Multi-Target Search in Euclidean Space with Ray Shooting

**Ryan Hechenberger, Daniel Harabor, Muhammad Aamir Cheema,
Peter J Stuckey, Pierre Le Bodic**

Faculty of Information Technology, Monash University, Australia
{Ryan.Hechenberger, Daniel.Harabor, Aamir.Cheema, Peter.Stuckey, Pierre.LeBodic}@monash.edu

Introduction

The shortest path problem (SPP) is a well studied problem where we want to find a minimum path length along a graph. Many search algorithms are based around the A* algorithm (Hart, Nilsson, and Raphael 1968), which works by repetitively expanding a vertex v based on an *f value* given by the sum of a *g value* (shortest path length from start to v) and a heuristic *h value*, an under-estimate path length from v to target. This expansion will add the edges from v to other vertices onto the path ending at v to iteratively progress the best-looking path towards a target point until that target is reached.

We also consider the Euclidean environment in 2-dimensions, where we can assign points on a Cartesian coordinates, in a 2D plane as an (x, y) value from the $(0, 0)$ origin. This environment can make use of polygonal shapes as *obstacles* that directly correlate to objects on the plane, such as a table being represented as a rectangle relative to the center of the plane $(0, 0)$. A set of these obstacles can be considered as untraversable on the plane, with a start and target point to find the shortest path that does not cross-over into any of these obstacles to be the Euclidean SPP (ESPP). This kind of problem is useful in areas such as video games (Algfoor, Sunar, and Kolivand 2015), as the ESPP is not defined as a graph but a set of obstacles.

Studied methods of solving the ESPP primarily includes the visibility graph (VG) (Lozano-Pérez and Wesley 1979) and navigation meshes (Cui, Harabor, and Grastien 2017), which uses pre-processing to convert the Euclidean environment into another form more suitable for search. We developed a search algorithm called RayScan (Hechenberger et al. 2020) that can search without pre-processing by discovering on-the-fly the Sparse VG (Oh and Leong 2017). This allows for search on dynamic environments, where obstacles can be moved, added or removed between start-target queries, which other methods will have to spend additional pre-processing time to complete.

RayScan

RayScan works by use of A* search algorithm with on-the-fly edge discovery. The Euclidean distance to the target is used as the h value. A core concept is ray shooting, were we shoot a ray from some point a in a direction and determine the first intersecting obstacle and where it intersected.

When shooting a ray from u to v, if there are no obstacles between the two then we have discovered a edge on the VG (along with an obstacle pass v), otherwise we have discovered an impeding obstacle. Either case we have a new obstacle we must consider how to get around, in which only clockwise (`CW`) and counter-clockwise `CCW` are possible. This leads to second concept, the scan. We scan by rotating a *scan-line*, a line from expanding u to the intersection point, and rotate around u following along the obstacle in a specified orientation `CW` or `CCW` until we find a vertex we can bend around, called the *turning point*. This point is the v point, in which a new ray will be shot from u to v to determine an edge or not, leading to a recursive algorithm.

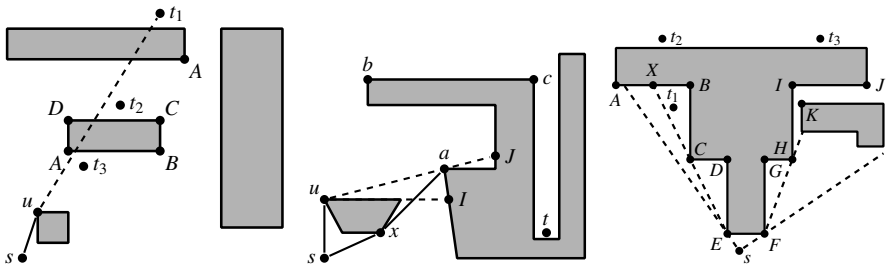
With recursive algorithms, we need a base case to prevent infinite looping. This is done with two methods, the first is ending a recursion when a ray that we have already shot being shot again; the other is by means of an *angled sector* (AS). The AS has an extreme `CW` and `CCW` angle, where any angle from the `CW` angle truing `CCW` up to the `CCW` angle is consider to be inside the AS, otherwise it is outside. Each scan is given an AS that it must stay within, otherwise ends the recursion; when can then split the AS into a `CW` and `CCW` split along a ray shot, which will be given as the new AS in the next recursion from that ray.

RayScan spends the most time working on ray shooting, as such this paper will detail extension methods that reduces the number of shot necessary. These extensions are known as the blocking, skip and bypass extension, and the three of them comprises the enhanced algorithm RayScan+. We also consider RayScan for the multi-target search, where as start at a point s to find the shortest paths to from s to all targets t_i.

Blocking Extension

The blocking extension is primarily useful for multi-target search. RayScan searches for the target by first shooting a ray towards some t_i. If during a scan, the scan lines passes a target that has not been shot, it may decided it does not need to shoot to that target later.

Consider Figure 1a, when expanding s we first shoot to t_1, which is blocked by obstacle $ABCD$. When scanning `CW`



(a) Blocking extension (b) Skip extension (c) Bypass extension

Figure 1: Auxilliary diagrams

from that ray, the scan-line follows the edges unto it reaches turning point C; we find that scan-line passing t_2 and t_3, the scan-line does not intersect t_2, thus is blocked by the obstacle we are scanning, therefore we do not need to shoot to that target later, while t_3 intersect the scan-line therefore is not blocked. Once t_1 scan is completed, we normally start t_2, except we do not shoot to t_2 thus we skip to t_3, which we will have to scan.

Skip Extension

The skip extension works by avoiding to shoot to a vertex whose path length is worse than we already discovered. When expanding u, we consider turning point v; normally we shoot a ray and continue, except if g value of u plus distance from u to v is greater than g value of v, we know that edge uv will not improve the search to v. Skip will avoid shooting the uv ray by trying to *scan-past* the turning point v by continuing the scan, ignoring the orientation of the scan line until that line passes the uv ray, in which case we skip shooting that ray and continue to the next turning point.

Take Figure 1b for example, where expanding u our scan CCW goes from I to turning point a. Path $s-u-a$ is longer than $s-x-a$, thus we do not need to consider edge ua, and will attempt to skip the ray shot by continuing the scan, ignoring the CCW orientation requirements. Since this scan does succeed at passing the ua ray when it passes point J, the skip succeeds and we continue to find turning point b. If we were unable to skip a turning point, we will resort to shooting a ray and progress to the next scan(s).

Bypass Extension

The bypass extension avoids shooting ray's from expanding vertex u to vertex v in cases where v leads to a dead end. We explain by example with Figure 1c, let $u = E$ and $v = C$, we consider C for bypass by attempting to scan-past C, in this case passing the EC ray at point X. Being able to scan-past C is the first requirement of bypass, the second is that the scan-past *must* stay within the scans AS, the third is dependent on the targets location; if the scan-past scan-line intersects the target then we cannot bypass, if it does not than the third condition will be met. For this example, if t_1 is our desired target, then the scan-line intersects and thus we cannot bypass, while if t_2 is our target then bypass is possible, and the scan will continue on from X.

Algs	Aftershock	Aurora	ArcticStation
R	400 (1.0R)	2548 (1.0R)	2689 (1.0R)
RB	388 (0.97R)	2492 (0.98R)	2627 (0.98R)
RS	375 (0.94R)	2392 (0.94R)	2090 (0.78R)
RP	262 (0.65R)	1638 (0.64R)	1859 (0.69R)
RBSP	**248** (0.62R)	**1560** (0.61R)	**1395** (0.52R)
RC	150 (0.38R)	736 (0.29R)	683 (0.25R)
RBSPC	**132** (0.33R)	**723** (0.28R)	**625** (0.23R)

Table 1: Average runtime comparision of extensions in µs for maps Aftershock, Aurora and ArcticStation; Algs list the extensions: R- RayScan; B- blocking; S- skip; P- bypass; C- ray caching

This leads to the possibility of considering the same vertex twice, where first we bypass but second time we do not. For example, let $t = t_3$, $u = F$ and $v = H$, the first time reaching H is by a CW scan from FH, bypass succeeds and we find turning point J, leading into a CCW scan from ray FJ to find turning point K, followed by a CW scan from ray FK to again find H, this time when considering H for bypass the scan-past will leave the AS and thus fail to bypass, therefore will will shoot FH and find it as a successor.

Results

Table 1 shows an ablation study of our extension methods applied individually and all together. The experiments are done using Starcraft 2 maps from the Moving AI Lab pathfinding benchmarks (Sturtevant 2012).

The ray caching is a simple method of storing results of shot rays from previous queries to speed up ray shooting as a whole, as we see gives significant improvements; although requires additional memory and invalidates on changes to the environment.

We see minor improvements with the blocking extension; although for multi-target this extension vastly improves. Skip extension has some small improvements except for ArcticStation in which the improvements are greater. Bypass has the best improvements overall (excluding caching), and as long as the maps are not using convex obstacles exclusively, it will have noticeable speedups.

RH: Will add repository link, have not forgotten

Conclusion

We have developed RayScan+ that incorporate extension methods that greatly improve RayScan query times.

The full paper details multi-target RayScan in greater details, along with additional benchmarks and comparisons to the state-of-the-art ESPP Polyanya.

References

Algfoor, Z. A.; Sunar, M. S.; and Kolivand, H. 2015. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology* 2015: 1–11. ISSN 1687-7047.

Cui, M. L.; Harabor, D.; and Grastien, A. 2017. Compromise-free Pathfinding on a Navigation Mesh. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 496–502. AAAI Press.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2): 100–107. ISSN 0536-1567.

Hechenberger, R.; Stuckey, P. J.; Harabor, D.; Le Bodic, P.; and Cheema, M. A. 2020. Online Computation of Euclidean Shortest Paths in Two Dimensions. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling*, 134–142. AAAI Press.

Lozano-Pérez, T.; and Wesley, M. A. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22(10): 560–570. ISSN 0001-0782.

Oh, S.; and Leong, H. W. 2017. Edge N-Level Sparse Visibility Graphs: Fast Optimal Any-Angle Pathfinding Using Hierarchical Taut Paths. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS 2017)*.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2): 144 – 148. URL http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.