Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

22th December 2016

Algorithms & Data Structures Solutions to Sheet 13 AS 16

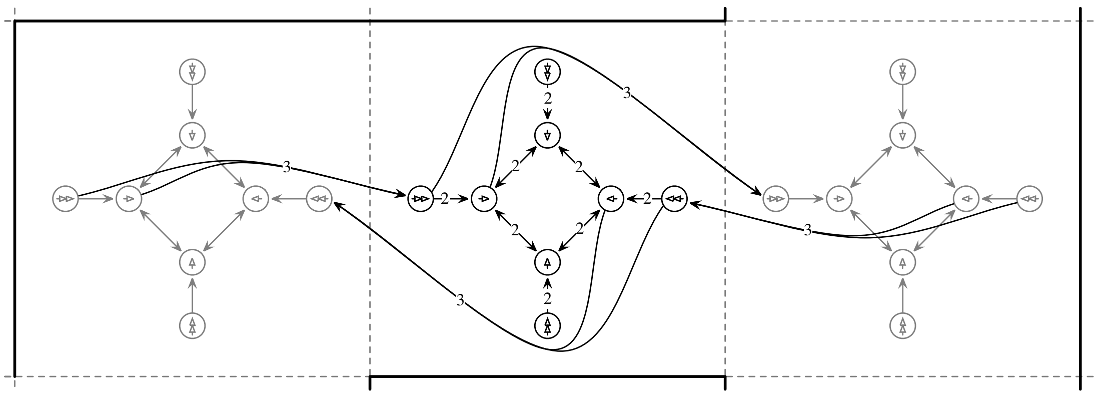
Solution 13.1 *Path Planning in Labyrinths.*

a) We define a directed graph that contains a vertex for every possible state of the system. A state is fully defined by the following three parameters:

   1) The field in which the robot is standing,
   2) the viewing direction of the robot and
   3) if the robot is stationary or in motion.

   For each field that the robot can visit, we thus have 8 states (4 viewing directions multiplied by 2 possibilities, i.e. whether the robot is stationary or in motion). We construct our graph with 8 vertices for each field of the labyrinth. Moreover, we need a vertex that represents the exit of the labyrinth. We add edges for moving, rotating and stopping and weight these edges by the time that the corresponding operation needs. The shortest path in the graph from the starting position of the robot to the exit corresponds to a sequence of operations of the robot that leads to a fastest possible escape.

   The following figure illustrates our construction for a small part of a labyrinth. The labels of the vertices denote the viewing direction of the robot; double arrows mean that the robot is in motion. Of course, we can remove states, if they cannot be reached.



b) As the graph contains no negative edge weights, we can find a shortest path using the algorithm of Dijkstra.

c) For each field in the labyrinth, we use a constant number of vertices (that is 8) and a constant number of edges (at most 20). If n is the number of fields in the labyrinth, then $|V| \in \mathcal{O}(n)$ and $|E| \in \mathcal{O}(n)$. Thus, the running time of the algorithm of Dijkstra is bounded by $\mathcal{O}(n \log n)$.

Note: For a directed, unweighted graph one can also find a shortest path using a breath-first search. As our graph contains only integer edge weights that are all larger than 0 and bounded above by a constant, we can construct an extended, unweighted graph by replacing every edge with weight $w > 1$ by w edges with weight 1 and adding $w - 1$ dummy vertices. If that edge in our original graph connects a vertex x with a vertex y, these vertices are connected by a path with w edges in the extended graph. Therefore, the extended graph only has edges with weight 1 and we can use a breadth-first search to find a shortest path. As the edge weights in the original graph are bounded above by a constant, we have added only $\mathcal{O}(n)$ vertices and edges and the running time is in $\mathcal{O}(n)$.