Basically a road network has to be maintained which will be represented by a graph of nodes (representing junctions) and edges (representing street segments between junctions).

The graph has to be updated regularly to reflect the availability of car2go vehicles. This includes, for each car2go vehicle, inserting an edge (call it boarding edge) from the geographically closest node in the graph to a new artificial node representing the car2go vehicle. The cost of the boarding edge could be set to the buffer time required usually to open the car2go. Additionally another edge is required, with cost zero, from the node representing the car2go vehicle to the same closest node as shown in the following figure:

car2go

0

0

1

1

1

1

1000 walk

100 driving

Open questions:

1. How efficient and scalable will be the graph update considering the dynamicity of car2go vehicles
2. Could happen that car2go vehicles are connected to the wrong node. E.g. If car2go is being parked in a park where the closest street node somewhere behind the park and not directly reachable.
3. Algorithmic theoretical questions. More specifically how to switch travel mode (e.g. from walk to drive) on the same graph and keep Dijkstra logic correct. An example problem is the following. Assume that the source is node A and the destination is point T and we want to use walk and car2go. Dijkstra expands a walk from A to both B and C and assigns a cost of 1 for both of them. Then Dijkstra settles (expands) nodes B and C (at this time the optimal least cost for B and C is known). From node B we reach the car2go node which triggers the mode change to driving instead of walking. Then dijkstra reaches node D with cost 2. Now an interesting problem happens. Dijkstra will settle node D and reach node C with mode driving and with cost 3. However, node C is already reached with cost 1 through A and Dijkstra will not accept to change the predecessor of C from A to D as reaching C through A is cheaper. But not accepting node D as a predecessor to node C means that we cannot take the edge from C to T with driving mode as we cannot get a car2go in the path A->C . So in case of not taking a car2go we will reach T with cost 1002 which is not optimal. A naïve solution to this problem could be by duplicating the graph. One copy for driving and one copy for walking with glue edges between the copies of the same node.

**Proposed solution to point 3**

While duplicating the street graph is the easiest solution to the mentioned problem, it results in high memory consumption. A better solution would be an implicit duplication of the graph. Implicit duplication of the graph means that the routing algorithm has to be modified to behave as if the graph is duplicated. In this case, only the nodes that are touched by the algorithm are temporarily duplicated if required. Following is a more detailed description of the proposed solution for the scenario walk-car2go-walk (the same applies for bike sharing).

Assume that we have *n* nodes, from *0* to *n-1*, in our graph including the car2go representative nodes. The routing algorithm starts in walking mode from the source. A driving mode is activated only after a car2go representative node is settled (removed from the priority queue). When a car2go node is settled, its outgoing edges have to be relaxed. In our case, a car2go node has only one outgoing edge leading to the node *x* from which the car2go node has been reached. Given that the car2go node can be reached only through node *x* and given that edges costs are non-negative then reaching node *x* through the car2go node can never improve its cost and hence mode driving cannot be started. To solve this problem, a duplication of node *x* for mode driving is required. This is done by adding a new node to the priority queue with id *x* + *n*. Now the new node *x* + *n* is reachable through the car2go node.

Whenever a node greater than or equal to *n* is settled, a duplication of the nodes reachable in driving mode from that node is done as previously explained. In this way, we are separating the nodes space into two spaces: from *0* to *n* for walking mode and greater than or equal *n* for driving mode. In the worst case we will have *n* duplicate nodes from *n* to *2n-1*.

As the original given graph is defined only for nodes less than *n*, a modulo *n* operation is used to get the outgoing edges for settled nodes during the run of the algorithm. This means node *x* and its duplicate *x+n* will always relax the same set of edges defined in the original graph (x mod n and x+n mod n will lead to the edges of node *x*).

Following the idea will be explained using an example graph in Fig.1 to find a path from node 0 to node 2. Walking speed is 4kmh and driving speed is 60kmh.

car2go

d,0km

w,0km

w,d,60km

w,d,4km

**Fig. 1:** sample graph with n=4 nodes. Edge label w means walk is allowed and d means drive is allowed. Edges are also labeled by distance. Node 3 represents a car2go vehicle.

The algorithm starts by settling node 0 and relaxing its edges. The edge leading the node 1 allows the modes walk and drive however mode drive cannot be used as no car2go is yet available. Therefore node 1 will be reached with time 1 hour (assuming walk speed of 4kmh). Then node 1 is settled and its edge leading to 3 is relaxed. Now the time comes to settle node 3 which has only one outgoing edge leading to node 1. However, node 1 is already settled which means that its best cost is already known. As node 3 is a car2go vehicle and as the edge allows drive mode then node duplication can be done. A new node with id 5, 1+4, is created and inserted in the queue with cost 1 hour and with leading node 3. Now node 5 is to be settled. As node 5 is greater than or equal *n* (*n=4*), this means we are in mode driving. Now we get the outgoing edges of node 5 mod 4 = 1 and relax them. There is only one outgoing edge from node 1 leading to node 2 and allows both walk and drive modes. The edge will be relaxed using both modes where driving mode will be preferred as it is faster. As result node 2 will be reached with cost 2 hours assuming driving speed of 60kmh.