

# Schedule-Based Path Planning: An Optimal-Time Algorithm

Harry Kao  
harry@hairycow.name

November 9, 2008

## 1 Introduction

Many well-known graph search algorithms have been invented, including Dijkstra and A\*. Most research has been directed toward traditional graphs where one may “travel” along edges at will. However, there exists a set of problems involving graph-like structures where one may only travel from one vertex to another according to a well-defined schedule. One practical example is planning trips via buses and trains in a public transportation network. This paper addresses the problem of optimal-time path planning in schedule-based graphs.

## 2 Problem Definition

Let  $V$  be a set of vertices and  $S$  be a set of *segments* defined as follows:

$$S = \{(t_s, v_s, t_e, v_e) : t_s \leq t_e \wedge v_s, v_e \in V\}$$

Each segment is a 4-tuple containing a start time ( $t_s$ ), start vertex ( $v_s$ ), end time ( $t_e$ ), and end vertex ( $v_e$ ). Together, the segments in  $S$  define a *schedule* and indicate when travel is permitted from one vertex to another.

Furthermore, let  $T = (t_0, \delta, v_0, v_\zeta)$  where  $\delta \geq 0$  and  $v_0, v_\zeta \in V$ .  $T$  defines a *trip*, the optimization problem to be solved. We wish to find an optimal time path, called a *plan*, from  $v_0$  to  $v_\zeta$  starting at  $t_0$  that takes no longer than  $\delta$ .

## 3 Algorithm

We present a single-source all-paths algorithm for determining the optimal-time plan. Like Dijkstra’s algorithm, our strategy is to build a tree that encodes the optimal path from  $v_0$  to every other  $v \in V$ .

Define a function  $F(S, t_0, \delta)$  that returns the segments in the following set, ordered by  $t_e$ :

$$\{s \in S: t_0 \leq t_e \leq t_0 + \delta\}$$

Such a list can be constructed by sorting the elements of  $S$  by  $t_e$  and culling the segments that do not fall within the requested time range.

Let the *predecessor* of  $v$  be the vertex that a route passes through immediately prior to reaching  $v$ . Define a map  $M: v \rightarrow s$  where  $s$  is the segment that was used to travel to  $v$  from its predecessor.<sup>1</sup> Optimal routes can be retrieved from  $M$  by following predecessors from the end node back to  $v_0$ .

The following pseudo-code constructs an optimal-time  $M$ :

```

; initialize M
M = { v_0: null }

; iterate through the relevant segments
for s in F(S, t_0, d):

    ; unpack the segment
    (t_s, v_s, t_e, v_e) = s

    ; see if the segment spans the visited and unvisited vertex sets
    if (v_s in keys(M)) and (v_e not in keys(M)):

        ; retrieve the segment associated with the predecessor
        s_p = M[v_s]

        ; unpack the predecessor segment
        (t_p_s, v_p_s, t_p_e, v_p_e) = s_p

        ; if the segment start time is >= the predecessor segment end
        ; time, update the map
        if t_p_e <= t_s:
            M[v_e] = s

```

Upon termination, if  $v_\zeta \in \text{keys}(M)$ , then  $M$  contains the optimal-time path from  $v_0$  to  $v_\zeta$ .<sup>2</sup> It is then trivial to recover the optimal route from the predecessor information stored in  $M$ . If  $v_\zeta \notin \text{keys}(M)$ , a plan that satisfies  $T$  does not exist.

Optimality can be proven by induction. Suppose  $M$  has been partially constructed and  $A(v)$  is the arrival time at vertex  $v$ . Let the set of visited vertices be  $V_v = \text{keys}(M)$  and the set of unvisited vertices be  $V_u = V - V_v$ . Since we iterate through the segments in order of increasing  $t_e$ , the next successive segment from  $F(S, t_0, \delta)$  that allows travel from  $v_v \in V_v$  to  $v_u \in V_u$  for which  $t_s \geq A(v_v)$  must be the final segment in the optimal route from  $v_0$  to  $v_u$ . Thus,

<sup>1</sup>Note that the predecessor itself is represented by  $v_s$  in  $s$  so there is no need to include it explicitly in  $M$ .

<sup>2</sup>Clearly, if one desires only the optimal path from  $v_0$  to  $v_\zeta$ , this can be converted to a single-path algorithm by terminating the loop after  $v_\zeta$  is added to  $M$ .

the requirement that  $F$  return a list of segments sorted in order of increasing  $t_e$  is a necessary and sufficient condition for the optimality of  $M$ .

## 4 Extensions

- This algorithm can be adapted to find the optimal-time route from every vertex to a specific vertex arriving no later than a target time by making time run backwards.
- A trivial modification allows  $M$  to represent the optimal route from a set of vertices  $V_0$  to every other  $v \notin V_0$ . Simply initialize  $M$  by setting all  $v \in V_0$  to `null`.