Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-19)

Pathfinding and Abstraction with Dynamic Terrain Costs

Nathan R. Sturtevant
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
nathanst@ualberta.ca

Devon Sigurdson,
Bjorn Taylor, Tim Gibson
Improbable Canada Inc.
Edmonton, AB, Canada
{devon, bjorntaylor, timgibson}
@improbable.io

Abstract

Abstraction and refinement is a common approach used in games to improve the speed of pathfinding by planning in an abstract space and then refining abstract paths to traversable paths. While there are many variants of this approach that have been developed and studied, research on this problem has largely ignored the problem of pathfinding with terrain types, terrain costs, and dynamic terrain. This paper studies the problem of pathfinding in domains with terrain costs and proposes an abstraction approach that is built around handling terrain costs and dynamic terrain. The resulting approach is able to handle costs in a way that existing approaches do not, and provides a good balance between memory usage, path quality, and pathfinding speed.

1 Introduction and Overview

Abstraction and refinement approaches, also known as hierarchical pathfinding, have a long history (Holte et al. 1996) and can be very effective. Generally speaking, abstraction techniques require a small amount of memory and allow small amounts of suboptimality in exchange for large speedups in pathfinding. This approach was applied on top of a grid representation in Dragon Age: Origins (Sturtevant 2007), and more recently over a voxel representation in Castle Story (Alain 2018).

But, existing work has focused primarily on the problem of moving through uniform cost terrain. Although some work has experimented with non-uniform terrain costs (Mould and Horsch 2004; Harabor and Botea 2008), no work has looked at the problem of hierarchical pathfinding in a dynamic world with changing terrain types and costs. Work in this area is needed, because incorporating terrain costs in search typically weakens the heuristic, making the pathfinding problem more difficult, and increasing the need for faster pathfinding. Terrain costs are particularly important for making pathfinding more realistic by adding interesting constraints on how characters move in the world (Kapadia et al. 2013; Sturtevant 2013; Ninomiya et al. 2015). With terrain costs optimal pathfinding is particular difficult (Mitchell and Papadimitriou 1991), because the optimal path should 'refract', in a similar manner to how light refracts when it passes from air to water, as it passes from terrain of one type to another, which motivates the need for suboptimal approaches. Furthermore, some algorithms require pre-processing (Mata and Mitchell 1997; Aleksandrov, Maheshwari, and Sack 2000), which makes the approaches more difficult to apply when terrain changes dynamically at runtime.

The research in this paper was motivated by and is the result of an ongoing collaboration between the University of Alberta and Improbable Inc's Canada studio. The work represents a first approach to improving the performance of pathfinding with weighted terrain so that significantly different weights can be used to dynamically create novel and unique behaviors for different character types in the game. In this context, the paper makes the following contributions. First, it describes a method of dynamically abstracting a map based on terrain types, which we call a Dynamic Terrain Abstraction. The approach assumes that there are a small number of terrain types in the map, each with different cost, but the approach can be used on both grids or triangle/polygon meshes. The paper then shows how to perform abstract pathfinding and refinement from this representation, describing a number of design choices that may influence the performance of the approach. The approach is then evaluated in comparison to algorithms such as A*, Weighted A*, and NBS. The resulting implementation is approximately 50x faster than optimal pathfinding algorithms. When compared to suboptimal algorithms with comparable solution quality, the approach is still more than 10x faster. Thus, this represents an important approach to pathfinding with terrain costs.

2 Related Work

A* (Hart, Nilsson, and Raphael 1968) is the canonical algorithm for pathfinding in games, but it is known for being slow, as it finds optimal paths and does not take any input besides a heuristic and the state space to search. Thus, many papers claim to have approaches that are 'better than A*'. But, what is usually meant by this is that something new has been added to the basic A* algorithm to improve its performance. This includes things like better heuristics (Sturtevant et al. 2009), different constraints on successor generation (Harabor and Grastien 2011), or different search representations (Botea, Müller, and Schaeffer 2004). One of the few alternatives to unidirectional A* search is bidirectional search. Near-Optimal Bidirectional Search (NBS) (Chen et al. 2017) is a recent bidirectional algorithm which is guaranteed to do no more than twice the work of A* or any other algorithm, but can be arbitrarily better in practice. Bidirectional search is currently not used broadly in practice, but the weakening of the heuristic that occurs from the different terrain types can be handled by bidirectional search (Barker and Korf 2015).

A* is commonly used as part of the abstraction/refinement process in hierarchical pathfinding approaches including Hierarchical Pathfinding A* (HPA*) (Botea, Müller, and Schaeffer 2004) and Partial-Refinement A* (PRA*) (Sturtevant and Buro 2005). These papers describe abstraction approaches that use A* to find a path in an abstract graph before refining the path into the actual search space. The refinement process also uses A* (with local subgoals), the results of which can be cached. PRA* was adapted to ship in Dragon Age: Origins using a memory-efficient abstract representation (Sturtevant 2007). Variations on this approach have been studied with other search techniques used at the abstract layer (Sturtevant and Geisberger 2010). They have also been applied to dynamic worlds (Sturtevant 2011), adapted to work with navigation meshes (Pelechano and Fuentes 2016), and in 3D state spaces (Wardhana, Johan, and Seah 2013). Theoretical bounds for path quality and total planning time have also been developed (Bulitko et al. 2007; Sturtevant and Jansen 2007). Other work has looked at alternate representations for planning, but not necessarily using hierarchical methods (Ferguson and Stentz 2006; Brewer 2017).

Researchers have recognized a range of scenarios which require planning in weighted terrain (Kapadia et al. 2013; Sturtevant 2013; Ninomiya et al. 2015). Weighted terrain has also been explored suboptimally in Theta* (Daniel et al. 2010) and has been used experimentally with weighted terrain (Mould and Horsch 2004). The only work which has modeled weighted terrain (along with variable-sized agents) is HAA* (Harabor and Botea 2008). This is important related work, but it requires re-computing edge costs for the entire map if the relative cost of a terrain changes at runtime.

3 Problem Definition

Imagine a world populated by both people and deer. While everyone is able to move through all types of terrain, each creature type has preference for its own terrain. People prefer to travel on roads, while deer prefer to travel in the woods and off of the trails used by humans. But, outlaws may prefer to avoid roads when possible. If a fire is started in the woods, though, all creatures might then want to avoid the fire. Speaking generally, the problem studied in this paper is how to perform pathfinding in a world with different terrain types, where each player in the game has their own dynamic preferences about how to travel. This problem is related work in multi-criteria objectives for pathfinding (Funke and Storandt 2013), but with more dynamic costs.

More formally, the input to a search problem is a map or a state space consisting of a set of primitive locations in Euclidean space, whether grids or polygons, which can be connected to form a graph $\mathcal{G} = \{V, E\}$. It is assumed that there are a finite number of discrete edges between locations, and that general movement through free space is not considered as part of the search. Each location $v \in V$ is associated with a specific terrain type $t(v)$ with cost $t_c(v)$. The cost of traversing an edge $e = \{v_i, v_j\}$ depends on the cost of the terrain being traversed. Assuming that $d(e) = d(v_i, v_j)$ is the distance between v_i and v_j, the cost of this edge is defined as $c(e) = c(v_i, v_j)$. In grids it is assumed that half the edge is traversed at cost $t_c(v_1)$ and the other half is traversed at cost $t_c(v_2)$ so $c(e) = d(e)(t_c(v_1) + t_c(v_2))/2$, but other definitions can be used in other representations. Note that, the length of an edge, a measure of distance, is a different measure than the cost of an edge, which measures the cost of moving a given distance.

In addition to the search graph, a search problem is also defined by a start state v_s, a goal state v_g, and a heuristic function $h(v_i, v_g)$. The heuristic is assumed to be admissible meaning that $h(v_i, v_g) \leq c(v_i, v_g)$. The goal is to return a path between v_s and v_g. It is assumed that an algorithm for solving search problems will be presented sequentially with many different search problems, and between any two problems the search graph is allowed to change. Any algorithm that solves this problem must balance the total memory used, the time required to find a path, and any time for static processing of $\mathcal{G}$. There is no single correct balance between these measures; instead there is a pareto-optimal frontier of approaches (Sturtevant et al. 2015) which possibly provide the best performance depending on the needs of a specific application.

4 Dynamic Terrain Abstraction

This paper proposes a Dynamic Terrain Abstraction (DTA) approach to solve dynamic terrain search problems. The approach builds most closely on the minimum memory (MM) abstraction used for Dragon Age: Origins (Sturtevant 2007).

In the MM abstraction, the map is divided into sectors, large grid squares of uniform size. Then, all groups of states within a sector that form a connected component (can be reached without leaving the sector) are abstracted into a single region. Each region becomes a node in an abstract graph $\mathcal{G}'$. Edges are added between regions that are adjacent (have adjacent states in $\mathcal{G}$). Because all connected states within a sector are grouped into a single region, edges in $\mathcal{G}'$ are only added between regions in different sectors.

This is illustrated in the left portion of Figure 1, with the colors representing a variety of different terrain types and black representing blocked terrain. The 32x32 grid is broken into 16x16 sectors. All sectors have a single region except the lower-left region, which is split into two regions by the obstacle. Ignoring terrain types will cause paths to pass through regions of different cost. The middle of Figure 1 shows the HPA* representation of the same state space. The HPA* representation is a dual representation of the MM abstraction. In the MM abstraction, nodes represent regions

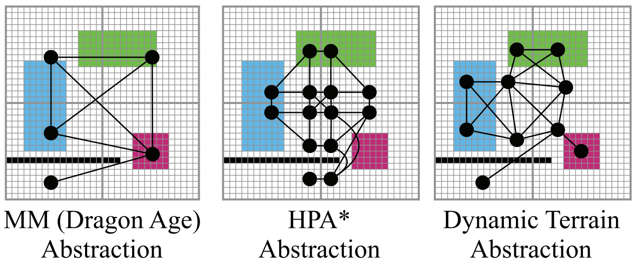

Figure 1: The Dragon Age MM abstraction (left), an HPA* abstraction (middle), and the dynamic terrain abstraction (right).

of space, and edges represent transitions between those regions. In the HPA* abstraction edges represent regions of space and nodes represent the transitions between those regions. HPA* can incorporate terrain costs into abstract edges (Harabor and Botea 2008), but it does not directly reason about regions of a given terrain type.

The DTA makes a number of changes from the basic MM approach. First, states are only abstracted together if they have the same terrain type $t(v)$. As before, abstracted states form regions, which are vertices in the abstract graph. But, because a single sector may contain several terrain types, a sector can also have several adjacent regions. Edges are added to $\mathcal{G}'$ if two different regions have adjacent states in the original graph. Thus, there may now be internal edges between regions within a single sector in addition to edges between regions in different sectors. Each region is associated with a single representative state in the region, which is used as the location of the region in $\mathcal{G}'$ and for computing edge and heuristic costs. The representative state is the state closest to the average location of all states in the region.

This is illustrated on the right side of Figure 1. There is now one vertex in $\mathcal{G}'$ for each region of different cost in a sector. Edges can be found within a sector between regions of different costs, and between regions in different sectors.

The cost of an edge in $\mathcal{G}'$ is defined in a similar manner to the edge costs in $\mathcal{G}$. The length of an edge is the distance between the two representative states, and the cost of an edge takes into account the terrain type for the two representative states by splitting the cost between them. This may be inaccurate, because in $\mathcal{G}$ the regions may not be uniformly sized. If this is too inaccurate, it is possible to compute edge costs directly, something that will be discussed later in the paper.

The heuristic in $\mathcal{G}'$ is the heuristic distance between the two states, ignoring the terrain costs. This is because there may be a very small edge between the start state and a neighbor which then leads to the least-cost terrain type for the remainder of the path. This heuristic is both simple to compute, but is also the same in both the forward and backwards directions, ensuring that bidirectional search can be used.

To facilitate changes to the terrain, the DTA stores each sector independently, with a data structure containing the terrain type of each state in the sector, a list of regions in the sector, and a list of outgoing edges between sectors. When the map changes and the regions inside a sector are altered,

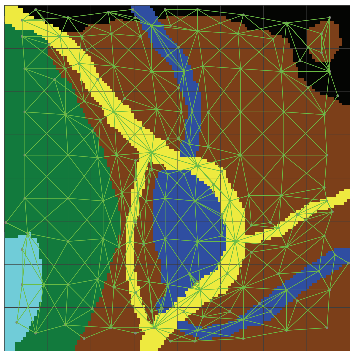

Figure 2: The dynamic terrain abstraction on a larger map with five different terrain types.

the regions in the sector must be re-computed and the edges between neighboring regions need to be re-built.

A larger example of the DTA abstraction is shown in Figure 2. In this figure there are 8x8 sectors overlaying the map, with different regions for each sector. There are 6 types of terrain, each shown in a different color, with the edges in the DTA drawn in green. Previous abstraction methods would build the abstraction as a uniform grid, since all cells are connected within each sector. Thus, there would be a single region in each sector. Looking at the upper-left sector, there are three terrain types in this sector, a road (in yellow), trees (in green), and a valley (in black).Within the sector, there are nodes for each of these terrain types as well as edges within the region between the road and the trees and between the road and the valley.

Using this abstraction, if it is desirable for a character to walk along the road, the road can be given the default terrain cost (1), and all other terrain can be given a higher cost. Because there are abstract regions (nodes in $\mathcal{G}'$) all along the road, it is easy to plan and stay inside these regions, something that cannot be done without explicit representation of terrain types. Thus, even if the road takes a more circuitous route, characters will automatically stay on the road when planning, and not accidentally try to cut corners through bits of forest or lakes. The exact details of how this work depend on decisions of how refinement is performed, which has been discussed elsewhere (Sturtevant 2007).

One key feature of the DTA is that no additional memory is required to support different cost functions for each character in the game. The abstract edge costs are estimates based on the weight of each terrain type and the length of abstract edges. Thus, they can be dynamically modified at runtime to achieve any desired pathfinding behavior.

4.1 Implementation

Regions are built for the abstraction by doing a breadth-first search (BFS) from each unlabeled state in a sector. This BFS is limited to stay within states of the same terrain type in the sector, and states are labeled with their region id during the BFS. After each state's region is labeled, one further iteration through the sector looks at the neighbors of each cell to identify if they are in a different sector/region. If they are, an edge is added between them.

When the map is changed, the low-level terrain type for a particular cell is modified. If the modification is local, meaning that the change is not on the edge of a sector, and all neighbors of the changed cell are still connected through neighbors with the same region type, then no further work is needed. But, if the broader connectivity of a region is changed, then the sector is rebuilt and edges are re-computed. A sector is re-built by re-computing regions, as described above. When a sector is rebuilt, edges are re-computed both in that sector and in the neighboring sectors, as their edges may be influenced by the change.

Given a pathfinding request between a start and goal state, the DTA must first localize the start and goal state onto the $\mathcal{G}'$. This can either be done by storing the region explicitly for each state or by doing a BFS within the region to find the representative state which identifies the region. Once the start and goal are localized, a search can be performed in $\mathcal{G}'$ to find an abstract path between the abstract start and goal. This path must then be refined in order to return a path between the actual start and goal states. The refinement process uses each successive representative state as a subgoal along the path. So, if an abstract path $\{v_0, v_1, v_2, v_3)$ is found, it can be refined in three steps. First, by planning between start and v_1, and then by planning from v_1 to v_2. Finally, a path is found from v_2 to the goal. Note that the refinement process skips the first and last states along the abstract path, since these states are in the same regions as the start and goal respectively. Note when using the DTA, it is possible for an agent to start following a path as soon as the first segment has been refined, instead of waiting for a complete path (Sturtevant and Buro 2005). This works because, when using abstraction approaches, a path in $\mathcal{G}$ is guaranteed to exist once a path in $\mathcal{G}'$ has been found (Bulitko et al. 2007).

4.2 Parameterizations and Performance

There are many enhancements which can be used with the DTA and parameterizations of the search which will impact performance. We describe some of these enhancements here. A select subset of enhancements will be included in the experimental results.

First, many different pathfinding algorithms can be used both in $\mathcal{G}'$ and in $\mathcal{G}$. Candidates we consider here include A*, which finds optimal paths, NBS, a bidirectional algorithm which finds optimal paths, and Weighted A*, which finds paths which are w-optimal for some fixed parameter w. These algorithms can be used for planning directly in $\mathcal{G}$, or with the DTA both for planning in $\mathcal{G}'$ and for refining path segments in $\mathcal{G}$. In the experimental results we will show that Weighted A* can find paths quickly in $\mathcal{G}$, but that the quality of those paths is poor. Thus, the DTA has better overall performance.

Second, better heuristics can be built in either $\mathcal{G}$ or $\mathcal{G}'$. Example approaches include compressing the full all-pairs shortest-path data (Botea and Harabor 2013), building differential heuristics (Sturtevant et al. 2009), or building inexpensive Euclidean embeddings (Cohen et al. 2018) for use as heuristics. Each of these approaches trade computation and memory for faster search. These are interesting approaches for future work, as they can be applied to search both in $\mathcal{G}$ and $\mathcal{G}'$. But, until more data is available regarding how often maps change in practice, it is difficult to evaluate how much time it is worth spending doing pre-computation on maps. Another factor which will determine whether it is worth building heuristics is the number of different character types that would need heuristics. If there are only 1-2 character types in the world, or if the majority of pathfinding requests are made by a single character type, then the overhead of building heuristics may be effectively amortized across pathfinding requests. With more character types it could be important to either only build heuristic for a subset of characters, or share heuristics between character types (making them less accurate) in order to reduce the precomputation time or the memory overhead.

Third, the quality of paths can be improved by computing more accurate edge costs, and the speed of pathfinding can be increased by caching the result of edge refinement. The edge costs in $\mathcal{G}'$ are just estimates, but they can be computed exactly with a low-level search. This would improve distance estimates when an edge is not evenly split between two different terrain types. If paths are also stored, then the amount of search needed to re-construct the final path can be greatly reduced, although this information could be invalidated when the map changes. Either of these approaches increases the total memory needed to store the abstraction, and, as with heuristics, may require many copies of the pre-computed data as the number of character types increases.

More broadly, there are broad domain-specific questions about the design of maps and weights which will influence performance. For instance, the number of terrain types and the relative weights of each terrain type will determine the suitability of the DTA approach. The approach would not be effective on a map where each cell has a random terrain type, as no abstraction would actually occur in practice. Similarly, the runtime distribution of queries will also have a large impact on performance. If characters always walk between terrain types that are inexpensive for their character profiles, then the cost of pathfinding will be significantly reduced over the case where characters must walk through expensive terrain to reach their destinations.

5 Experimental Results

Experimental results are used to illustrate the potential of the DTA, and to compare it to alternate approaches. The source code and maps used for the experiments in publicly available[1]. Experiments were run on a laptop with a 2.3GHz Intel Core i7 processor running macOS Mojave.

Most maps in existing benchmark sets (Sturtevant 2012) do not have significantly different terrain types across maps. So, new maps were needed for our testing purposes. We used a public map generator[2] to generate 20 maps. 2048x2048 screen captures of these maps were saved for use in the experiments, with the intent to use pixel colors for terrain types in an 8-connected grid map. Although maps appear to have a limited number of distinct terrain types, the actual image

[1]https://github.com/nathansttt/hog2/

[2]https://www.redblobgames.com/maps/mapgen2/embed.html

Abstract	Regions	Edges	Memory	Repair Time
None	-	-	4,194,304	-
32	8,195	59,896	5,259,154	0.31855
16	24,112	183,132	7,541,755	0.08675
8	79,774	621,017	15,783,227	0.02345

Table 1: Memory overhead and overhead of dynamic repairs to the DWA.

data was far from uniform and contained pixel-level smoothing. Thus, colors were masked to create at most 64 terrain types, but in practice less than 16 terrain types resulted from each map. A final processing step merged regions with less than 80 pixels in their neighbors. These were then converted to text files and used as grid maps for experimentation. 250 random start and goal locations were chosen on each map (5,000 problems total over 20 maps) and used for the experiments reported here.

5.1 Memory Usage and Repair Cost

First, we look at the memory required to store the DTA and to make change to the terrain at runtime. These results are shown in Table 1. In this experiment we vary the size of the sector, and measure the number of regions and edges in the graph, the total memory required to store the graph, and the average time to repair/rebuild a single sector (in ms). To measure the repair time, we rebuilt each sector in turn (including re-computing the edges between the neighbors), and then divided this time by the number of sectors to get the average repair time per sector. This is the cost of a full rebuild, not an internal modification in a sector, which can be much cheaper. This was repeated for each of the 20 maps.

The default map uses 1 byte per cell to store the terrain type; on 2048x2048 maps this requires 4MB of storage. The 32x32 DTA required 5MB of memory, with approximately 1MB of overhead for the abstraction and 4MB for the map, while the 8x8 DTA required 15MB of memory, with 11MB for the abstraction. The advantage of the smaller sector size is that the sector contains fewer cells and thus be repaired more quickly when the maps change. Changes to the 8x8 DTA average 0.023ms per repair versus 0.319ms per repair for the 32x32 DTA. Because repairing a sector is a local operation, the time required is independent of the map size, unlike rebuilding pre-computed heuristics.

5.2 Pathfinding Cost

Next we study the cost of finding partial and full paths when solving each individual problem and look at the path quality of different algorithms. The results for these experiments are found in Table 2. The first column is the algorithm being tested. The second column is the total node expansions required to build a complete path. The third column is the node expansions required to return the first segment of walkable path. The fourth column is the average path length, and the last column is the average time to find a complete path. In this table, the terrain costs for each map are randomized with costs between 1 and 4 at intervals of 0.2.

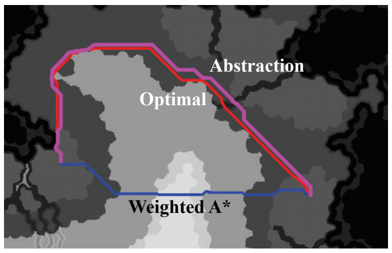

Figure 3: The optimal path as compared to the path from the abstraction and weighted A*.

The first two entries in the table, A* and NBS, are finding optimal paths. A* searches unidirectionally, while NBS searches bidirectionally. NBS's data structures are slightly more expensive per node expansions. With the cost range we selected, both algorithms perform similar numbers of node expansions. If the range of costs is reduced (e.g. to a maximum of 2), then A* will outperform NBS. However, if the range of costs is increased, A*'s performance will continue to degrade relative to NBS. There are only 4 million states in each map, so the optimal algorithms are searching one quarter of the map per problem instance. This illustrates why terrain costs are not often used in practice – they effectively weaken the heuristic and increase the cost of pathfinding.

Weighted A* (WA*) can be used to find suboptimal solutions and reduce the total pathfinding cost. With a weight of 4, WA*(4) find solutions that are 37% longer than optimal, while doing 238x less work than NBS. Thus, it might seem that WA* solves the problem of using terrain costs. However, this is not the case. In the maps used here, it is always possible to walk straight from the start to the goal ignoring terrain types. It ends up that WA*, with large enough weights, is just returning the path directly from the start to the goal, ignoring the terrain. This is illustrated in Figure 3 where the task is to find a path from the left to the right side of the portion of the map shown. In this example the lighter terrain is more expensive to cross and the darker terrain is less expensive. WA*(4) finds a path that has near-minimal distance, but suboptimal cost. The cost of the path found is 1879 while expanding 6657 states. NBS finds the optimal path with cost 1136 while expanding 300k nodes, while DTA find a path with cost 1295 while expanding 8744 nodes.

Algorithm	Nodes (all)	Nodes (first)	Path Len	Time (ms)
A*	1,089,811	-	2,406	625.9
NBS	1,044,995	-	2,406	879.3
WA*(2)	349,865	-	2,567	180.7
WA*(3)	84,413	-	2,907	39.0
WA*(4)	4,392	-	3,298	2.6
DTA(8)[A*]	28,907	22,228	2,456	25.4
DTA(8)[NBS]	27,702	21,024		26.9
DTA(16)[A*]	19,366	7,409	2,495	11.4
DTA(16)[NBS]	18,923	6,971	2,495	10.8
DTA(32)[A*]	26,016	4,031	2,560	10.8
DTA(32)[NBS]	25,802	3,850	2,560	10.8
DTA(32)[w1.2]	21,700	3,837	2,561	9.6

Table 2: Node expansions and path quality.

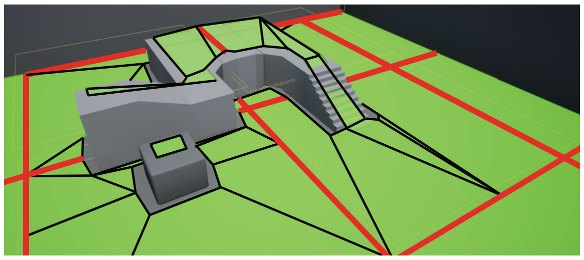

Figure 4: The navmesh generated by Recast which is already divided into large sectors (red).

While the DTA does slightly more work than WA*(4) on this problem, it is finding a path that does a much better job of respecting the actual terrain costs in the map. Returning to Table 2 we see that only WA*(2) has similar path quality to DTA, but it expands at least 10x more nodes that the DTA approach. Furthermore, with the DTA an agent can begin moving once the first path segment is computed, meaning that delays waiting for pathfinding requests can be further reduced. Note that we didn't experiment with previous abstraction techniques here, as they would either produce paths similar to WA*(4) that ignore terrain costs or, perhaps worse, would be forced to travel between poorly chosen region centers, resulting in equally poor quality paths.

In the bottom of Table 2 we report several variants of the DTA approach. As we increase the sector size the number of node expansions required before the first path segment is complete is reduced. This is because the size of $\mathcal{G}'$ is smaller, reducing the cost of the search. But, the larger abstraction results in slightly longer paths. The paths returned using the 8x8 DTA are only 2% suboptimal on average, while the 32x32 DTA is 6% suboptimal. With the 8x8 and 16x16 DTA, A* is used to find the abstract path and to refine the path segments. With the 32x32 DTA, we also consider using NBS to find the abstract path. This results in a very small improvement over A*, but using a broader range of weights would likely result in a larger performance gain for NBS. The other modification we made was to use WA*(1.2) during the refinement step instead of A*. This has almost no impact on path quality, but reduced the average node expansions to 21,700 for the full path and 3,837 for the partial path. This is still more node expansions than we would prefer (a few hundred would be better), but these gains make it computationally feasible to consider a broad range terrain costs during pathfinding. In alternate representations, such as polygon navmeshes, there are typically vertices than what is found in a grid map.

5.3 Unreal Implementation

Navigation meshes, commonly found in modern video games, are well suited for use with the DTA. The Unreal Engine 4 uses Recast for its navigation mesh and can be easily abstracted into dynamic regions. Recast generates a map?s navigation mesh by creating polygons within tiles throughout the map. This can be seen in Figure 4, where polygons are clipped to the edge of square tiles in the map. The size of the tiles is a user-controlled parameter.

These tiles align with the sectors used in DTA for building abstract regions. We can separate the polygons within a tile into groups with a simple connected component labeling algorithm. We group all connected polygons that have same navigation type within a tile to form abstract regions. Different navigation types can be implemented using navmesh modifiers which allow polygons to be marked with types representing modified pathfinding costs. We use the mean location of the polygons within an abstract region for its location. After creating the abstract regions, we can check which polygons contain neighbors to polygons in different abstract regions allowing edges to be built between the abstract regions.

Maintaining the abstract graph as the underlying navigation mesh changes is inexpensive. When a tile is updated in the Recast navigation mesh the corresponding sector is updated with the same repair operation used in grids. This is done by discarding all the abstract nodes within that sector and rebuilding the abstract nodes. After the abstract nodes are rebuilt edges can again be added by checking neighboring polygons belonging to different abstract regions.

In initial experiments with DTA on a relatively simple 1km by 1km open world map with a tile size of 2000, DTA reduced 102,193 polygons to 7,338 abstract nodes with 13,298 edges. This implementation is continuing to be developed for use in an unannounced game, with a plan to freely release the DTA implementation. We tested our implementation of the DTA on this map against Unreal Engine 4's out of the box pathfinding by running 5,200 queries ranging from 1,685 unreal units (UU) to 138,383 UU. The average time for Unreal to solve these problems was 5.64 (95% confidence interval of $\pm$0.12ms). With the DTA this was reduced to 1.68 ($\pm$0.03ms). However, DTA?s path length was 82,371 UU versus 77,722 UU for Unreal. Our initial experiments with the DTA have been very promising for its use in modern video game pathfinding.

6 Conclusions

This paper has shown how to modify abstraction techniques to account for terrain costs. A new abstraction, DTA, is designed for dynamic maps with terrain costs. The abstraction approach significantly outperforms exiting techniques, able to find paths that are within 2-6% of optimal on average, while expanding far fewer nodes in practice.

Given the work in this paper that establishes the feasibility of planning with complex terrain costs, future work will look to understand the real-world conditions under which planning requests occur, including the magnitude and frequency of changes to the terrain, the frequency of pathfinding requests, the number of different creature types, the rate at which player preferences change, and other information that be used to tune the parameters of DTA to maximize overall performance.

References

Alain, B. 2018. Hierarchical dynamic pathfinding for large voxel worlds. *Game Developers Conference*.

Aleksandrov, L.; Maheshwari, A.; and Sack, J.-R. 2000. Approximation algorithms for geometric shortest path problems. In *Annual ACM symposium on Theory of Computing*, 286–295.

Barker, J. K., and Korf, R. E. 2015. Limitations of front-to-end bidirectional heuristic search. In *AAAI Conference on Artificial Intelligence*, 1086–1092.

Botea, A., and Harabor, D. 2013. Path planning with compressed all-pairs shortest paths data. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1(1):7–28.

Brewer, D. 2017. 3d flight navigation using sparse voxel octrees. In *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. CRC Press.

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 30:51–100.

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Cohen, L.; Uras, T.; Jahangiri, S.; Arunasalam, A.; Koenig, S.; and Kumar, T. S. 2018. The fastmap algorithm for shortest path computations. *International Joint Conference on Artificial Intelligence (IJCAI)* 1427–1433.

Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research* 39:533–579.

Ferguson, D. I., and Stentz, A. 2006. Multi-resolution field D*. In *International Conference on Intelligent Autonomous Systems*.

Funke, S., and Storandt, S. 2013. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 41–54. SIAM.

Harabor, D., and Botea, A. 2008. Hierarchical path planning for multi-size agents in heterogeneous environments. In *2008 IEEE Symposium On Computational Intelligence and Games*, 258–265. IEEE.

Harabor, D. D., and Grastien, A. 2011. Online graph pruning for pathfinding on grid maps. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Hart, P.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics* 4:100–107.

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1-2):321–361.

Kapadia, M.; Ninomiya, K.; Shoulson, A.; Garcia, F.; and Badler, N. 2013. Constraint-aware navigation in dynamic environments. In *Motion on Games*, 111–120.

Mata, C. S., and Mitchell, J. S. B. 1997. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *IN PROC. 13TH ANNU. ACM SYMPOS. COMPUT. GEOM*, 264–273. ACM Press.

Mitchell, J. S., and Papadimitriou, C. H. 1991. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM* 38(1):18–73.

Mould, D., and Horsch, M. C. 2004. An hierarchical terrain representation for approximately shortest paths. In *Pacific Rim International Conference on Artificial Intelligence*, 104–113. Springer.

Ninomiya, K.; Kapadia, M.; Shoulson, A.; Garcia, F.; and Badler, N. 2015. Planning approaches to constraint-aware navigation in dynamic environments. *Computer Animation and Virtual Worlds* 26(2):119–139.

Pelechano, N., and Fuentes, C. 2016. Hierarchical pathfinding for navigation meshes (hna*). *Computers & Graphics* 59:68–78.

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *National Conference on Artificial Intelligence (AAAI)*, 47–52.

Sturtevant, N. R., and Geisberger, R. 2010. A comparison of high-level approaches for speeding up pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Sturtevant, N., and Jansen, R. 2007. An analysis of map-based abstraction and refinement. *Symposium on Abstraction, Reformulation and Approximation (SARA)* 344–358.

Sturtevant, N. R.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. *International Joint Conference on Artificial Intelligence (IJCAI)* 609–614.

Sturtevant, N. R.; Traish, J.; Tulip, J.; Uras, T.; Koenig, S.; Strasser, B.; Botea, A.; Harabor, D.; and Rabin, S. 2015. The grid-based path planning competition: 2014 entries and results. In *Eighth Annual Symposium on Combinatorial Search*, 241–251.

Sturtevant, N. R. 2007. Memory-efficient abstractions for pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 31–36.

Sturtevant, N. 2011. A sparse grid representation for dynamic three-dimensional worlds. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 73–78.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

Sturtevant, N. 2013. Incorporating human relationships into path planning. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 177–183.

Wardhana, N. M.; Johan, H.; and Seah, H. S. 2013. Enhanced waypoint graph for surface and volumetric path planning in virtual worlds. *The Visual Computer* 29(10):1051–1062.