Hierarchical path planning for multi-size agents in heterogenous environments

Daniel Harabor and Adi Botea

National ICT Australia and The Australian National University

f daniel.harabor j adi.botea g at nicta.com.au

Abstract

In this paper we present new techniques for the automated construction of state space representations of complex multi-terrain grid maps with minimal information loss. Our ap-proach involves the use of graph annotations to record the amount of maximal traversable space at each location on a map. We couple this with a cluster-based hierarchical ab-straction technique to build highly compact yet complete representations of the original problem. We further out-line the design of a new planner, Annotated Hierarchical A\* (AHA\*), and demonstrate how a single abstract graph can be used to plan for many different agents, including differ-ent sizes and terrain traversal capabilities. Our experimental analysis shows that AHA\* is able to generate near-optimal solutions to a wide range of problems while maintaining an exponential reduction in comparative effort over low-level search. Meanwhile, our abstraction technique is able to gen-erate approximate representations of large problem-spaces with complex topographies using just a fraction of the stor-age space required by the original grid map.

Introduction

Single-agent path planning is a well known and extensively studied problem in computer science. It has many appli-cations such as logistics, robotics, and more recently, com-puter games. Despite the large amount of progress that has been made in this area, to date, very little work has focused specifically on addressing planning for diverse-size agents in heterogenous terrain environments.

The problem is interesting because such diversity in-troduces much additional complexity when solving route-finding problems. Modern real-time strategy or role-playing games for example often feature a wide array of units of dif-fering shapes and abilities that must contend with navigating across environments with complex topographical features – many terrains, different elevations etc. Thus, a route which might be valid for an infantry-solider may not be valid for a heavily armoured tank. Likewise, a car and an off-road vehi-cle may be similar in size and shape but the paths preferred by each one could differ greatly.

Unfortunately, the majority of current path planners, in-cluding recent hierarchical planners ((Botea, Muller,¨ and Schaeffer 2004), (Sturtevant and Buro 2005), (Demyen and Buro 2006), (Geraerts and Overmars 2007)), only perform

well under certain ideal conditions. They assume, for ex-ample, that all agents are equally capable of reaching most areas on a given map and any terrain which is not traversable by one agent is not traversable by any. Further assumptions are often made about the size of each respective agent; a path computed for one is equally valid for any other because all agents are typically of uniform size. Such assumptions limit the applicability of these techniques to solving a very nar-row set of problems: homogenous agents in a homogenous environment.

We address the opposite case and show how efficient solu-tions can be calculated in situations where both the agent’s size and terrain traversal capability are variable. Our method extends recent work emerging from the areas of robotics and computer games which has shown the effectiveness of using clearance annotations to measure obstacle distance at key lo-cations in the environment and using this information to help agents plan better paths.

We contribute in several ways: first, we introduce AHA\*, a new clearance-based hierarchical path planner; second, we show how to leverage clearance in order to produce compact yet information rich search abstractions; third, we provide a detailed empirical analysis of our new technique on a wide range of problems involving multi-size agents in heteroge-nous multi-terrain environments.

The rest of this paper is organised as such: first, we cover existing work in the area of hierarchical path planning and multi-size agent search. We then define the problem and describe our map annotation approach before showing how to adapt A\* to plan for a range of agents of different sizes and capabilities. We go on to detail a new map abstraction technique that leverages map annotations and characterise the worst-case performance. In the final sections we intro-duce our hierarchical planner, AHA\*, and provide a detailed analysis of its performance before concluding.

Related Work

A very effective method for the efficient computation of path planning solutions is to make the original problem more tractable by creating and searching within a smaller approx-imate abstract space. Abstraction factors a search problem into many smaller problems and thus allows agents to reason about pathfinding strategies in terms of macro operations. This is known as hierarchical path planning.

Two recent hierarchial path planners relevant to our work are described in (Botea, Muller,¨ and Schaeffer 2004) and (Sturtevant and Buro 2005). The first of these, HPA\*, builds an abstract search graph by dividing the environment into square clusters connected by entrances. Planning involves inserting the low-level start and goal nodes into the abstract graph and finding the shortest path between them.

The second algorithm, PRA\*, builds a multi-level search-space by abstracting cliques of nodes; the result is to nar-row the search space in the original problem to a “win-dow” of nodes along the optimal shortest-path. Both HPA\* and PRA\* are focused on solving planning problems for homogenous agents in homogenous-terrain environments and hence are not complete when either of these variables change. Our technique is similar to HPA\* but we extend that work to solve a wider range of problems.

In robotics, force potentials help autonomous robots find collision-free paths through an environment. The basic intu-tion is that a robot is attracted to the far-away goal and re-pulsed away from obstacles as it nears them. A well known method for potential-based path planning is the Brushfire algorithm (Latombe 1991), which proceeds by annotating each tile in a grid-map with the distance to the nearest ob-stacle. This embedded information allows the robot to cal-culate repulsive potentials and makes it possible to plan using a gradient descent strategy. Brushfire is similar to AHA\* in that the annotations it produces allow an agent to know something about its proximity to a nearby obsta-cle. AHA\* differs by explicitly calculating the maximal size of traversable space at each location on the map. Further-more, unlike Brushfire, AHA\* does not suffer from incom-pleteness which can occur when repulsive forces cancel each other out and lead the robot into a local minimum.

The Corridor Map Method (CMM) (Geraerts and Over-mars 2007) is a recently introduced path planner able to answer queries for multi-size agents by using a probabilis-tic roadmap to represent map connectivity. The roadmap (or backbone path) is comprised of nodes which are anno-tated with clearance information that indicates the radius of a maximally sized bounding sphere before an obstacle is en-countered. Nodes are placed on the roadmap by creating Voronoi regions to split the map and identify locations that maximise local distance from fixed obstacles.

Like CMM, AHA\* calculates the amount of traversable space at a given location but our approach is adapted to grid environments, which are simpler to create than roadmaps and more commonly found in a range of applications. An-other key difference is that we allow fine-grain control over the size of the abstract graph; CMM abstractions have a fixed size. Finally, we deal with multi-terrain cases making our method more information rich.

Representing an environment using navigation-meshes is increasingly popular in the literature. Two recent planners in this category are Triangulation A\* and Triangulation Reduc-tion A\* (Demyen and Buro 2006). TA\* makes use of a tech-nique known as Delaunay triangulation to build a polygonal representation of the environment. This results in an undi-rected graph connected by constrained and unconstrained edges; the former being traversable and the latter not. TRA\*

is an extension of this approach that abstracts the trian-gle mesh into a structure resembling a roadmap. Like our method, both TA\* and TRA\* are able to answer path queries for multi-size agents. The abstraction approaches used by TA\* and TRA\* however are very distinctly different from our work. Where we use a simple division of the environ-ment into square clusters, their approach aims to maximise triangle size. We also handle additional terrain requirements while both TA\* and TRA\* assume a homogenous environ-ment.

Problem Definition

A gridmap is a structure composed of square cells, of unit size, each of which represent a unique area in the environ-ment. Each grid cell is an octile, connected to k neighbours, where 0 k 8.

Each octile, t, is associated with a particular terrain type, terrain(t) 2 T where T is the set of all possible terrains and there are r = jT j : r 1 possible terrains. Each octile is either blocked or traversable. Blocked octiles are called hard obstacles, since no agent can occupy them.

Each gridmap is representable as a graph, G = (V; E) where each traversable tile generates one node v 2 V and each cell adjacency is represented by an edge e 2 E.

An agent is any entity attempting to move across a grid environment. Every agent is square in shape and has a size s 1 : s 2 S where S is the set of finite sizes agents traversing across the gridmap may take. While stationary or moving, each agent occupies s2 octiles which together correspond to its current location.

Agents can move in any of the four cardinal directions. Diagonal moves are allowed only if there exists an equiva-lent two-step move using the cardinal directions.

Every agent has a terrain traversal capability, c 2 C, where c comprises a non-empty subset of terrains. An agent can never occupy a tile whose terrain type is not included in its capability.

A soft obstacle is a tile which is not traversable by a spe-cific agent because it lacks the correct capability or its size is larger than the associated clearance value of the tile, as defined below.

A clearance value is an obstacle-distance metric associ-ated with a particular tile in the grid environment. Each clearance value measures the maximal size of an agent at a given location without intersecting any obstacle in the envi-ronment. A tile can have several clearance values associated with it, one for each capability.

A problem instance is defined as a pair of locations, a start and goal, associated with an agent. A problem is valid if at least one path exists between the locations comprising only tiles traversable by the agent.

Computing Clearance Value Annotations

On a grid map, a clearance value is perhaps best explained as representing the length or width of a square that begins at some octile being evaluated and is expanded symmetrically to the right and down until it intersects an obstacle. To make our ideas more concrete we will use as a running example

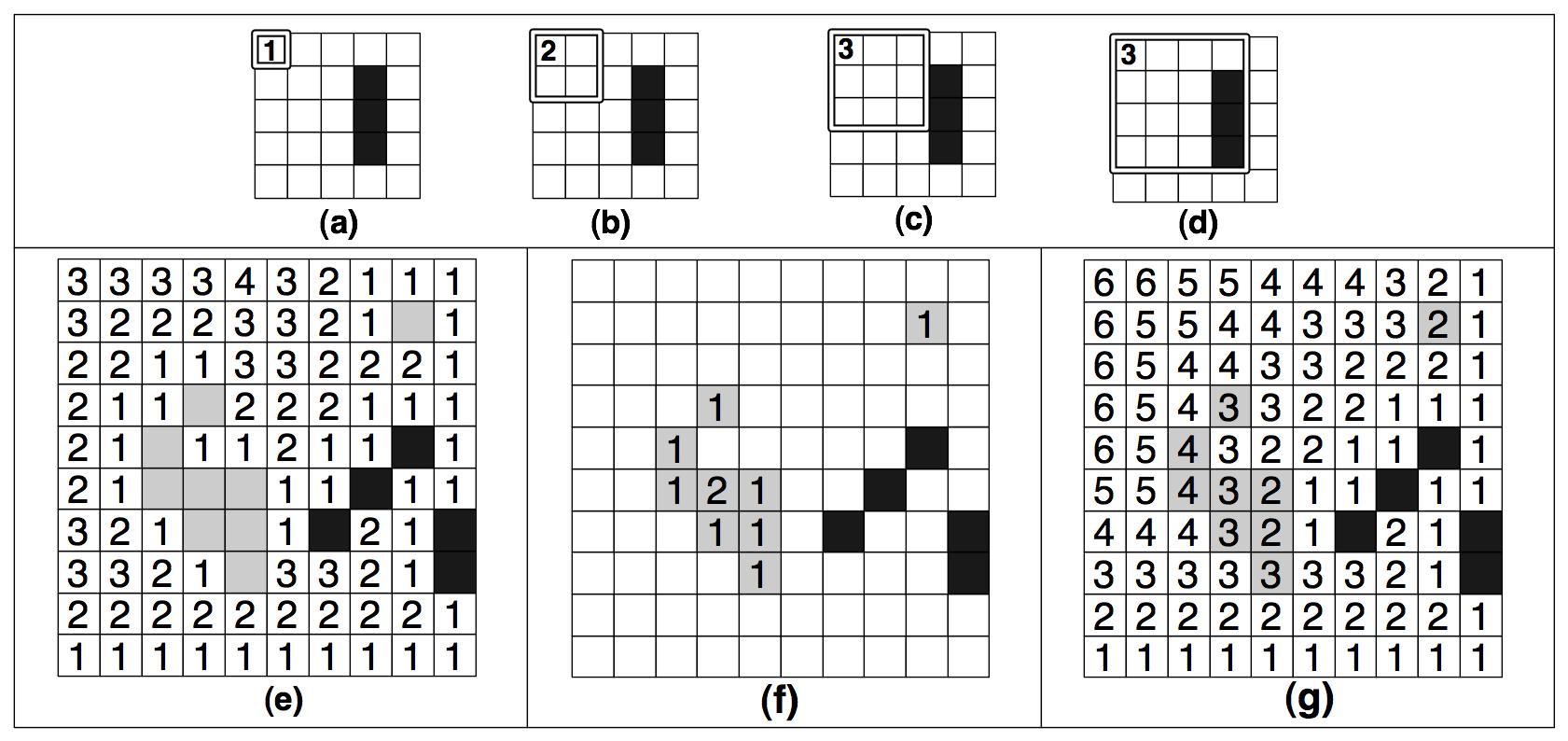
a simple environment featuring two terrain types: Ground (represented as white tiles) and Trees (represented as grey tiles). To distinguish traversable tiles from non-traversable tiles we will colour hard obstacles black. The set of capa-bilities, C, required to traverse such a map is thus defined as

* = ffGroundg; fT reesg; fGround \_ T reesgg. We will work with agents of two sizes traversing across this environ-ment and thus let S = f1; 2g.

Figure 1 (a) to (d) illustrates how clearance can be com-puted with an iterative procedure in an environment as de-scribed above. In Figure 1(a) the clearance square for the highlighted traversable target tile is initialised to 1. Sub-sequent iterations (Figures 1(b)-(c)) extend the square and increment the clearance. The process continues until the square contains an obstacle (Figure 1(d)) or extends beyond a map boundary at which point we terminate and do not in-crement clearance any further.

In Figure 1(e) we show the resultant clearance values for the single-terrain fGroundg capability on a toy map exam-ple (note that we omit zero-value clearances). Similarly, Fig-ure 1(f) and Figure 1(g) show the clearance values associ-ated with the fT reesg and fGround \_ T reesg capabilities respectively.

Figure 1: (a)-(d) Computing clearance. (e)-(g) Clearance values for different capabilities.



Once a clearance value is derived we store it in memory and repeat the entire procedure for each capability c 2 C. The algorithm terminates when all octiles t 2 gridmap have been considered. The worst-case space complexity associ-ated with computing clearance values in this fashion is thus characterised by:

Lemma 1. Let CV be the set of all clearance values re-quired to annotate an octile gridmap with r terrains. Fur-ther, let G = (V; E) be a graph representing the gridmap where VHO 2 V is the set of hard obstacles. Then,

jCV j = (jV j jVHOj) 2r 1

Proof. For a node to be traversable for some capability, the capability must include the node’s terrain type. There are 2r capabilities but each terrain type is included in only 2r 1 of these. There are jV j nodes in total to represent the envi-ronment, and we avoid storing any clearance values for all nodes in VHO. 

The result from Lemma 1 is an upper bound; if no agent has a given capability c there is no need to store the corre-sponding clearances. Despite this observation, the associ-

ated exponential growth function suggests that it is imprac-tical to store every clearance value as there are (2r) per node. Fortunately, clearance values can be computed on-demand with little effort. In particular, calculating clearance for any agent a of size s 2 S only requires building a clear-ance square of maximum area s2 octiles. We present such an approach in Algorithm 1.

Algorithm 1 Calculate Octile Clearance Value

Require: t 2 gridmap and c 2 C and s 2 S square t

cv 0

while traversable(square; c) ^ area(square) s2 do cv cv + 1

square expand(square)

return cv

The key advantage of calculating clearance is that we are able to plan for both large and small agents using a fixed size grid. We achieve this by mapping our extended prob-lem into a classical problem with only two types of tiles (traversable and blocked) and reducing the problem to the case of a small-size agent that occupies the upper-left corner of the area required by the original, large-size agent.

Theorem 2. Given an annotated grid map, any search prob-lem involving an agent of arbitrary size and capability can be reduced into a small-agent search problem, where the size of the small agent is one tile and the capability of the agent is one terrain.

Proof. A tile t is only traversable by an agent a if t has a clearance value cvt associated with the agent’s capability ca which is at least as large as the size of the agent, sa.

t(ca) = cvt sa : terrain(t) ca 2 C; sa 2 S (1)

If equation 1 holds, it must be the case that the terrain type of every tile in the clearance square used to compute cvt is included in ca and hence traversable for the agent. Thus, the agent is able to navigate across a map by only considering the traversal requirements of a single node. Since each tile being evaluated is either traversable or not this is equivalent to solving a single-terrain problem. 

This is a useful result because it indicates that we can ap-ply abstraction techniques from classical path planning to answer much more complex queries involving a wide range of terrain type and agent-size variables.

Annotated A\*

Low-level planning for diverse sets of agents using clearance values is a straightforward application of the ideas thus far. We use a variation on the A\* algorithm (Hart, Nilsson, and Raphael 1968) to compute an optimal shortest path between a start and goal node. Our approach differs from standard A\* by requiring two additional parameters for each query: the agent’s size and capability. This allows us to map any query into an instance of small-agent search as shown earlier by using the parameters to evaluate nodes before they are added

to A\*’s open list. We term the resultant algorithm Annotated A\* (AA\* for short).

Cluster-based Map Abstraction

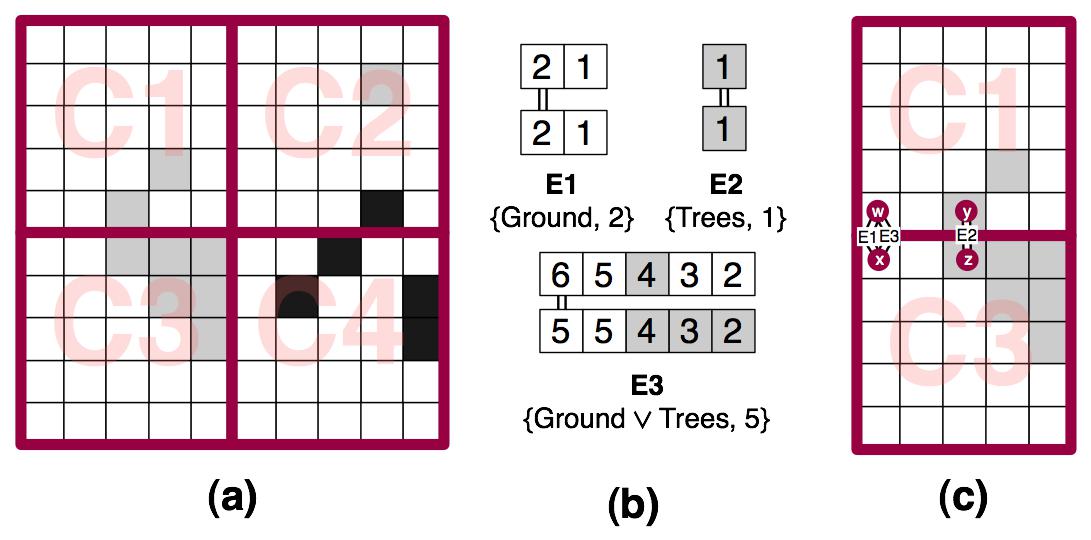
AA\* is sufficient for low-level planning on the original gridmap but inefficient for large problem sizes; we would prefer to express a more general strategy using macro-operators. Our result from theorem 2 is key to the spatial abstraction described in this section.

We extend the process in (Botea, Muller,¨ and Schaeffer 2004) which involves dividing a grid map into fixed-size square sections called clusters. Figure 2(a) shows the re-sult of this decomposition approach; we use clusters of size 5 to split our toy map into 4 adjacent sections.

In the original work entrances are defined as obstacle-free transition areas of maximal size that exist along the border between two adjacent clusters. Each entrance has one or two transition points (depending on its size) which are rep-resented in the abstract graph by a pair of nodes connected with an undirected inter-edge of weight 1.0. We use a sim-ilar approach but require as a parameter C, the set of all capabilities, and thus attempt to identify entrances for each

* 2 C.

Figure 2: Building clusters and identifying entrances



We start at the first pair of traversable tiles along the adja-cent border area and extend each entrance until one of three termination conditions occurs: the end of the border area is reached, an obstacle is detected or the clearance value of nodes along the border area in either cluster begins to in-crease. The last condition is important to preserve represen-tational completeness for large agents in cases where a clus-ter is partially divided by an obstacle (such as a wall) near the border. By leveraging clearance we are able to reason about the presence of such obstacles and build a new en-trance each time we detect the amount of traversable space inside either cluster is increasing.

Once an entrance is found, we choose as the transition point the first pair of adjacent nodes in each cluster which

maximise clearance for c. This latter metric, cvinter is com-puted by taking the minimum clearance among each pair of

adjacent nodes in the entrance area and selecting the largest value from the set. Thus, we add a new edge to the graph,

einter and annotate it with a single capability and corre-sponding clearance value, einter(c) = cvinter. The algo-rithm repeats for each c 2 C and terminates when all adja-

cent clusters have been considered. This ensures we identify all possible entrances for each available capability.

In Figure 2(b) we present three entrances identified by scanning the border between clusters C1 and C3. Entrances E1 and E2, each of which span only part of the border area, are discovered using the fGroundg and fT reesg capabil-ities respectively. E3 meanwhile, which spans the whole border area, is discovered using the fGround \_ T reesg ca-pability. The connected tiles represent the locations of the subsequent transition points; the final result is shown in Fig-ure 2(c). Note that E1 and E3 are incident on the same pair of nodes in the abstract graph. This is due to our strategy of actively attempting to re-use any existing nodes from the abstract graph.

The final step in the decomposition involves attempting to add to the abstract graph a set of intra-edges for each pair of abstract nodes inside a cluster. We achieve this by running multiple AA\* searches 8(c; s) : c 2 C; s 2 S. Once a path

is found we annotate the new edge, eintra, with the capabil-ity and clearance parameters used by AA\* and set its weight

equal to the cost of the path. The algorithm terminates when all clusters have been considered.

We thus construct an abstract multi-graph in which each edge e is annotated with a single capability ce and associated clearance value cve. Each e 2 Eabs is traversable by an agent a iff:

e(ca) = cve sa : ce ca 2 C; sa 2 S

Where ca represents the capability of the agent and sa its size. We term the resultant abstraction initial and give the following lemmas to characterise its space complexity:

Lemma 3. Let Vabs represent the set of nodes in an abstract graph of a gridmap which is perfectly divisible into c c

clusters, each of size n n. Then, in the worst case, the total number of nodes is given by:

jVabsj = 4(2n 1) + (4c 4)(3n 2) + (c 1)2(4n 4)

Proof. Each transition point results in two nodes in the ab-stract graph. In the worst case the number of terrains r n. If there are no hard obstacles and every pair of nodes along the adjacent border between two clusters has different ter-rain type then there will be a maximal number of transition points. In this scenario, clusters in the middle of the map, of which there are (c 1)2, have 4 neighbours and each one contains 4n 4 nodes. Clusters on the perimeter of the map (excluding corners), of which there are 4c 4, have 3 neigh-bours and 3n 2 nodes. Corner clusters, of which there are 4, have 2 neighbours and each contains 2n 1 nodes. 

Lemma 4. Let Eabs(L) Eabs represent the set of intra-edges for a cluster L that contains x abstract nodes. Fur-

ther, let r be the total number of terrains found in the map and k the number of distinct terrain types found inside L. Then, the number of intra-edges required to connect all nodes in L is, in the worst case:

jEabs(L)j = jSj 2k 1 x(x 1)

2

Proof. For each pair of abstract nodes in a cluster and each size/capability combination, we compute at most one path of optimal length. From lemma 1 we know each node is

traversable at most by 2r 1 capabilities thus there must be at most jSj 2r 1 ways of covering 2 nodes. However, the number of terrains inside a cluster is governed by its size; only k r terrains may be found. From this, it follows that the upper-bound on the size of the set of edges covering each pair of nodes is in fact jSj 2k 1. In the worst case there will be a maximal number of edges between each pair of nodes

and there are x(x 1) such pairs in total per cluster.

2

Lemma 5. Let Einter Eabs represent the set of inter-edges in an abstract graph of a grid map. In the worst case,

the map is perfectly divisible into c c clusters, each of size n n and the number of inter-edges is given by

jEinterj = (2c2 2c) n(n 1)

2

Proof. We know from the proof of lemma 3 that in the worst case each tile along the border between two adjacent clus-ters is represented by a node in the abstract graph. If we count the number of adjacencies, avoiding duplication, we find there are 2c2 2c in total.

Each transition results in an inter-edge and there are n single-terrain transitions with clearance 1 per adjacency and some number of inter-edges to represent multi-terrain tran-sitions with larger clearances. By observation we can see that that each adjacency will produce [n single-terrain tran-sitions]...[1 n-terrain transition]. This recurrence relation

holds for the general sequence counting formula n(n 1)

2

The above results are interesting for several reasons. Firstly, lemma 3 shows that the number of nodes in the graph is a function of cluster-size. This suggests that by varying the dimensions of clusters we can trade a little performance (the time it takes to traverse a cluster) for memory (less ab-straction overhead). The results in lemma 4 and 5 seem to support this hypothesis. We see that the number of edges between nodes in the graph is mostly dependent on the com-plexity of the clusters in which they reside rather than ex-ponential in the number of capabilities. This is exciting be-cause it means that, despite having an exponential abstract edge growth function, we can directly control the size of the exponent! The cluster-based decomposition technique al-lows us to include as much or as little complexity in each cluster as we require.

Optimising Abstract Graph Size

As we have observed in lemmas 4 and 5 the initial abstrac-tion algorithm attempts to represent every optimal path be-tween clusters and inside clusters. However, most maps have far simpler topographies than the worst-case; in our experi-mental scenarios we often observed the same path returned for different pairs of (c; s) parameters when discovering intra-edges. This presents us with an opportunity to com-pact the graph by removing unnecessary duplication from the abstract edge set.

Consider the initial abstraction in Figure 3(a) and contrast it with our desired result in Figure 3(b). fE3; E5g repre-sent the same path between nodes w and y but are anno-tated with different clearance values. The same is true for

fE4; E6g which both cover nodes u and y. In such cases we say that E3 and E4 are strongly dominant, which we denote E3 E5 and E4 E6. This is an irreflexive and asymmetrical relationship between edges which we for-malise with the following theorem:

Theorem 6. Let fea; ebg 2 Eabs be two edges which con-nect the same pair of abstract nodes and are annotated with

capabilities ca cb 2 C such that:

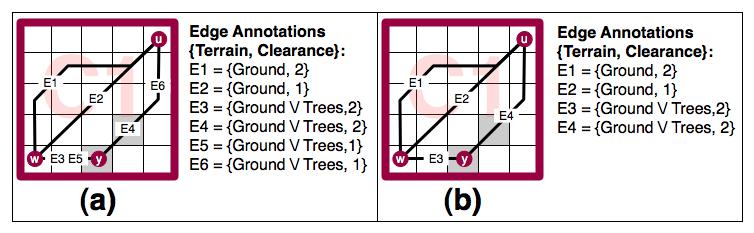
* ea(cb) eb(cb) ^ weight(ea) = weight(eb)

Then ea eb and we may remove eb from Eabs without loss of generality or optimality.

Proof. Since ca cb it must be the case that any agent with the correct capability to traverse eb must be likewise able to traverse ea. Further, if ea(cb) eb(cb) holds, it must also be the case that any agent large enough to traverse eb is also large enough to traverse ea. These conditions are sufficient to preserve generality. Finally, since ea is equal in weight to eb we cannot lose optimality by removing removing eb. 

We term the resultant graph in which all strongly domi-nant edges have been removed a high-quality abstraction.

Figure 3: Strong edge dominance



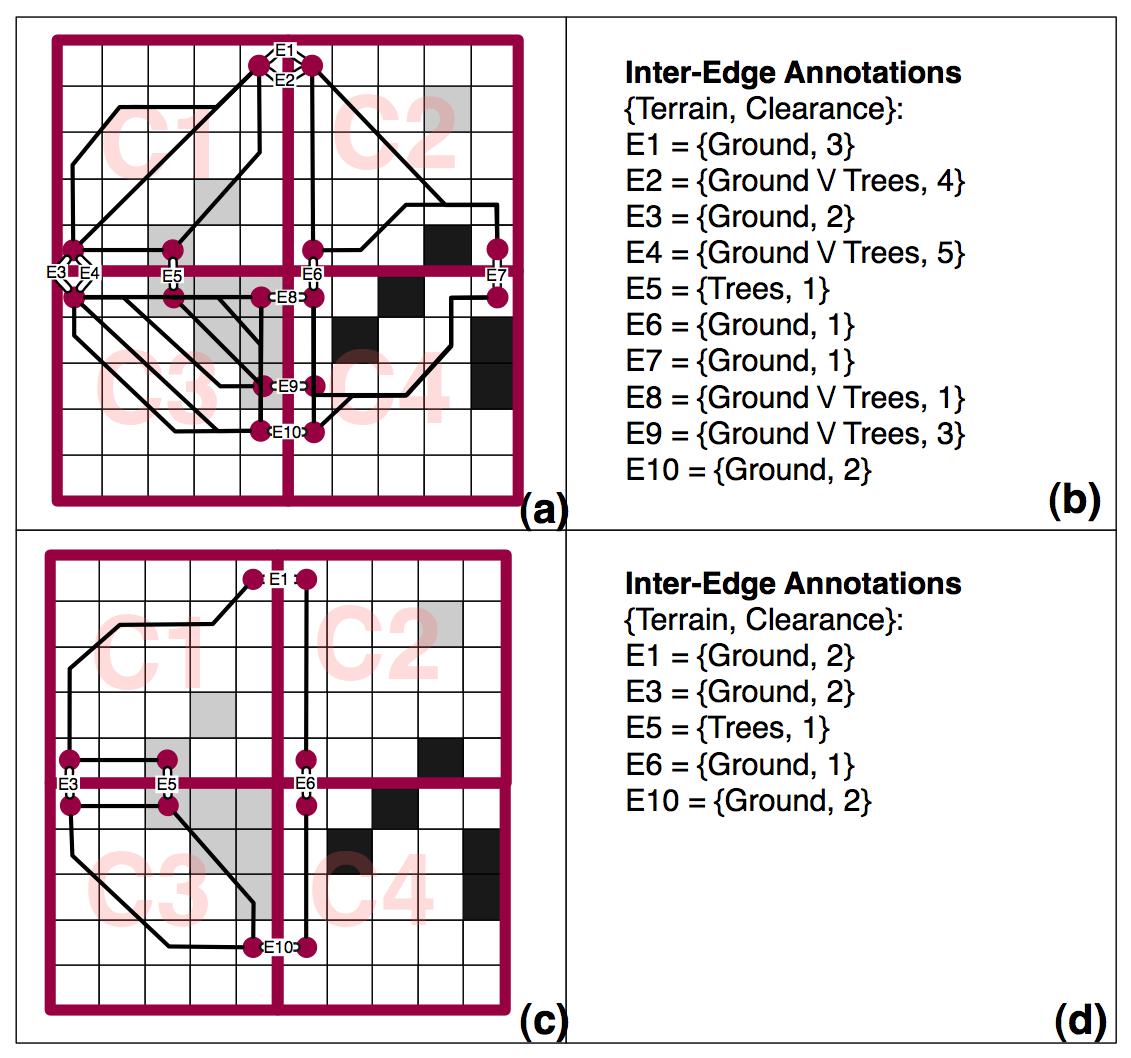
A further observation made during our analysis of this problem was that in many cases there exist multiple alterna-tive routes to reach a goal location. The shortest paths tended to involve the traversal of optimal-length multi-terrain edges however, it was often possible to reach the same destination using slightly longer single-terrain edges. This suggests that the abstract graph can be further compacted without affect-ing the completeness of the representation.

A reasonable analogy to highlight our intuition here is to compare the way off-road vehicles opportunistically use roads where possible even if an off-road route of trail exists which has a smaller distance cost. We prefer roads because they connect most points of interest, are smoother to drive on and have other benefits such as less wear and tear and better fuel consumption.

Figure 4(a) and 4(b) show a typical high quality abstrac-tion while in Figure 4(c) and 4(d) we highlight the desired result after further compacting the graph. In this example we can see that although edges E1 and E2 have different traversal requirements any agent of size s 2 S : S = f1; 2g capable of traversing E2 can also traverse E1 without loss of generality. In such cases we say E1 is weakly dominant and denote it as E1 % E2. Notice also that E3 % E4, E6 % E7, E10 % E8 and E10 % E9.

As with theorem 6, this relationship is irreflexive and asymmetric. Unlike strong dominance however, only rep-resentational completeness (and not optimality) is retained. We formalise it as:

Figure 4: High and low quality abstraction results



Theorem 7. Let La and Lb be two adjacent clusters, and

fwa; xbg; fya; zbg 2 Vabs two pairs of abstract nodes, each pair connecting La and Lb. Denote the inter-edges

associated with these node pairs as fewx; eyzg 2 Eabs and suppose they are annotated with clearance values

ewx(cwx); eyz(cyz) : cwx cyz 2 C. In this scenario, ewx % eyz iff the following conditions are met:

1. The capability dominance condition: ewx(cyz)

eyz(cyz).

1. The circuit condition: 9ewy; exz 2 Eabs which connect fwa; yag and fxb; zbg such that ewy(cyz) eyz(cyz) and

exz(cyz) eyz(cyz).

Then, any location which can be reached by traversing eyz can also be reached via ewx.

Proof. If a circuit exists between the set of edges

fewx; eyz; ewy; exzg in which every edge is traversable by cyz with a clearance value at least equal to eyz(cyz), then

it follows that any nodes in La or Lb which are reachable from ya or zb must be reachable from wa or xb. Thus, any destination an agent can reach via eyz can also be reached via ewx. 

Corollary 8. If ewx % eyz, then ya and zb and are also dominated and can be removed, unless required by another (non-dominated) inter-edge.

Proof. If ya and zb are required by a non-dominated inter-edge we cannot remove them without violating the capabil-ity dominance condition which is required to retain repre-sentational completeness. If this is not the case however, we know by the circuit condition that any node reachable by an intra-edge via ya or zb is also reachable via the endpoints of ewx. Thus, both nodes and any associated intra-edges de-pendent on them, can be safely removed. 

In many situations multi-terrain inter-edges tend to be as-sociated with very large clearances; much larger than the size of our largest agent. This unnecessarily limits the appli-cability of the capability dominance condition from theorem

1. Leveraging the fact that max(s) 2 S is known, we can maximise the number of edges which are weakly dominated by applying the following truncation condition to Eabs be-fore theorem 7:

e(c) > max(s) ) e(c) = max(s) : s 2 S; 8e 2 Eabs

Of course, opting for a low-quality abstraction in this way does affect the quality of computed solutions. In the worst case, a one-step transition of cost 1.0 in a high quality graph may be as long as f(n) = 4n+f(n 2) : f(2) = 3; f(3) = 13, where n 2 is the length of a cluster in a low-quality approximation. This is a pathological case however; as we will show the differences in real-world scenarios are much smaller and still near-optimal. The choice of which qual-ity abstraction technique to employ will depend on the re-quirements of the specific application; it is a classic tradeoff between run-time performance vs space.

Hierarchical Planning

Given a suitable graph abstraction, we can once more turn our attention back to agent planning. We use a similar pro-cess to that described in (Botea, Muller,¨ and Schaeffer 2004) but in our case we substitute A\* for AA\*. We provide a brief overview of the process here; for a more detailed descrip-tion, we direct the reader to the original work.

We begin by using the x; y coordinates of the start and goal nodes to identify the local cluster each is located in. Next, we insert a two temporary nodes into the abstract graph (which we remove when finished) to represent the start and goal. Connecting the nodes to the rest of the graph involves attempting to find an intra-edge from each node to every other abstract node in the cluster using AA\*. This phase involves i + j searches in total, corresponding to the number of combined abstract nodes in the start and goal clusters.

To compute a high-level plan we again use a variation on A\* – this time to evaluate the annotations of abstract edges before adding a node to the open list. Once the search ter-minates we can take the result, and, if immediate execution is not necessary, we are finished. Otherwise, we refine the plan by performing a number of small searches in the orig-inal gridmap between each pair of nodes along the abstract optimal path.

We term the resultant algorithm Annotated Hierarchical A\* (AHA\* for short).

Experimental Setup

We evaluated the performance of AA\* and AHA\* on a set of 120 octile-based maps, ranging in size from 50x50 to 320x320, which we borrowed from a popular roleplay-ing game. The same maps were used by (Botea, Muller,¨ and Schaeffer 2004) in their original study. In their de-fault configuration the maps only featured one type of traversable terrain interspersed with hard obstacles. We therefore created five derivative sets (making for a total of 720 maps) where each traversable tile on every map had one of f10%; 20%; 30%; 40%; 50%g probability of being con-verted into a second type of traversable terrain (a soft obsta-

cle). This allowed us to evaluate the algorithms in environ-ments featuring a range of soft and hard obstacles.

For each map we generated 100 experiments by ran-domly creating valid problems between an arbitrarily cho-sen pairs of locations and some random capability. We used two agent sizes in each experiment: small (occupy-ing one tile) and large (occupying four tiles) resulting in 144000 problem instances (720x200) overall. All experi-ments were conducted on a 2.4GHz Intel Core 2 Duo proces-sor with 2GB RAM running OSX 10.5.2. To implement both planners we used the University of Alberta’s freely avail-able pathfinding library, HOG (www.cs.ualberta.ca/ ˜nathanst/hog.html).

Results

In Figure 5 we present the size of the abstract graphs rela-tive to the size of the original graphs which featured an av-erage 4469 nodes and 16420 edges. We look at the effect of increasing the amount of soft obstacles (SO) from 0% (the original test maps with only one traversable terrain) to 50%. We also contrast high and low quality abstractions (denoted HQ and LQ) on a range of cluster sizes f10; 15; 20g (de-noted CS10, CS15 and CS20).

Figure 5: Size of abstract graph with respect to original graph.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0% SO** | **10% SO** | **20% SO** | **30% SO** | **40% SO** | **50% SO** |
| Nodes | 9.0% | 14.6% | 16.6% | 17.7% | 18.3% | 18.5% |
| **CS10 HQ** |  |  |  |  |  |  |
| Edges | 8.2% | 32.6% | 38.4% | 37.8% | 35.7% | 35.0% |
| Nodes | 5.3% | 7.9% | 10.3% | 12.8% | 15.0% | 15.7% |
| **CS10 LQ** |  |  |  |  |  |  |
| Edges | 2.2% | 6.7% | 11.6% | 17.0% | 22.0% | 23.6% |
| Nodes | 5.6% | 9.6% | 11.0% | 11.8% | 12.2% | 12.3% |
| **CS15 HQ** |  |  |  |  |  |  |
| Edges | 6.0% | 28.0% | 32.4% | 32.1% | 30.0% | 29.4% |
| Nodes | 2.9% | 4.8% | 6.4% | 8.1% | 9.7% | 10.3% |
| **CS15 LQ** |  |  |  |  |  |  |
| Edges | 1.2% | 4.6% | 8.5% | 12.5% | 17.2% | 18.8% |
| Nodes | 4.0% | 7.0% | 8.1% | 8.8% | 9.1% | 9.2% |
| **CS20 HQ** |  |  |  |  |  |  |
| Edges | 5.0% | 25.0% | 28.8% | 28.3% | 26.3% | 26.0% |
| Nodes | 2.0% | 3.4% | 4.6% | 5.9% | 7.2% | 7.6% |
| **CS20 LQ** |  |  |  |  |  |  |
| Edges | 0.9% | 3.6% | 6.9% | 10.1% | 14.4% | 15.8% |
|  |  |  |  |  |  |  |



The first thing to notice is that in all cases the abstract graph is only a fraction the size of the original graph. As ex-pected, larger clusters generate smaller graphs; the smallest abstractions are observed on the SO 0% problem set using CS20. In this case, using a HQ abstraction results in 4.0% the number of nodes found in the original graph and 5.0% edges. LQ abstractions fare even better featuring just 2.0% as many nodes and 0.9% edges.

The total space complexity associated with storing a graph is given by the total amount of space required to store both nodes and edges. If we assume each non-abstract node and edge require one byte of memory to store, then our smallest abstract graph, which contains 2 annotations per edge (capability and clearance, together requiring 1 ad-ditional byte), will have a space complexity 8.7% the size of the original graph using a HQ abstraction and just 1.8% using an LQ abstraction. Similarly, the largest HQ graph, which occurs for CS10 on SO 20%, has a space complexity 63.8% the size of the original. By comparison, LQ graphs are largest for CS10 on SO 50%; here 40.4% the size of the original gridmap. Moving from CS10 to CS20 reduces

the worst-case space complexity of HQ graphs to 47.0% and 26.4% for LQ graphs.

Interestingly, if we use the number of nodes as an indica-tor for the number of inter-edges in a graph, we may deduce that most HQ graphs are predominately composed of intra-edges. The exact number depends on the density of soft ob-stacles in clusters; less dense clusters (as found on SO 20%) result in more intra-edges as more unique paths (of differing sizes and capabilities) are found between each pair of ab-stract nodes. This is consistent with lemma 4 and useful to understanding the worst-case behaviour of HQ abstractions.

The linear increase in the size of LQ graphs is the result of a greater number of single-terrain entrances found as the number of soft obstacles increases (an observation consis-tent with lemma 3). Increasing the density of soft obstacles in a cluster causes the circuit condition from theorem 7 to be less often satisfied and leads to the observed worst-case on SO 50%.

Next we consider the performance of AHA\* with respect to path quality. We measure this as:

apl opl

%error = 100

where opl is the length of the optimal path as calculated by AA\* and apl the length of the abstract path used by AHA\*.

Figure 6: AHA\* performance (HQ vs LQ abstraction)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **(a) Path Quality vs. Soft Obstacles** | | | |  |  |  | **(b) Path Quality vs. Agent Size (CS15)** | | | | |  |
|  | 15 |  |  |  | ● | CS10 HQ |  | 15 |  |  |  | ● | Size=1 HQ |  |
|  |  |  |  |  |  | CS10 LQ |  |  |  |  |  |  | Size=1 LQ |  |
|  |  |  |  |  |  | CS15 HQ |  |  |  |  |  |  | Size=2 HQ |  |
|  |  |  |  |  |  | CS15 LQ |  |  |  |  |  |  | Size=2 LQ |  |
|  |  |  |  |  |  | CS20 HQ |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | CS20 LQ |  |  |  |  |  |  |  |  |
| %error | 10 |  |  |  |  |  | %error | 10 |  |  |  |  |  |  |
| ● |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |
|  |  |  |  |  | ● | ● |  |  |  |  |  |  |  |  |
|  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
|  | 5 | ● | ● |  |  |  |  | 5 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | ● | ● | ● | ● |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  | 0 |  |  |  |  |  |  |
|  | 0 | 10 | 20 | 30 | 40 | 50 |  | 0 | 10 | 20 | 30 | 40 | 50 |  |
|  |  |  | % soft obstacles | |  |  |  |  |  | % soft obstacles | |  |  |  |

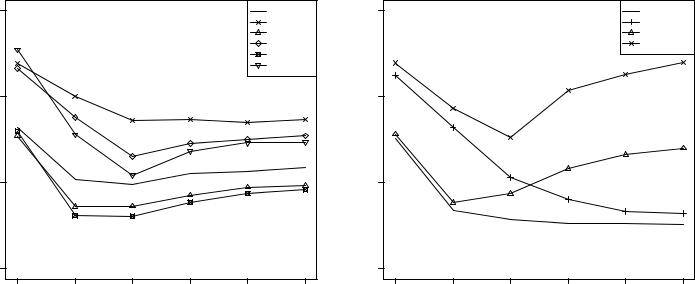


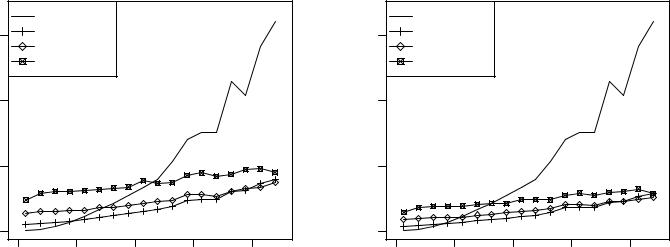
Figure 6(a) shows the average performance of AHA\* with respect to cluster size and soft obstacles. Notice that HQ graphs yield a very low error; in most cases between 3-6%. Perhaps most encouraging however are the results for LQ abstractions, where in most cases AHA\* performs within 6-10% of optimal. The highest observed error in both cases occurs on SO 0% and is due to our inter-edge placement strategy. In all situations the pair of nodes with maximal clearance in an entrance, which we choose as our transition point, tends to be towards the beginning of the entrance area which is not an optimal placement. On maps that produce low-complexity clusters of predominately one terrain this re-sults in long entrances that are poorly represented by the sin-gle inter-edge. Increasing the amount of soft obstacles pro-duces shorter entrances and generates more transition points leading to a significant reduction in error. It appears AHA\* is so optimised for complex cases that it suffers some minor performance degradation on simpler problems.

Interestingly, the error associated with both HQ and LQ abstractions reaches a minimum on SO 20% before gradu-ally increasing toward SO 50%. To better understand this

phenomenon we present in Figure 6(b) the performance of both small and large agents on HQ and LQ graphs using a fixed cluster-size of 15. Notice that the performance of small agents continues to improve beyond SO 20% while large agents begin to degrade. The observed rise in error stems from the decreasing size of entrances on the problem sets featuring denser clusters. As previously shown in Figure 5, maps with more soft obstacles result in a greater number of smaller entrances. This situation is beneficial for smaller agents (there are more transitions to choose from) but is dis-advantageous for larger agents. As the average entrance size shrinks fewer inter-edges exist with clearance > 1 and the location of such edges along the border area between clus-ters may be quite varied; sometimes we find an entrance to-ward the beginning of the border area, other times in the middle and sometimes toward the end. Consequently, clus-ter traversal by large agents is often not in a straight line; the abstract paths produced frequently feature a zig-zagging effect that is responsible for the observed error and is most pronounced on SO 50%.

Figure 7: AHA\* total search effort (nodes expanded).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **(a) Total search effort (SO 20%, HQ)** | | | | | | |  |  | **(b) Total search effort (SO 20%, LQ)** | | | | | | |  |
|  | 15000 | ● |  | AA\* | |  |  | ● |  | 15000 | ● |  | AA\* | |  |  | ● |  |
|  |  |  | AHA\* CS15 | |  |  |  |  |  |  | AHA\* CS15 | |  |  |  |  |
|  |  |  |  | AHA\* CS10 | |  |  |  |  |  |  |  | AHA\* CS10 | |  |  |  |  |
|  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |
| expanded | 10000 |  |  | AHA\* CS20 | |  |  |  | expanded | 10000 |  |  | AHA\* CS20 | |  |  |  |  |
|  |  |  |  |  |  | ● |  |  |  |  |  |  | ● |  |
| avg.nodes |  |  |  |  |  |  |  | ● | avg.nodes |  |  |  |  |  |  |  | ● |  |
| 5000 |  |  |  |  |  | ● | ● | 5000 |  |  |  |  |  | ● | ● |  |
|  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  | ● |  |
|  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |
|  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |
|  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |  |
|  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |  |
|  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |  |
|  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |  |  |
|  |  |  |  |  | ● |  |  |  |  |  |  |  |  | ● |  |  |  |  |
|  |  |  |  | ● | ● |  |  |  |  |  |  |  | ● | ● |  |  |  |  |
|  | 0 | ● | ● |  |  |  |  |  | 0 | ● | ● |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 0 |  |  | 100 | 200 | 300 | 400 |  |  | 0 |  |  | 100 | 200 | 300 | 400 |  |
|  |  |  |  |  | optimal solution length | | |  |  |  |  |  |  | optimal solution length | | |  |  |



Finally, we turn our attention to Figure 7 where we eval-uate AHA\* using a search effort metric. Here we contrast the total number of nodes expanded by AHA\* (during inser-tion, hierarchical search and refinement phases) with AA\* on both HQ and LQ graphs. We focus on the SO 20% prob-lem set in order to analyse the effect on search effort as path length increases but note that similar trends hold for the other problem sets.

Looking at Figure 7(a) we observe that agents using HQ graphs featuring large cluster-sizes are disadvantaged in this test. The insertion effort required to connect start and goal to each abstract node in their local clusters heavily domi-nates the total effort causing AHA\* CS20 to trail AA\* for problems up to length 250. We can see the gap between CS20 and the smaller cluster sizes decrease as problem size grows but our benchmark set of experiments are not hard enough for such coarse-grain map decompositions to be ad-vantageous. By comparison, in Figure 7(b) we see that the difference is less pronounced using LQ graphs (there are less abstract nodes per cluster) however it appears CS10 or CS15 are more suitable choices for problems up to our maximum length, 450.

Conclusion

date. In this paper we have addressed this issue by show-ing how clearance-based obstacle distances can be computed and leveraged to improve path planning for multi-size agents in heterogenous-terrain grid-world environments. Our ap-proach reduces complex problems involving agents of dif-ferent sizes and multi-terrain traversal capabilities to much simpler single-size, single-terrain search problems. Build-ing on these new insights, we have introduced a new planner, Annotated Hierarchical A\*, and have shown through com-parative analysis that AHA\* is able to find near-optimal so-lutions to problems in a wide range of environments yet still maintain exponentially lower search effort over standard A\*. Our hierarchical abstraction technique is simple to apply but very effective; we have shown that in most cases the over-head for storing the abstract graph is a small fraction of that associated with non-abstract graphs.

Future work could involve looking at computing annota-tions to deal with elevation and other common terrain fea-tures. We are also interested in finding a better inter-edge placement approach and reducing the effort to insert the start and goal into the abstract graph. Finally, we believe AHA\* could be usefully applied to solving heterogenous multi-agent problems.

Acknowledgements

NICTA is funded by the Australian Government as repre-sented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Coun-cil through the ICT Centre of Excellence program.

We would like to thank Philip Kilby and Eric McCreath for their help and insightful comments during the develop-ment of this work.

References

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

Demyen, D., and Buro, M. 2006. Efficient triangulation-based pathfinding. In AAAI, 942–947.

Geraerts, R., and Overmars, M. 2007. The corridor map method: a general framework for real-time high-quality path planning. Computer Animation and Virtual Worlds 107–119.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A for-mal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cyber-netics (4):100–107.

Latombe, J. 1991. Robot Motion Planning. Kulwer Aca-demic Publishers.

Sturtevant, N. R., and Buro, M. 2005. Partial pathfind-ing using map abstraction and refinement. In AAAI, 1392– 1397.

Heterogeneity in path planning is characteristic of many

real-world problems but has received very little attention to