Path planning in games

Marjan van den Akker [*] Roland Geraerts [†] Han Hoogeveen (Speaker) [‡]
Corien Prins [§]

1 Introduction

Path planning is one of the fundamental problems in games. The path planning problem can be defined as finding a collision-free path, traversed by a unit, between a start and goal position in an environment with obstacles.

The variant we study is the problem of finding paths for one or more *groups* of units, such as soldiers or tanks in a real-time strategy game, all traversing the same (static) environment. Each group has its own start and goal position (or area), and each unit will traverse its own path. The objective is to find the paths that minimize the average arrival times of all units.

Our main contribution is that we propose an efficient solution based on techniques from integer linear programming for the path planning problem with groups. Here we do not care about possible interference of paths; this will be taken care of by a collision-avoidance algorithm. Our solution can be used to handle difficult situations which typically occur near bottlenecks (e.g. narrow passages) in the environment. Initially, we restrict ourselves to the situation in which each unit has the same width and speed; we will address the consequences of relaxing this assumption later. Moreover, we will also discuss how we can use our approach in a game environment.

2 Basic algorithm

To solve the problem, we first construct a *directed* graph that resembles the free space in the environment; we will discuss the issue of undirected edges later. There are several ways to create such a graph. One possibility is to use tiles, as is done in earlier games, but this is considered to lead to unnatural paths. A better alternative is to use a waypoint graph [3] in combination with a navigation mesh [4]. No matter how the graph has been constructed, we determine for each arc in the graph the traversal time as the time it takes to traverse the arc. We further determine its capacity as the number of units that can traverse the arc while walking next to each other.

[*]`marjan@cs.uu.nl`. Department for Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[†]`roland@cs.uu.nl`. Department for Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[‡]`slam@cs.uu.nl`. Department for Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[§]`c.r.prins@tue.nl`. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Given this graph, the problem of finding a set of optimal paths boils down to a multi-commodity flow problem; see the overview paper by Skutella [5]. If there is only one group of units, then this problem can be solved efficiently by the dynamic flow algorithm by Ford and Fulkerson [6]. Baumann and Skutella [7] have proposed an efficient algorithm to solve the variant in which all units are identical and we have only one origin and several destinations. For the general case, no efficient algorithm is known. We want to present a heuristic that is based on techniques from (integer) linear programming. The basic idea is that we formulate the problem as an integer linear program (ILP), after which we solve the LP-relaxation and convert this to an integral solution. In this way, we make the problem tractable, without loosing too much on quality.

Instead of using variables that indicate for each arc at each time the number of units of group k that traverse this arc (an arc formulation), we use a formulation that is based on *paths* for each origin-destination pair. A path is described by the arcs that it uses and the times at which it enters these arcs. Here we require that the difference in the entering times of two consecutive arcs (i, j) and (j, k) on the path is no less than the traversal time $l(i, j)$ of the arc (i, j); if this difference is larger than $l(i, j)$, then this implies that there is a waiting time at j. Initially, we assume that there is infinite waiting capacity at all vertices. The cost coefficient of a path is equal to the time at which the end-point is reached.

Given the set of possible paths with their characteristics, we introduce for each path s a decision variable x_s corresponding to the number of characters that will follow this path. The goal is to minimize the total cost (total arrival time) subject to

- the constraint that for each origin-destination pair at least the desired number of characters are sent;
- the capacity constraints of the arcs at each time point.

To find out which paths are useful, we solve the LP-relaxation (which we obtain by allowing fractional values of x_s) using column generation. Here the pricing problem boils down to finding a shortest path for each origin-destination pair in the *time expanded* graph. This graph is created from the original graph as follows: we create a set of vertices (v, t) for each original vertex v and each timepoint $t = 0, \ldots, T$, where T is the time-horizon; we put an arc between vertices (v, t_1) and (w, t_2) if $t_2 - t_1$ is equal to the traversal time of the arc (v, w) in the original graph. The length of each arc is equal to the time-difference of the vertices with a correction for the dual multipliers. If we find an improving path, then we add it, together with the set of paths that are obtained from it by shifting the time. Eventually, we solve the LP-relaxation this way.

3 Extensions and applicability in a game

Games (and especially strategy and shooter games) are highly dynamic. This yields a number of game-specific challenges that we have to overcome before our approach can be applied in a game. These challenges are

- It must run in real-time;
- We do not want to use paths that lead to isolated units in a war game;

- The environment may change;
- There can be different types of units (different speed, width);
- It must be integrated with a local rule for collision avoidance.

The real-time aspect makes it impossible to solve the ILP, even if we have restricted the number of variables dramatically. But fortunately there is no need for finding the optimal solution. After we have solved the solution to the LP-relaxation, we can just round it down. Moreover, we do not even have to solve the LP-relaxation to optimality: we can use column generation to improve the current solution, until we run out of time. To avoid that not enough characters reach their target after rounding down, we increase the number of characters that have to be sent. This yields a very beneficial side-effect: we end up with a set of compatible paths, from which we can select the right number in a preprocessing step, in which we can take other criteria into account, like the 'no isolated units' constraint. Since a change in the LP can be solved starting with the former solution our method is well suited to deal with imminent changes of the environment affecting the capacities of the arcs (for example because of a bridge that gets destroyed).

Next to these challenges, there are some extensions to the model that we should incorporate. Release dates and deadlines for the arrival at a given destination are easily incorporated by putting constraints on the time-expanded graph. It is more difficult to model *undirected* edges in the graph. Enforcing the total capacity without any side-constraints is possible, but this leads to an enormous number of additional constraints, especially when the traversal time is big. Therefore, we replace an edge by two arcs with total capacity equal to the capacity of the edge. Currently, we use a constant division of the total capacity, where the capacity division is left to the LP; it is also possible to make this division time-dependent.

References

[1] S. Rabin (2004). *AI Game Programming Wisdom 2.* Charles River Media Inc.

[2] R. Geraerts (2010). Planning short paths with clearance using explicit corridors. *IEEE International Conference on Robotics and Automation*, 1997–2004.

[3] M. Skutella, 2008. *An Introduction to Network Flows Over Time.* http://www.math.tu-berlin.de/coga/publications/techreports/2008/Report-022-2008.xhtml

[4] L. Ford Jr. and D. Fulkerson (1958). Constructing maximal dynamic flows from static flows. *Operations Research 6*, 419–433.

[5] N. Baumann and M. Skutella (2009). Earliest arrival flows with multiple sources. *Mathematics of Operations Research 34*, 499–512.