Precomputed-Direction Heuristics for Suboptimal Grid-Based Path-Finding

Álvaro Parra and **Álvaro Torralba** and **Carlos Linares López**
Planning and Learning Group, Universidad Carlos III de Madrid
Leganés (Madrid) - Spain
alvaro.parradm@gmail.com, {alvaro.torralba, carlos.linares}@uc3m.es

Abstract

This paper describes *BubbleDragon*, an entry in the 2012 Grid-based Path-Planning Competition. We aim to solve path-finding problems in the minimum time possible by precomputing paths from states in a region to its frontiers. Experimental results show that suboptimal paths for 1024x1024 grids can be retrieved in less than 1ms on average.

Introduction

Path-finding is a particular class of problems whose goal consists of finding a path from a source node s to a goal state g in a graph $G(V, E)$. Grids are undirected planar graphs so that movement is restricted to cardinal and octile directions and obstacles in some tiles prevent us to move through them. The problem is divided in two phases. In the preprocessing phase a grid map of at most 2048x2048 tiles is given to the program which may produce up to 50MB of preprocessed data in at most 30 minutes. Then, the algorithm must find a path between pairs of states s and g as fast as possible.

Most approaches use A*-based algorithms with automatically derived heuristics which make use of the preprocessed data. There are several approaches taking advantage of a partitioning of the map to derive better heuristic estimates, such as gateway heuristics (Björnsson and Halldórsson 2006) or portal-based true-distance heuristics (Goldenberg et al. 2010). Other approaches aim to solve the problem without any search: by precomputing the optimal solutions and compressing the data (Botea 2011) but they do not meet the memory requirements allowed in the competition. Our approach seeks to use precomputed partial solutions using only the available memory at the expense of not optimizing the cost. It partitions the state space into regions to divide the path-finding problem into smaller subproblems and precompute their solutions, avoiding search whenever possible.

Our Approach

In the preprocessing phase, the map is divided into connected regions. The algorithm is initialized by considering each transitable state a region. At each step a pair of adjacent regions are merged, until the number of regions is below the desired threshold. In order to obtain regions as balanced and convex as possible, the region with the smallest frontier is merged with the neighbour with which it shares the largest frontier. Ties are broken in favour of smaller regions.

Regions are used as sub-goals, so that the search is divided into shorter searches. All-pair shortest distances between all regions are computed, considering unitary costs for traversing the regions in order to minimize the number of regions traversed (instead of the total distance).

A gateway between each pair of adjacent regions is defined as the pair of connected states in the frontier between both regions nearest to the centroid of the frontier. A backwards Dijkstra search is performed from every gateway, precomputing the next step to optimally reach it from every state in its region. In order to meet the memory requirements, we store up to 16 directions for each point [1]. If a region has more than 16 neighbours, only the path to the first 16 gateways is precomputed, so that search will be necessary for reaching others. When a region has less than 16 neighbours, the remaining memory is used to store the number of steps in the optimal path to the gateway following the precomputed direction. Furthermore, if even more memory is available (when there are 5 or less neighbours) more than one direction is stored, allowing more steps to be retrieved in one single lookup.

To find a path between two states s and g, there are two possible cases, whether they are in the same region or not. If s and g are in different regions we find a path from s to a neighbour region nearer to the region of g and update s. In this case, no search is necessary since the path until the frontier gateway is already precomputed.

When s and g are in the same region, if s is a gateway (which is likely, because we entered the region by using precomputed steps) the result is the inversion of the precomputed path from g to s. When the two points are in the same region but none of them is a gateway, a Dijkstra search is performed (in those cases, given that the regions are usually small, non-informed search has good performance).

As the probability of s and g being on the same region is

[1] Spending 3 bits per direction, 6 bytes are necessary for each tile, resulting in $(2048^2 \cdot 6 =)$24MB. The rest of the memory is spent in storing the region of each point $(2048^2 \cdot 2 =)$8MB, the distance between regions $(1023^2 \cdot 2 \approx)$2MB and some extra space for the open and close list.

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

	1023 regions			511 regions			255 regions		
	Total (µs)	Step (µs)	Subopt	Total (µs)	Step (µs)	Subopt	Total (µs)	Step (µs)	Subopt
BG	70 (14)	7 (6)	1.15 (0.04)	37 (11)	6 (8)	1.18 (0.03)	22 (3)	3 (1)	1.18 (0.03)
DAO	90 (50)	9 (13)	1.16 (0.11)	45 (24)	5 (7)	1.17 (0.10)	27 (15)	4 (7)	1.18 (0.09)
SC	83 (18)	7 (7)	1.23 (0.05)	46 (12)	6 (8)	1.23 (0.04)	336 (2632)	313 (2632)	1.23 (0.05)
WCIII	66 (14)	12 (13)	1.19 (0.03)	33 (7)	5 (6)	1.20 (0.02)	21 (1)	4 (1)	1.21 (0.03)
Maze 1	178 (26)	6 (1)	1.00 (0)	114 (17)	7 (0)	1.00 (0)	88 (14)	9 (1)	1.00 (0)
Maze 2	180 (13)	6 (0)	1.01 (0)	113 (8)	7 (0)	1.00 (0)	84 (6)	9 (0)	1.00 (0)
Maze 4	193 (32)	5 (0)	1.04 (0)	98 (16)	4 (0)	1.02 (0)	63 (10)	5 (0)	1.01 (0)
Maze 8	264 (48)	4 (1)	1.11 (0.01)	118 (21)	4 (0)	1.07 (0.01)	69 (12)	4 (0)	1.04 (0.01)
Maze 16	304 (47)	4 (0)	1.26 (0.02)	135 (19)	3 (0)	1.18 (0.02)	77 (10)	3 (0)	1.12 (0.02)
Maze 32	192 (30)	4 (1)	1.17 (0.01)	107 (17)	3 (0)	1.27 (0.02)	70 (10)	3 (0)	1.30 (0.02)
Random 10	59 (1)	5 (1)	1.20 (0.01)	37 (1)	6 (1)	1.20 (0.01)	26 (1)	6 (1)	1.21 (0.01)
Random 20	63 (1)	5 (0)	1.20 (0.01)	39 (1)	5 (0)	1.21 (0.01)	29 (1)	6 (0)	1.22 (0.01)
Random 30	68 (1)	6 (0)	1.20 (0.01)	42 (2)	6 (1)	1.21 (0.01)	34 (4)	7 (2)	1.23 (0.01)
Random 40	126 (19)	5 (1)	1.09 (0.01)	71 (10)	4 (1)	1.09 (0.02)	48 (7)	5 (1)	1.08 (0.02)
Rooms 8	64 (1)	5 (0)	1.18 (0.01)	38 (1)	5 (1)	1.19 (0.01)	28 (2)	6 (1)	1.20 (0.01)
Rooms 16	72 (1)	4 (0)	1.07 (0)	37 (2)	4 (0)	1.15 (0.01)	25 (1)	5 (0)	1.18 (0.01)
Rooms 32	68 (4)	4 (0)	1.16 (0.01)	39 (1)	4 (0)	1.24 (0.01)	25 (1)	4 (1)	1.06 (0.01)
Rooms 64	68 (4)	4 (1)	1.12 (0.01)	39 (1)	4 (1)	1.21 (0.02)	25 (1)	4 (1)	1.16 (0.01)

Table 1: Total search time, maximum time per step and suboptimality rate for different kind of maps in the benchmark suite. All the results show the average value and standard deviation.

very low, in most cases the algorithm just retrieves the precomputed steps. However, the algorithm is not optimal since adding the gateways between regions as sub-goals might worsen the path quality. On the other hand, it is trivial to see that the algorithm is complete.

Results

Experiments were performed over a set of benchmarks (Sturtevant 2012) with four types of maps: video games, mazes, random maps and rooms. All reported results are average values over 10 512x512 grids, except in video games where the number of maps ranges from 41 (WC III) to 156 (DAO) and their size from 21x30 to 1024x1024. The number of paths computed per map is around 10,000 in mazes and around 1,000 or 2,000 for the rest.

The machine used has a 2.26 GHz Intel Core i3-350M processor and 4 GB RAM. The precomputing phase uses four threads, with a time limit of 30 minutes. The maximum amount of precomputed data was 11.3MB for all the tested benchmarks. In the search phase a single thread is used.

Table 1 shows the total time, maximum time per step and suboptimality rate (*Subopt*) when executing our algorithm with different thresholds on the number of regions: 1023, 511 and 255. The suboptimality rate is the path cost divided by the optimal cost, so that a suboptimality value of 1 stands for an optimal solution. In all cases, the average total search time is under 1 ms as most of the path steps are precomputed and little or no search is needed. An exception occurs when using only 255 groups in the StarCraft (SC) set, where an anomaly is observed in a 1024x1024 map with 254 isolated components. The region partitioning does not generate enough adjacent regions so the algorithm derives in a non-informed Dijkstra search. On the other hand, the quality of the solutions is far from optimal, except in maps in which the frontiers between regions are really small, such as *Maze 1* or *Maze 2*. When comparing the different configurations on the number of regions, the ones with lower regions are faster, mainly due to less intermediate searches. Regarding the time per single step and the suboptimality rate, there are no significant differences among the different versions.

The version submitted to the competition used a threshold on the number of regions of 1023. Even though experimental results suggest lower runtimes for configurations with less regions, as maps in the competition can be larger than in our benchmarks, we expect the 1023-regions version to be more consistent for arbitrary maps.

Acknowledgements

This work has been supported by the Spanish MICINN projects TIN2011-27652-C03-02, TIN2008-06701-C03-30.

References

Björnsson, Y., and Halldórsson, K. 2006. Improved heuristics for optimal path-finding on game maps. In Laird, J. E., and Schaeffer, J., eds., *AIIDE*, 9–14. The AAAI Press.

Botea, A. 2011. Ultra-fast Optimal Pathfinding without Runtime Search. In *AIIDE-11*, 122–127.

Goldenberg, M.; Felner, A.; Sturtevant, N. R.; and Schaeffer, J. 2010. Portal-based true-distance heuristics for path finding. In *SOCS*.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*.