters page, then it must be defined there as well. *CrimeStat* will check whether that file exists; if it does not, the routine will stop and an error message will be issued. Therefore, if an alternative network is used, the user should probably change the distance measurement on the Measurement Parameters page to direct or indirect distance.

Type of network

Network files can be *bi-directiona*l (e.g., a TIGER file) or *single directional* (e.g., a transportation modeling file). In a bi-directional file, travel can be in either direction. In a single directional file, travel is only in one direction. Specify the type of network to be used.

Input file

The network file can either be a shape file (line, polyline, or polylineZ file) or another file, either dBase IV 'dbf', Microsoft Excel 'xls/xlsx', Microsoft Access 'mdb', Ascii 'dat', or an

Figure 30.20:

Network Assignment Module

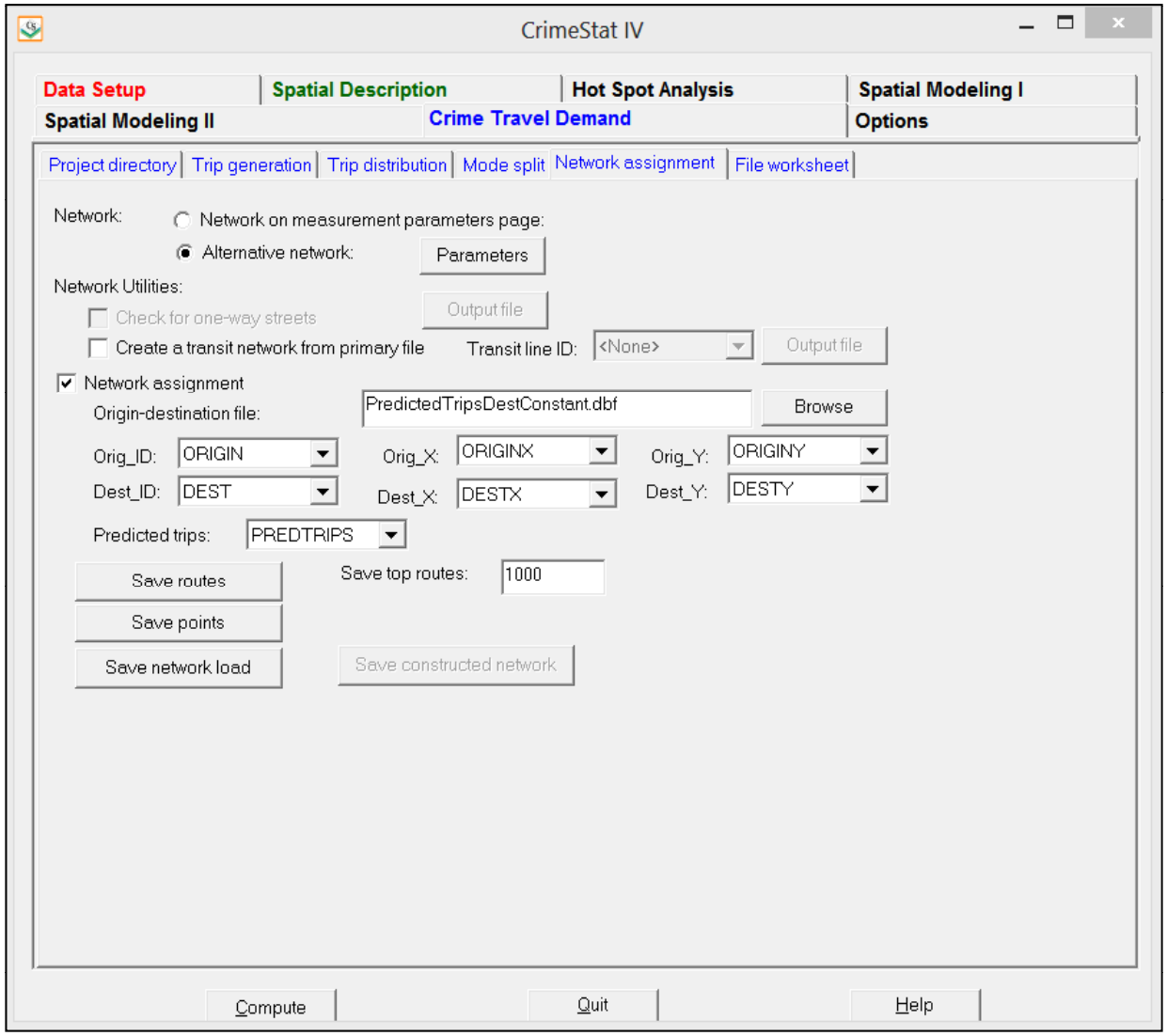
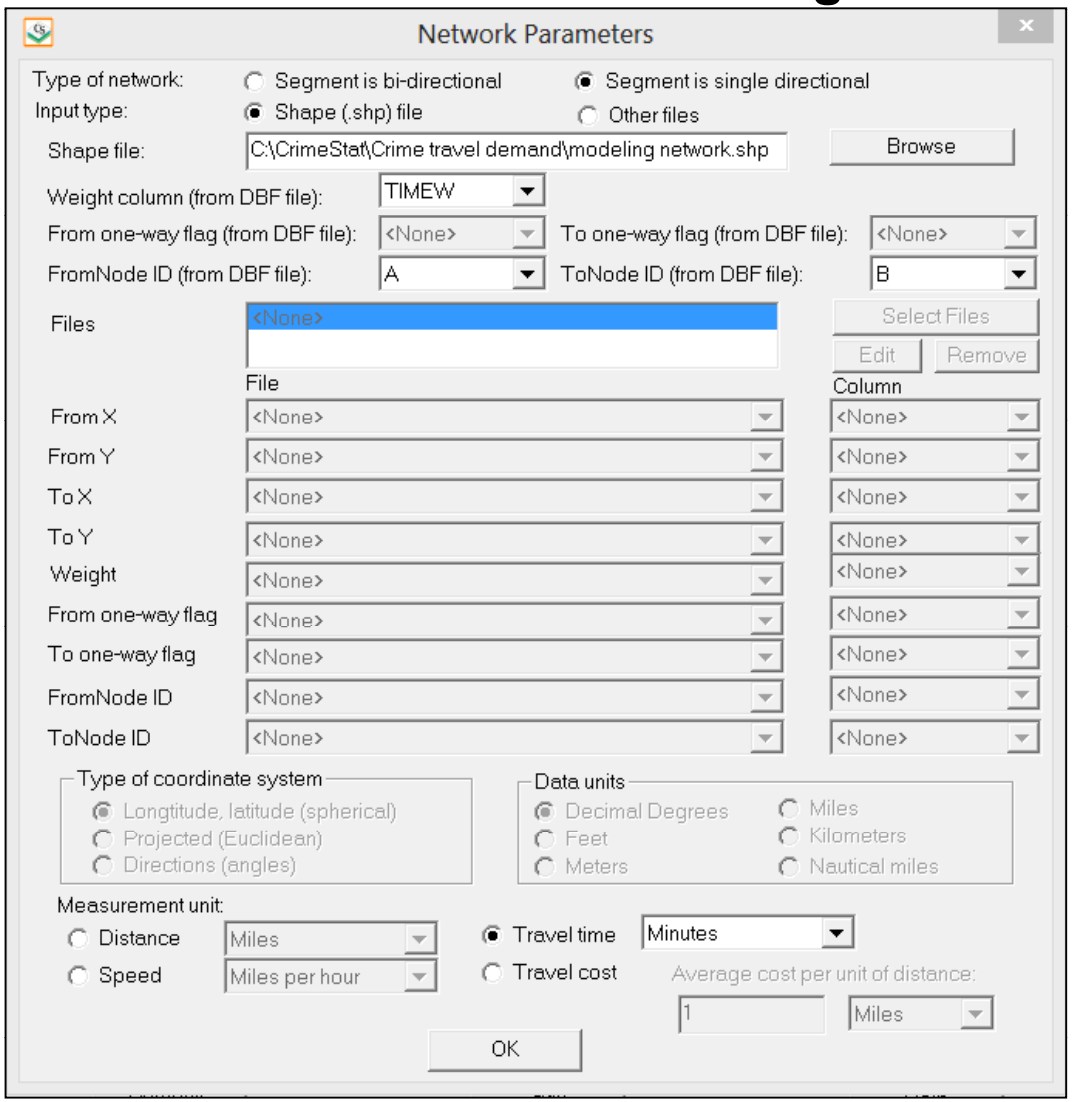


Figure 30.21:

Alternative Network Dialogue



ODBC-compliant file. The default is a shape file. If the file is a shape file, the routine will know the locations of the nodes. For a dBase IV, Excel or another file type, the X and Y coordinate variables of the end nodes must be defined. These are called the "From" node and the "End" node. An optional weight variable is allowed for both a shape or dbf file. The routine identifies nodes and segments and finds the shortest path. If there are one-way streets in a bi-directional file, the flag fields for the "From" and "To" nodes should be defined.

Weight field

By default, each segment in the network is not weighted. In this case, the routine calculates the shortest distance between two points using the distance of each segment. However, each segment can be weighted by travel time, speed or travel costs. If travel time is used for weighting the segment, the routine calculates the shortest time for any route between two points. If speed is used for weighting the segment, the routine converts this into travel time by dividing the distance by the speed. Finally, if travel cost is used for weighting the segment, the routine calculates the route with the smallest total travel cost. Specify the weighting field to be used and be sure to indicate the measurement units (distance, speed, travel time, or travel cost) at the bottom of the page. If there is no weighting field assigned, then the routine will calculate using distance.

From one-way flag and To one-way flag

One-way segments can be identified in a bi-directional file by a 'flag' field (it is not necessary in a single directional file). The 'flag' is a field for the end nodes of the segment with values of '0' and '1'. A '0' indicates that travel can pass through that node in either direction whereas a '1' indicates that travel can only pass from the other node of the same segment (i.e., travel cannot occur from another segment that is connected to the node). The default assumption is for travel to be allowed through each node (i.e., there is a '0' assumed for each node). There is a 'From one-way flag' field and a 'To one-way flag' field. For each one-way street, specify the flags for each end node. A '0' allows travel from any connecting segments whereas a '1' only allows travel from the other node of the same segment. Flag fields that are blank are assumed to allow travel to pass in either direction.

FromNode ID, ToNode ID

If the network is single directional, there are individual segments for each direction. Typically, two-way streets have two segments, one for each direction. On the other hand, one-way streets have only one segment. The FromNode ID and the ToNode ID identify from which end of the segment travel should occur. If no FromNode ID and ToNode ID is defined, the

routine will chose the first segment of a pair that it finds, whether travel is in the right or wrong direction. To identify correctly travel direction, define the FromNode and ToNode ID fields.

Type of coordinate system

The type of coordinate system for the network file is the same as for the primary file.

Measurement unit

By default, the shortest path is in terms of distance. However, each segment can be weighted by travel time, travel speed, or travel cost.

1. For travel time, the units are minutes, hours, or unspecified cost units. For speed, the units are miles per hour and kilometers per hour. In the case of speed as a weighting variable, it is automatically converted into travel time by dividing the distance of the segment by the speed, keeping units constant.

2. For travel cost, the units need to be defined as cost per unit distance (e.g., per mile, per kilometer). The routine will then indentify routes by those with the smallest total cost.

Network Utilities

There are two network utilities that can be used.

Check for one-way streets

First, there is a routine that will identify one-way streets *if* the network is single directional. In a single directional file, one-way streets do not have a reciprocal pair (i.e., a segment traveling in the opposite direction). This is indicated by a reciprocal pair of ID's for the “From” and “To” nodes. If checked, the routine identifies those segments that do not have reciprocal node ID's. The network is saved with a new field called “**Oneway**”. One-way segments are assigned a value of ‘1’ value and two-way segments are assigned a value of ‘0’. The output is saved as an *ArcGIS* '.shp', *MapInfo* '.mif' or various *Ascii* file types.

Create a transit network from primary file

Second, there is a routine that will create a transit network from the primary file. This is useful for creating a transit network from a collection of bus stops (bus network) or rail stations (rail network). If checked, the routine will read the primary file and will draw lines from one

point to another *in the order* in which the points appear in the primary file. Note, it is essential to order the points in the same order in which the network should be drawn (otherwise, an illogical network will be obtained). It is easy to do this in a spreadsheet program.

Transit Line ID

The routine can handle multiple lines, for example different rail lines or bus routes (e.g., Line A, Line B, Route 1, Route 2). In the primary file, the points must be grouped by lines, however, and must be classified by a Transit Line ID field. Within each group, the points must be arranged in order of occurrence; the routine will draw lines from one point to another in that order. In the Transit Line ID field, indicate which variable is the classification variable. The output is saved as an *ArcGIS* '.shp', *MapInfo* '.mif' or various *Ascii* file types.

Figure 30.5 above showed the effect of creating four separate rail lines from the station locations while Figure 30.6 showed the merged four lines implemented with the Group ID.

Network Output

There are three types of output for the network assignment routine. First, the most frequent inter-zonal routes (i.e., trips between different zones) can be output as polylines. Second, the most frequent intra-zonal routes (i.e., trips within the same zone) can be output as points. Third, the entire network can be output in terms of the total number of trips that occur on each segment (*network load*).

Save inter-zonal routes

The shortest routes can be saved as separate **polyline** objects for use in a GIS. Specify the output file format (*ArcGIS* '.shp', *MapInfo* '.mif' or various *Ascii* file types) and the file name.

Save top inter-zonal routes

Because the output file is very large (number of origin zones x number of destination zones), the user can select a zone-to-zone route with the most predicted trips. The default is the top 100 origin-destination combinations. Each output object is a line from the origin zone to the destination zone with a Route prefix. The prefix is placed before the output file name. The graphical output includes:

1. An ID number from 1 to K, where K is the number of links output (ID)
2. The feature prefix (ROUTE)
3. The origin zone (ORIGIN)

4. The destination zone (DEST)
5. The X coordinate for the origin zone (ORIGINX)
6. The Y coordinate for the origin zone (ORIGINY)
7. The X coordinate for the destination zone (DESTX)
8. The Y coordinate for the destination zone (DESTY)
9. The number of trips on that particular route (FREQ)
10. The distance between the origin zone and the destination zone (DIST).

Figure 30.22 shows the top 300 routes calculated with the modeling network. The assignment was weighted by travel time and the thickness and color of the line is proportional to the number of predicted trips.

To see how this differs from the trip distribution matrix, Figure 30.23 zooms into a high volume route in eastern Baltimore County. The modeling streets are displayed as are the predicted links from the trip distribution for that area. As seen, the trip distribution simply produces straight-line links between origins and destinations. In this case, the crime trips come into to the centroid of the Traffic Analysis Zone (TAZ) in the middle of this hot spot of crimes (TAZ 610). The actual routes, on the other hand, follow the streets (in this case, the modeling network) and are more circuitous. Several of the streets are used much more heavily than others, according to the assignment.

An additional point should be noted, however. Since the modeling network was used rather than the TIGER network, the trips into and from the centroid of the TAZ do not follow any particular road; the algorithm simply draws a straight line from the centroid to the nearest road segment. In subsequent modeling, it might be worthwhile to digitize additional streets in this neighborhood since there are many crimes being attracted to it. A crime mapping analyst can easily add the additional features to improve resolution. The model would have to re-run, however, to get a more accurate display.

Save intra-zonal routes

Intra-zonal routes (trips in which the origin and destination are the same zone) can be output as separate **point** objects as an *ArcGIS* '.shp', *MapInfo* '.mif' or various *Ascii* file types. Again, the top K points are output (default=100). Each output object is a point representing an intra-zonal trip with a RoutePoints. The prefix is placed before the output file name.

The graphical output for each includes:

1. An ID number from 1 to K, where K is the number of links output (ID)
2. The feature prefix (ROUTEPoints)

Figure 30.22:

Predicted Baltimore County Crime Trips: 1993-1997

Routes and LInks for Zone-to-zone Trips: All Crimes

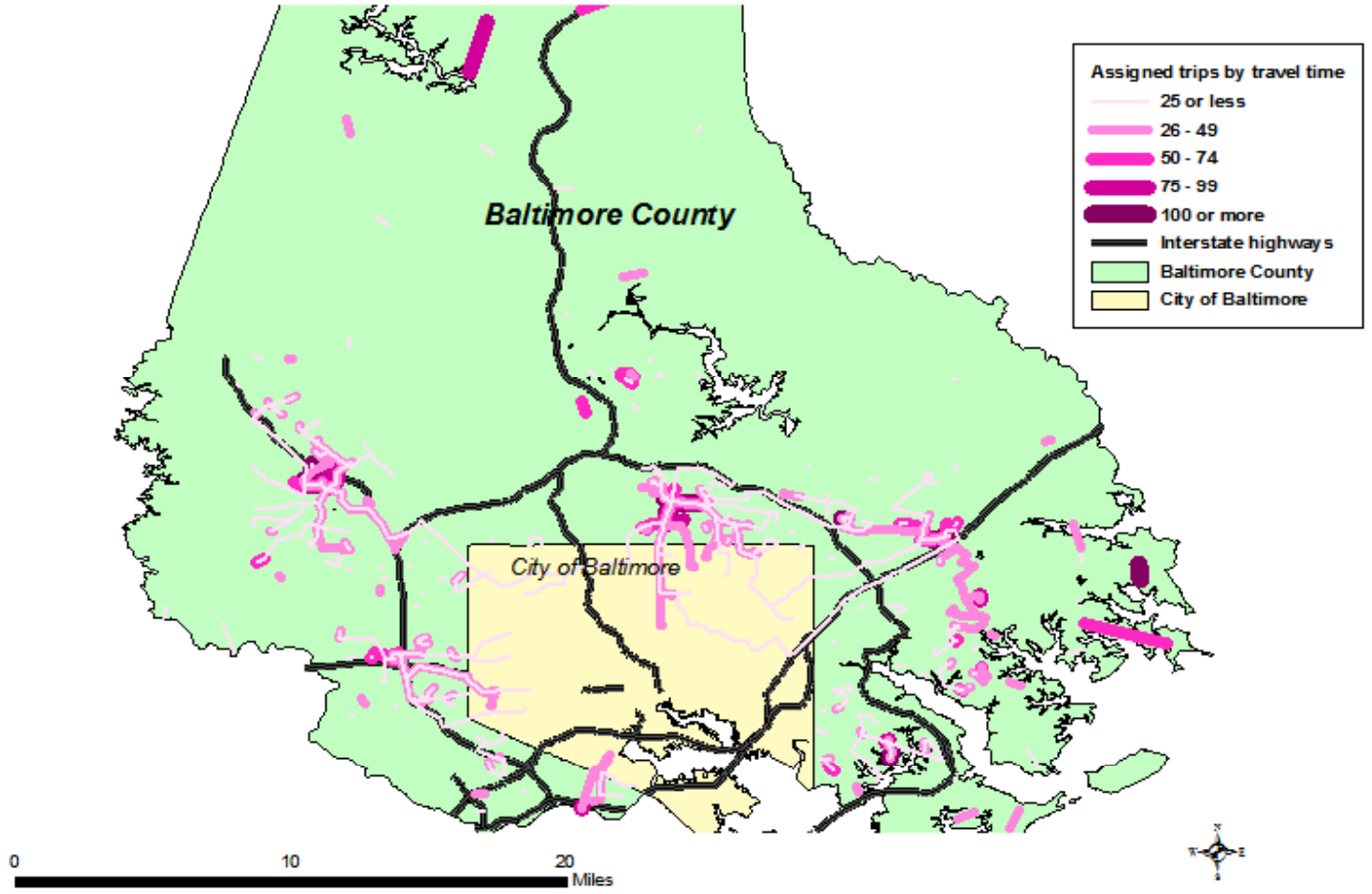
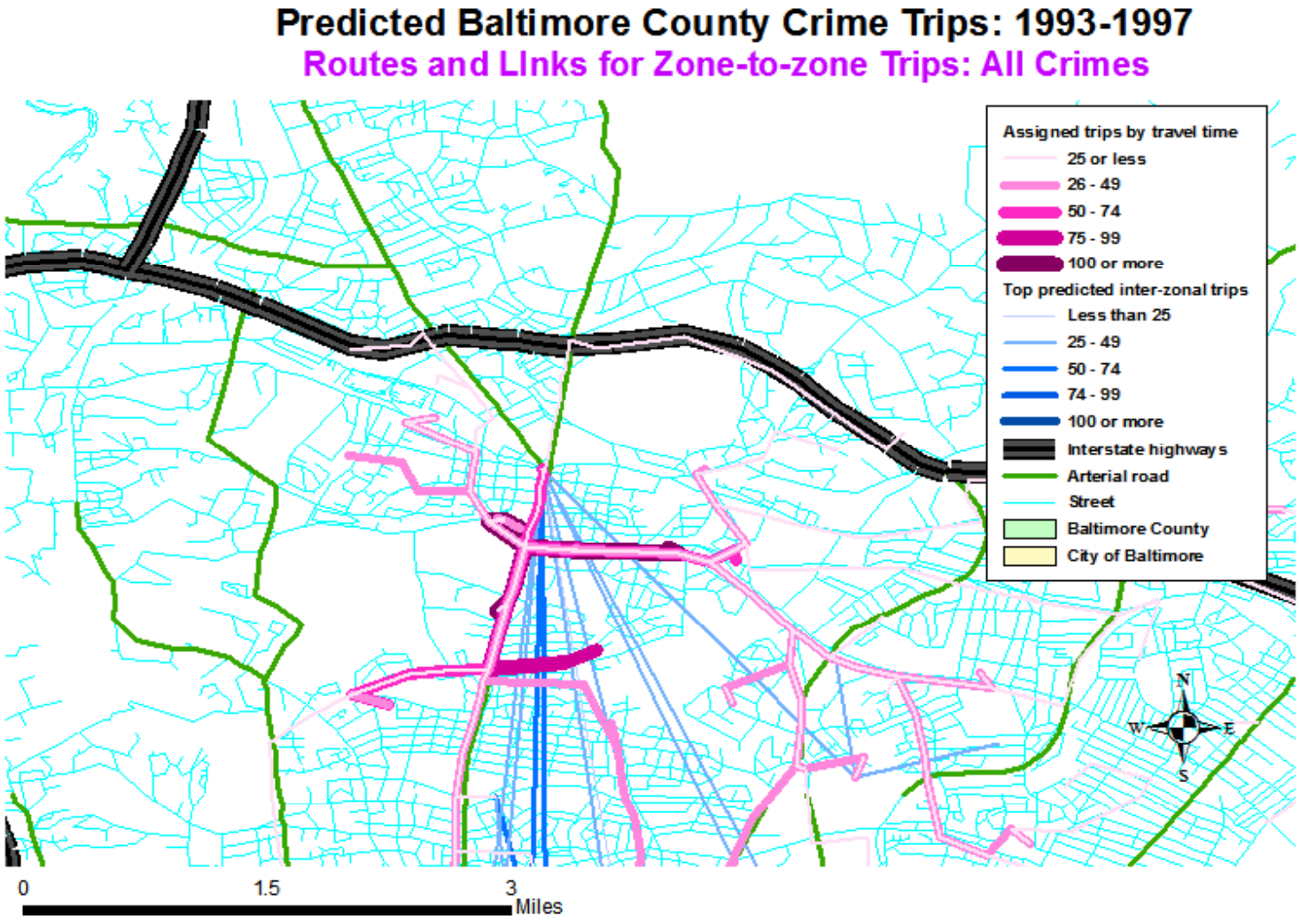


Figure 30.23:


Predicted Baltimore County Crime Trips: 1993-1997
Routes and LInks for Zone-to-zone Trips: All Crimes
Assigned trips by travel time
25 or less
26 - 49
50 - 74
75 - 99
100 or more
Top predicted inter-zonal trips
Less than 25
25 - 49
50 - 74
74 - 99
100 or more
Interstate highways
Arterial road
Street
Baltimore County
City of Baltimore
N
W
E
S
0
1.5
3
Miles

3. The origin zone (ORIGIN)
4. The destination zone (DEST)
5. The X coordinate for the origin zone (ORIGINX)
6. The Y coordinate for the origin zone (ORIGINY)
7. The X coordinate for the destination zone (DESTX)
8. The Y coordinate for the destination zone (DESTY)
9. The number of trips on that particular route (FREQ)

These are not illustrated in this chapter because they are identical to the intra-zonal output of the trip distribution module (see Chapter 28).

Save network load

It is also possible to save the total network *load* as an *ArcGIS* '.shp', *MapInfo* '.mif' or various *Ascii* file types. This is the total number of trips on each segment of the network. The routine takes every origin zone to destination zone combination and sums the number of trips that occur on each segment of the network. Click on the "Save output network" box and specify a file name for the output.

Figure 30.24 shows the entire crime trip volume on the network (network load). The assignment was weighted by travel time. Notice how there are many trips on the circular Baltimore Beltway (I-695). Because the road is a freeway, travel is generally much faster than on most arterial roads. Consequently, there are many crime trips being assigned to the freeway even though it is longer than many direct links.

To see how this differs from a shortest distance assignment, the routine was re-run using only distance as the weighting variable. Figure 30.25 displays the results. As seen, the routine does not use the Beltway very much, but instead uses the arterial roads more, particularly the diagonal arterial roads coming out of the City of Baltimore. Since the routine was determining the shortest path on the basis of distance only, it will inevitably find the most direct routes in terms of distance. In terms of travel time, however, many of those routes will be much slower because of traffic lights, cross-traffic, drivers pulling in and out of parking spaces, and so forth. Thus, the freeway is almost always quicker for travel than an arterial road except at peak rush hour conditions. This points out the importance of using travel time and, better yet, travel cost as an impedance variable. Distance is much too simple an indicator of it.

The network load routine can even be used for specific travel modes (and usually is for transportation travel demand modeling). Figure 30.26, for example, shows the network volumes (load) of bus crime trips, again weighted by travel time. According to the model, many of these trips originate in the City of Baltimore. But at the high crime locations, multiple

Figure 30.24:

Crime Volume by Road Segment Weighted by Travel Time

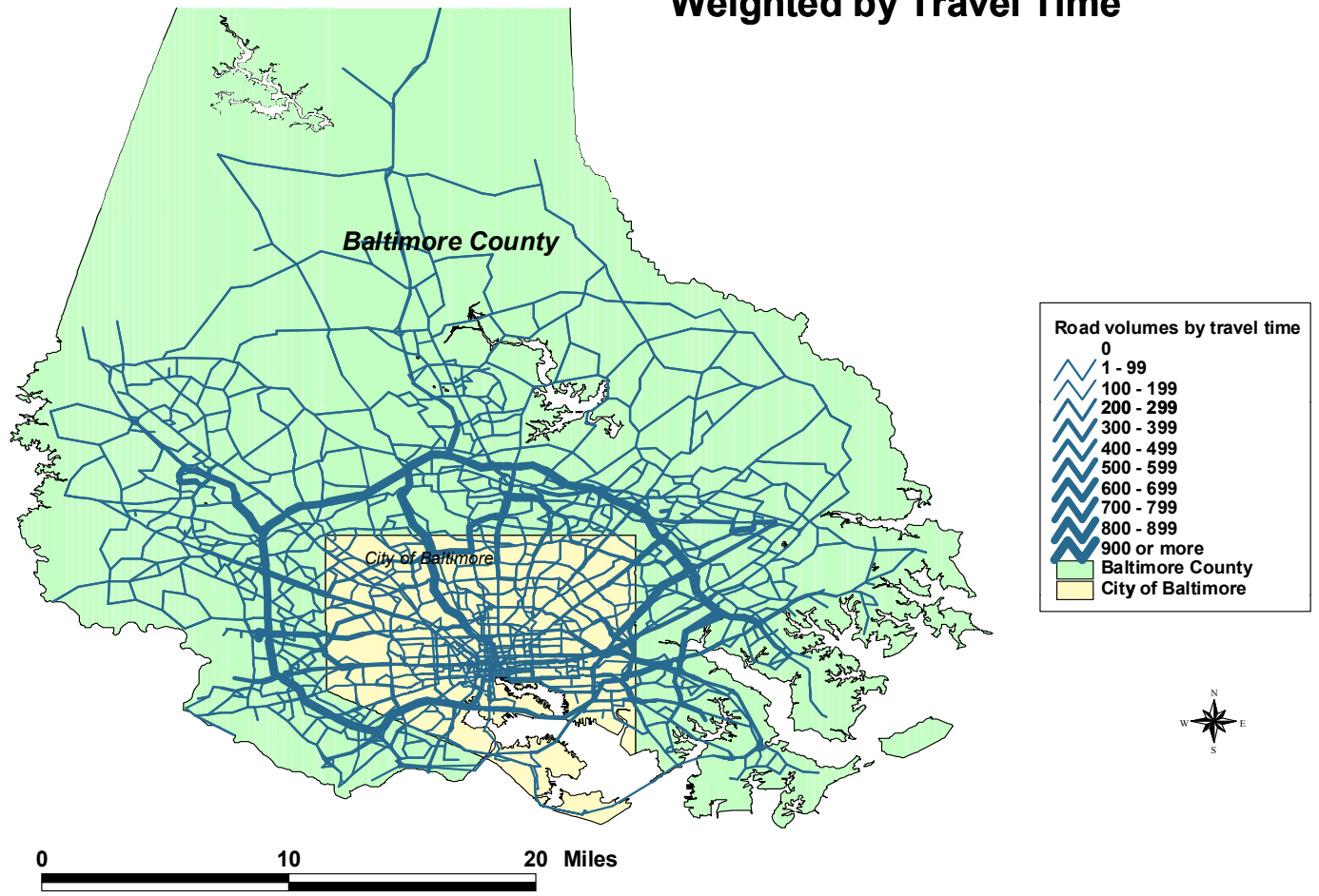


Figure 30.25:

Crime Volume by Road Segment Weighted by Distance

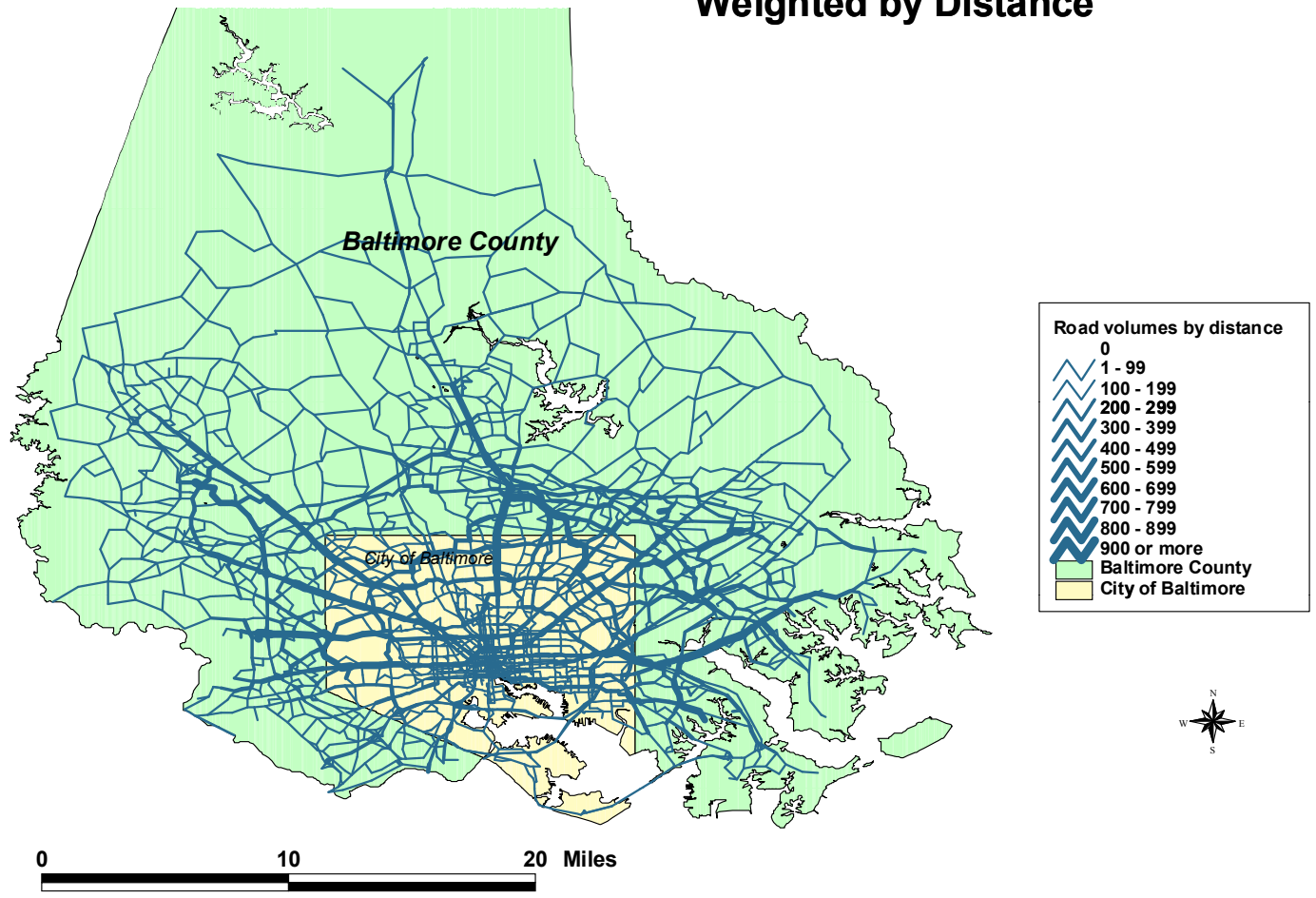
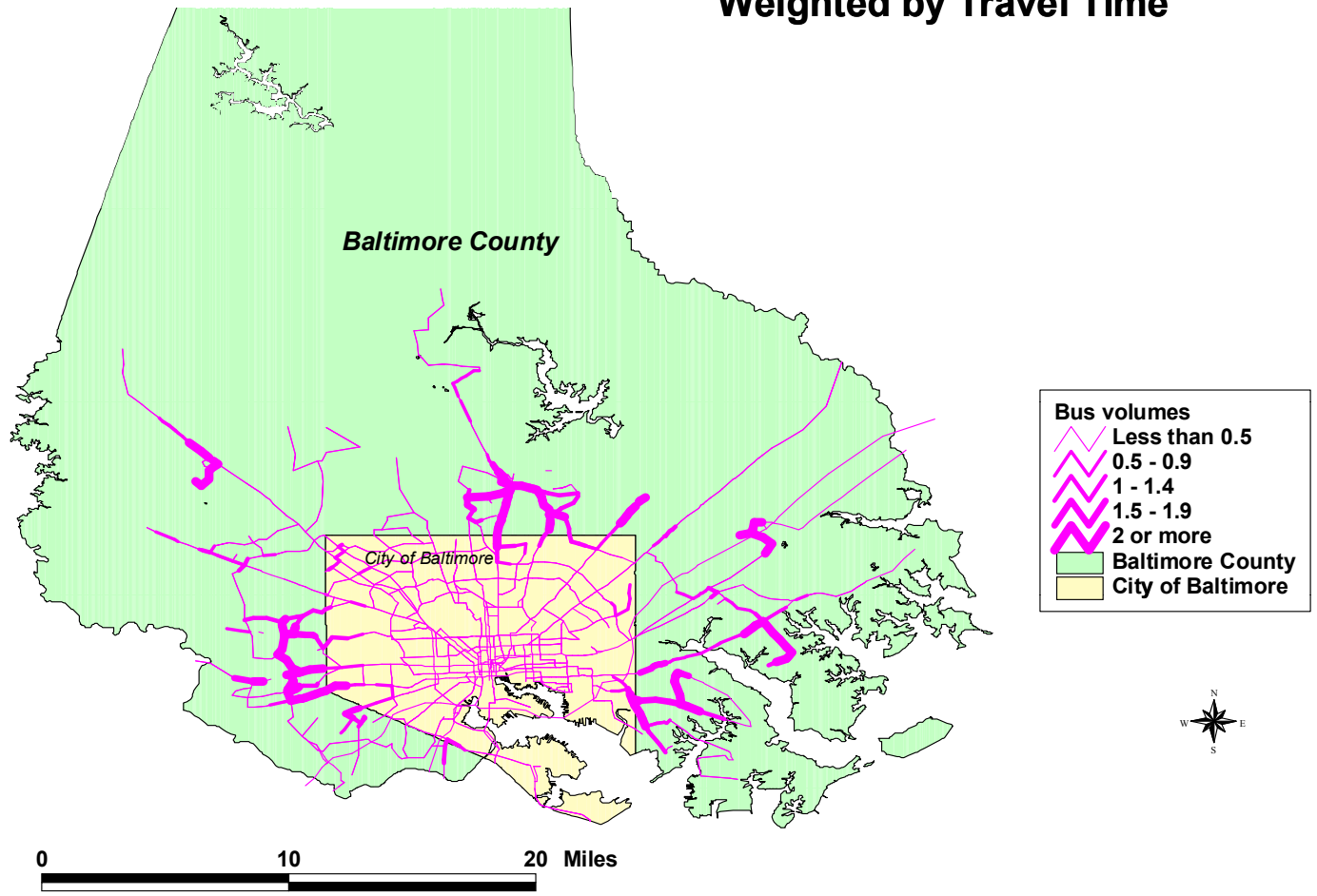


Figure 30.26:

Crime Volume by Bus Route Segment Weighted by Travel Time



bus routes tend to converge producing a high bus trip volume on the adjacent streets. Because of the very small number of bus crime trips predicted by the mode split model, the volumes are not high, even for the highest volume links. Also, notice how the Beltway is not used very much for bus trips, compared to the total network load in Figure 30.24. The reason is that most bus routes do not use the freeway but stay on arterial roads (express buses would be an exception, but those tend to be used primarily for commuting).

Figure 30.27 shows the network volumes of train trips. Since there was no data on travel times along each train segment, the volumes are weighted only by distance. The number of crime trips by train, of course, are limited as was noted in Chapter 29. Also, notice how most of the crime trips taken by train occur on two lines, the Metro line to the west and the MarcP line to the east. In both cases, the train trips start in the City of Baltimore and travel to Baltimore County. These, of course, are predictions of crime travel volumes on the rail network, not empirical verifications.

Modeling Network Assignment of Crime Types

The network assignment routine can be applied to specific crime types. In general, it is a good idea to calibrate a general assignment for all crimes before analyzing specific crimes. The reason is that there are volume dimensions that assign most crime trips to the same segments. Still, some differences can be observed. Figure 30.28 shows the likely routes for vehicle thefts (in blue) and compares it to the likely routes for all crimes (in red). There are similarities and differences. There is overlap in the predicted routes in the southeast and southwest edges of the County with the City of Baltimore, and there is some overlap at the northwest border with the City of Baltimore. At the same time, though, some differences are visible, particularly at the western border with the City of Baltimore.

In other words, the network assignment model shows different routes for vehicle thefts than for crimes in general. This difference, of course, represents differences in the trip distribution matrix of the vehicle thefts compared to all crimes.[3]

Uses of Network Assignment of Crime

A network assignment routine is the culmination of the crime travel demand modeling process. Essentially, it assigns predicted trips (whether for entire origin-destination trip pairs or for mode-specific trip pairs) to an actual network and usually on the basis of least cost. The

[3] The differences could be due to the mode split routine as well as the trip distribution matrix. However, in the case of vehicle thefts, the travel mode is not very relevant since the return trip is always by vehicle - the stolen vehicle, at least to the disposal location.

Figure 30.27:

Crime Volume by Rail Segment Weighted by Distance

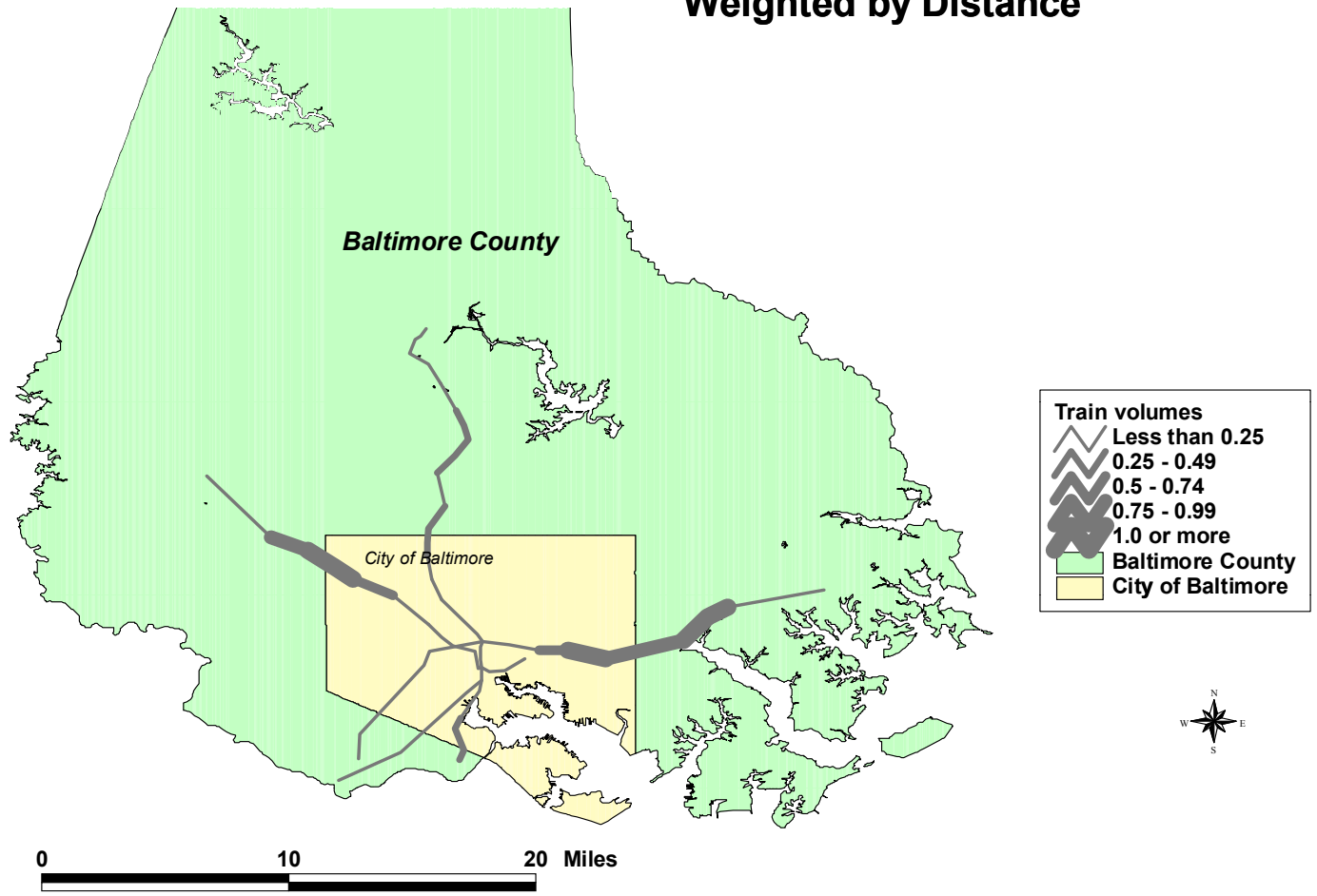
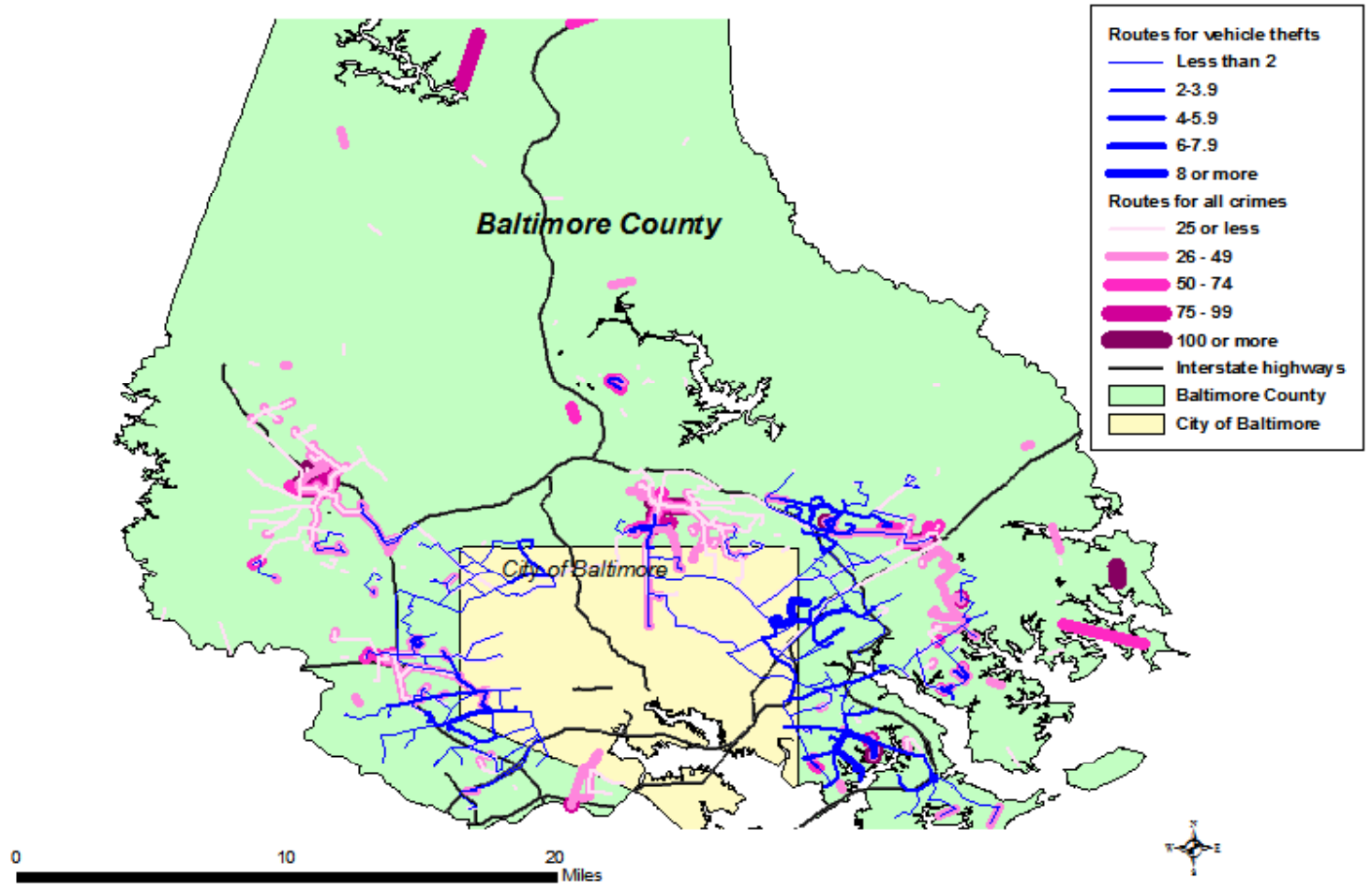


Figure 30.28:

Baltimore County Network Assignment for Crime Trips: 1993-1997

Routes for Zone-to-zone Trips: All Crimes and Vehicle Thefts



algorithm used in the *CrimeStat* network assignment routine calculated the shortest path (in terms of distance, travel time, or cost) and assigned all the trips for each origin-destination pair to this route. The representation is more complex than a simple trip link (which is a straight line) since it uses information on the actual network used. The result is a prediction of routes that are taken to commit crimes and a prediction of the total crime trip volume on each network segment. This is clearly an advance on the geographic profiling/journey-to-crime approach, which has simply analyzed travel distance as an explanatory variable.

Network assignment also has many uses for police. First, it can point out where police need to focus their deployment. In this sense, the progression of the four modeling stages represents adding information to the knowledge of the crime events. Simply mapping the crime events tells a police department where the crimes are occurring. Analyzing the trip distribution tells the department from where the crimes might be originating.

Splitting the distribution by travel model provides information about the likely travel mode used. Finally, assigning the predicted trips to actual routes gives information about how offenders may have traveled to the crime location. The model provides a lot more information than a simple description of a high crime area.

Second, knowing the likely routes of offenders can allow for increased surveillance' and target hardening. Not only can police patrol the likely routes in a more focused manner, but other surveillance tools can be used, too. For example, surveillance cameras that monitor traffic can be used for a variety of purposes. In the U.S., they have tended to be used for monitoring traffic signals for red-light running (IIHS, 2012). However, in Europe they are widely used for a variety of traffic monitoring purposes - speed enforcement, bus lane enforcement, entering the London congestion zone, as well as monitoring traffic signals. In London, for example, the entire monitoring process is automated. For a vehicle making a violation, the camera takes a picture and a software package identifies the license plate. The license number is then matched against a database of vehicles and a traffic citation is sent to the owner. There is no reason why this type of technology could not be structured to also look for stolen vehicles or vehicles belonging to individuals for which outstanding citations have been issued. In short, knowing on which roads high crime trips volumes are likely to occur can help police focus a range of surveillance tools on those locations.

Conclusion

In short, network assignment is a logical step in the modeling of crime trips and one that brings the trips down to actual routes that are used. It is a more realistic representation of travel behavior and one that can allow focused deployment by police.

References

Dijkstra, E. W. (1959). A note on two problems in connection with graphs, *Numerische Mathematik*, 1, 269-271.

IIHS (2012). *Q&A: Red Light Cameras*. Insurance Institute for Highway Safety: Arlington, VA. http://www.iihs.org/research/qanda/rlr.html. Accessed June 5, 2012.

ITE (2010). *Highway Capacity Manual* (5th edition) Institute of Transportation Engineers: Washington, DC. http://www.ite.org/emodules/scriptcontent/orders/ProductDetail.cfm?pc=LP-674. Accessed June 5, 2012.

Levine, N. (2007). Crime travel demand and bank robberies: Using CrimeStat III to model bank robbery trips. *Social Science Computer Review*, 25(2), 239-258.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc.: Los Altos, CA.

Ortuzar, J. D. & Willumsen, L. G. (2001). *Modeling Transport* (3rd edition). J. Wiley & Sons: New York.

Rabin, S. (2000a). A* aesthetic optimizations. In DeLoura, Mark. *Game Programming Gems*. Charles River Media, Inc.: Rockland, MA., 264-271.

Rabin, S. (2000b). A* speed optimizations. In DeLoura, Mark. *Game Programming Gems*. Charles River Media, Inc.: Rockland, MA., 272-287.

Sedgewick, R. (2002). *Algorithms in C++: Part 5 Graph Algorithms* (3rd edition). Addison-Wesley: Boston.

Shekhar, S. & Chawla, S. (2003). *Spatial Databases: A Tour*. Prentice-Hall: Upper Saddle River, NJ.

Stout, B. (2000). The basics of A* for path planning. In DeLoura, Mark. *Game Programming Gems*. Charles River Media, Inc.: Rockland, MA., 254-263.

U.S. Census Bureau (2011). *Tiger Products*. U.S. Census Bureau: Washington, DC. http://www.census.gov/geo/www/tiger/. Accessed May 8, 2012.

Modeling Bank Robbery Trips in Baltimore County, MD

Ned Levine
Ned Levine & Associates
Houston, TX

A study was conducted of 258 bank robberies that occurred in Baltimore County, MD, from 1993 to 1997. The crime travel demand model showed that the bank robbery trips tended to originate in poorer, denser neighborhoods and, in general, rob banks that were close to the offender's residence. Possible travel routes to the banks were modeled as well as escape routes on the assumption that the impedance of using the same routes would be higher after the robberies than before. The alternative routes can provide insights to the police for surveillance after bank robberies have occurred. The full study can be found at Levine, N. (2007). Crime travel demand and bank robberies: Using CrimeStat III to model bank robbery trips. *Social Science Computer Review*, 25(2), 239-258.