A Survey of Path-finding Algorithms Employing Automatic Hierarchical Abstraction

Jonathan Vermette
vermett@uwindsor.ca
University of Windsor

The demand for fast, near-optimal search driven by the needs of modern computer games, which can feature hundreds of simultaneous requests, has spurred the development of a wide range of pathfinding techniques. One approach that is seeing increased use by industry programmers and researchers is the incorporation of automatically generated hierarchical abstractions into their pathfinding systems. The following is a survey of the research that has been conducted in this area, including papers that present new algorithms, novel hierarchical constructions, and analyses of the effectiveness of hierarchical abstraction to the pathfinding field.

Categories and Subject Descriptors: []:

Additional Key Words and Phrases:

Contents

1. INTRODUCTION

1.1 Overview

Pathfinding is a well known and studied problem in the Artificial Intelligence community, being applicable to many fields. It can be found for example in robotics, logistics, or simulation systems that study real world issues like crowd dynamics. One domain where pathfinding is especially prevalent is video games. Here, the need for highly efficient techniques is apparent as modern games place high demands on the CPU and memory, also needing time for graphics, physics simulation. Pathfinding itself is typically only part of a larger overall game AI system, which may incorporate goal selection, group coordination, animation, etc.. One researcher who collaborates with a major computer game developer cites their limit of only 1-3ms of a update frame towards pathfinding for *all* agents [Bulitko et al. 2007]. With problem complexity tied to search space size, and sizes ever growing, faster and accurate pathfinding techniques are always in demand by game industry professionals.

For this reason, video games are finding greater prominence amongst researchers as a testbed. The majority of the papers examined in this survey focused on introducing new algorithms intended for the video game AI community, and those that incorporate empirical experiments do so using video game derived test problems.

The particular technique that is the object of this survey is the use of the *abstraction hierarchies* in pathfinding algorithms. The concept of abstraction itself is natural, based on the way people reason about the world and how they move through it. [Botea et al. 2004] provide a motivating example in their paper of the general idea: Consider a road trip across North America. If one had a complete map of the continent down to the level of individual streets and address for every town, planning a trip street by street is the not the normal way of planning out the trip. Rather, one would consider only a means of leaving their city, followed by a high level plan moving through states and provinces towards their destination. Here, the various streets in Windsor, Ontario or any other city are *abstracted* into a city. The various cities are abstracted into states and provinces. The states and provinces are abstracted into the United States and Canada, and the two countries are abstracted into North America. Abstraction hierarchies are a way of representing this human style method of path-finding for a computer program.

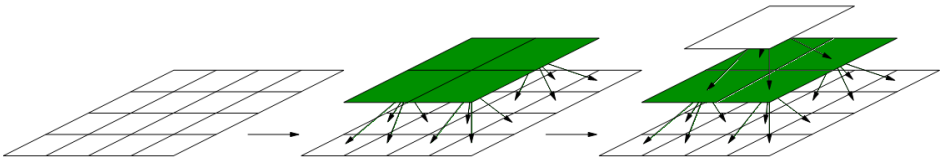

Fig. 1. Abstracting into a hierarchy (taken from [Sturtevant and Buro 2005])

A hierarchy then is nothing more than a series of successive abstractions. This is illustrated in Figure 1. A common notation is to reference the hierarchy as layers or

levels in ascending order, with the lowest, L_0 being the un-abstracted game space and subsequent layers numbered L_1, L_2 and so on, up to a L_l.

Researchers have taken different approaches in how a hierarchy is used in their search algorithms. [Holte et al. 1996] runs A* at L_0 and uses its hierarchy to query and refine heuristic distance estimates. [Bulitko et al. 2007] conducts search at a high level to generate a constraint for the lowest level. The remaining algorithms in this survey however follow the approach outlined in the North America trip example. A complete abstract path is defined, which is then 'filled in' with more refined sections of path at lower abstraction levels, until a complete path is defined at the lowest level which the agent can follow.

The results presented in the papers examined in this survey show that hierarchical abstraction has led to many-fold reduction in running times for pathfinding problems. Claimed results have been as high as one hundred times faster than simple A* search [Sturtevant 2007].

2. SURVEY OF RESEARCH

This survey is broken down into sections thematically. Each defines a different style of constructing a abstraction hierarchy. Section 2.1 examines clique-based hierarchies, which are built around the particular topography of the search space. Section 2.2 examines work conducted in the area of sector-based hierarchies, which forms a regular structure. Finally, Section 2.3 looks at the newest form of abstraction applied to game pathfinding, the Contraction Hierarchy, which differs significantly from the other two.

2.1 Clique-Based Hierarchies

2.1.1 Overview.

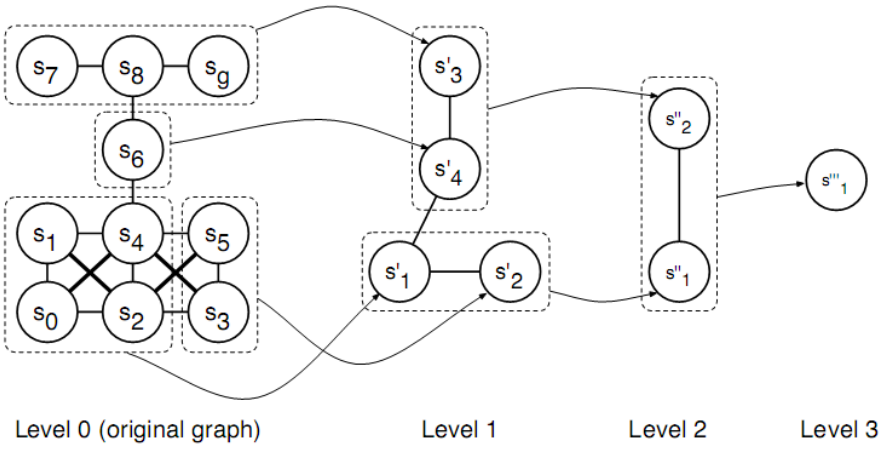

Fig. 2. An illustration of the clique-abstraction, (from [Bulitko et al. 2007], Fig. 9). Each level is a successive abstraction of the previous level.

In graph theory, a *clique* is understood to be a subset of an undirected graph where every pair of nodes in the sub-graph are joined by an edge. Though not adhering to the formal definition, 'clique' is a useful shorthand for describing this style of abstraction. With each, groups of nodes are abstracted based on their relationship to each other. Figure 2 illustrates this idea.

This technique originates with [Holte et al. 1996], who discussed grouping nodes with its neighbors, starting with the node of maximal degree, and working down.

Later work focused on two-dimensional grid spaces representing a game 'map'. [Sturtevant and Buro 2005] describes building an abstraction around 4-cliques, and smaller groupings where nodes have lower degrees. [Bulitko et al. 2007] used this same implementation for its hierarchy construction.

[Demyen and Buro 2006] took this idea and applied it to triangulations of 2-d polygonal space mapped to a graph based on triangle adjacency. The abstract nodes represent polygonal corridors which can be searched through.

2.1.2 Hierarchical A* (HA*).

Holte et al. were concerned with the use of search space abstractions to create heuristic estimates, but in a manner that was more efficient than the techniques known at the time.

The authors note that others examined the use of abstraction techniques to generate admissible heuristics, mentioning [Gaschnig ; Guida and Somalvico 1978; Pearl 1984; Prieditis and Davis 1995] as examples of this approach. In general, these other techniques work by using the true cost of moving from start to goal in an abstracted search space as the heuristic estimate of the problem in the search space.

However, they point out that using such a heuristic results in increased computational cost to find a solution. They claim that these existing approaches cannot assure that the cost of generating a heuristic will exceed the cost of using no heuristic at all. They cite the analysis performed by [Valtorta 1984] as evidence of this claim. Further, they claim that the only existing program that broke "Valtorta's Barrier" was the Absolver II in [Prieditis and Davis 1995], but only for certain types of problems.

Holte et al. devised an algorithm to search for a solution in an abstraction hierarchy, calling this algorithm Hierarchical A*. To build an abstraction hierarchy, Holte et al. used the STAR graph abstraction technique from [Holte et al. 1996]. In the STAR technique, the node on the search graph with the highest degree is abstracted with it's neighbours into a single abstract state, repeating until all nodes have been mapped to this abstract graph. A hierarchy is produced by repeating the process on the created abstract graph, producing a higher level abstraction, and repeating this process until it is collapsed to a single node.

They describe the operation of the algorithm as running much the same as regular A* search. When a state is expanded, the heuristic costs of its successors is determined by discovering the cost from state to goal in the next level of the abstraction hierarchy, which when determined is passed down to the lower abstraction level as a heuristic estimate. Knowing that an abstracted state at level l represents many states in $l-1$, Hierarchical A* caches these costs to speed up the algorithm by preventing rework.

To test their new algorithm, the authors set up several series of experiments to compare their Hierarchical A* against what they call A* with no heuristic scoring ($h(s) = 0$). They call this "blind search". Both approaches were tested in different search spaces, with examples including the 5-puzzle and Hanoi-7.

Holte et al. claim to have expanded fewer search nodes than indicated by Valtorta's Barrier in the majority of problems they tested. In two particular search spaces, they claim Hierarchical A* was the more efficient algorithm for 95% of the problems they tried.

The authors state that they have shown that "A* search using heuristics created automatically by homomorphic abstractions . . . can outperform blind search". They claim that this shows that hierarchical path-finding is worth further study.

2.1.3 Partial Refinement A* (PRA*).

Sturtevant and Buro examined the problem of real-time path-finding, which allows an interleaving of planning and execution to occur. Traditional path-finding algorithms like A* return a complete solution, leaving a search agent to wait until the algorithm is completely finished. Real-time path-finding allows the search agent to perform actions while search is being conducted by returning partial solutions to follow or otherwise incorporate actions into the solution. The authors are particularly interested in the domain of real-time strategy games, which require planning for many agents in a common space while subject to limited resources and the desire for a high quality result.

Sturtevant and Buro refer to [Korf 1990], which introduced the Learning Real-time A* (LRTA*) algorithm as one of the first attempts at developing a real-time path-finding algorithm. Hierarchical A*, which was introduced in [Holte et al. 1996] is not a real-time algorithm, but is mentioned as an algorithm that is similar in using an abstraction hierarchy to attempt to speed up the discovery of a solution in path-finding. Finally, [Botea et al. 2004] is cited as introducing another hierarchical approach, the HPA* algorithm.

The shortcoming noted by Sturtevant and Buro of most previous algorithms is that they are not strictly real-time algorithms. Classic search algorithms slow down as problems grow more difficult, breaking time constraints. They also cannot deal with changes such as minor detours. LRTA* is noted as satisfying real-time requirements, but can produce solutions of very poor quality.

To address these shortcomings, the authors introduce a path-finding algorithm that interleaves planning and execution by calculates partial solutions which result in a complete, valid path.

First they describe a means of building an abstraction of the search environment. Diverging from the sector-style of HPA*, Sturtevant and Buro instead abstract based on areas where search nodes form cliques of up to 4 nodes. Nodes with only a single edge are treated as part of the clique they share the edge with. This produces an abstract graph of the original, which is itself, repeating until a hierarchy of graphs is created with each level being the abstract graph of the level below, and the top level being a single node for the entire search space.

The Partial Refinement A* algorithm builds on a simple approach they call *QuickPath*. QuickPath works by walking up the abstraction hierarchy from the

start and goal locations to determine the lowest abstract node that encompasses both. The path at an abstraction level by looking at the edges one level below which joining the given abstraction nodes. The authors claim this process is guaranteed to return a valid result.

The authors go on describe several enhancements to distinguish PRA* from QuickPath, mostly by expanding on how precisely how traversing between two abstract nodes is refined in the level below in the hierarchy.

The authors conducted an experiment to measure PRA* against the contemporary HPA* algorithm. Building the abstraction hierarchy was ignored. They created a problem set of 148480 paths across 116 maps taken from the Baldur's Gate and Warcraft 3 computer games [Corp et al. 1998; 2000; Entertainment 2002].

Sturtevant and Buro make two major claims with regards to the PRA* algorithm. First, they claim that it performs similar to HPA* in that is outperforms the classic A* algorithm in running time while finding solutions with near-optimal path lengths. Specifically they claim that in 95% tested cases that PRA* produced a solution with 5% of the optimal, similar to result reported in [Botea et al. 2004]. Secondly, they claim that PRA* performs well in a real-time context where only a small percentage of running time is available for path planning and execution is being performed. Citing the Real-time Strategy problem domain, they claim from their experimental results that even when planning is restricted to 1% of the CPU budget, "long paths ... less than 5% away from optimal in 98% of the cases".

The authors state PRA* is "... the first time that partial path-finding methods have been implemented and analysed".

2.1.4 Triangulation Reduction A* (TRA*).

In this paper the authors address the problem of abstracting the traversable space of an environment into a set of triangles for efficient path-finding. Further, they are concerned about abstracting in a fashion that deals with complex obstacles, to reduce path suboptimality.

The authors refer to the work of [Holte et al. 1996], which is a search algorithm using layers of abstractions to produce heuristic values. They also note the similarity of their prosed algorithm to HPA* [Botea et al. 2004] and PRA* [Sturtevant and Buro 2005] in quickly defining an imprecise, high-level path and later refining the individual steps.

The only shortcoming the authors address is a problem common to hierarchical algorithms in general in that there is a trade-off between the time to find a path, and a reduction in quality. They do put forward a general claim that none of the previous solutions use a polygonal representation.

In this paper Demyen and Buro introduced two new path-finding algorithms designed to work specifically in environments composed of triangles. Triangulation A* (TA*) is a triangulation path-finding algorithm, and Triangulation Reduction A* (TRA*) which finds paths in a triangulation abstraction. The authors construct a Delauney triangulation to which forms their graph, stating that this guarantees an optimal path will not revisit any triangles.

Triangulation A* (TA*) is their proposed algorithm for path-finding across a series of connected triangles. They mention it is similar to the algorithm proposed by

[Kallmann 2005]. They claim to differ from this technique in estimating the g-cost to traverse the triangle whereas Kallmann method of measure centroid distances. The results is a "funnel" of triangles used to produce a path sufficiently far enough away from triangles to prevent a collision by the agent following it.

This serves as the basis for the main algorithm they introduce, Triangulation Reduction A* (TRA*), which incorporates abstraction to further reduce the size of the search graph. In TRA*, each triangle mapped to a node in an abstract graph, joined by edges where they share edges in the triangulation. By examining the degrees of the nodes, a series of reductions are performed to collapse corridors of triangles to a single abstracted node. This is repeated until the abstraction forms a tree. TRA* walks this tree in a search query to produce a reduced triangulation for TA*to work on and produce a final low-level path.

The authors performed an experimental comparison of TA*, TRA*, and PRA*, electing to use the identical data set used int [Sturtevant and Buro 2005], testing for running time and path quality

The authors claim to have outperformed PRA* with TA* alone, with 95% of paths being found at least twice as fast, attributing the ability of triangles to sparsely represent a map over grids. Applying abstraction to run TRA* led to an even larger reduction in running time. 95% of paths fell withing a few percentage points of the optimal as well. Jansen and Buro did note that TRA* falters when dealing with a search space of many small obstacles, which caused large growth in the size of the abstraction.

Jansen and Buro state that their new algorithm outperforms current algorithms, specifically mentioning PRA*, especially on larger maps and shows the promise of triangulation-based path-finding over grid-based in general.

2.1.5 Partial Refinement Learning Real-time Search (PR LRTS). In this paper, the authors look at the problem of real-time heuristic search as defined by Koenig in [Koenig 2001]. These are path-finding algorithms that interleave planning and execution. They state that such algorithms must be capable of performing a constant amount of planning actions in a constant time interval, while a search agent performs actions. Specifically, they were concerned with improving the trade-off between planning time and the rate of *convergence* to an optimal or near-optimal solution. They note that these measure are antagonistic; "reducing the amount of planning done before each action improves the agent's response time, but leads to slower learning due to lower-quality actions taken by the agent".

Bulitko et al. note that others in the field have introduced state abstractions to reduce the running time in exchange for a (usually slight) reduction in path quality. As examples, they cite Hierarchical A* [Holte et al. 1996] and HPA* [Botea et al. 2004] as earlier algorithms which do this.

The authors state that to date that has not been any work showing that the tradeoff between path quality and running time is being done in an optimal fashion.

The primary contribution of this paper is an algorithm they call Path Refinement Learning Real-time Search (PR-LRTS).

For their abstraction hierarchy, the authors use a clique hierarchy. Fully connected components are abstracted into a single node, with single-edged nodes considered to be part of a clique. Abstracting all such groups forms an abstraction

layer, which can be repeated l times. The authors state they are using the technique as done previously by [Sturtevant and Buro 2005].

To conduct search, the authors employ LRTA* [Korf 1990] at level l. LRTA* performs a fixed-depth search and determine a single step. The authors say this 'step' represents a groups of nodes at the base level of search, which they use as a search corridor. Within this corridor, they perform another search to establish a path to follow. The authors claim that because there is a constant upper bound on the number of nodes that form a corridor, they can assert a real-time claim as there is therefore an upper bound on the amount of search being conducted between moves.

The authors note throughout the paper that they are assuming the *freespace* assumption, which means the agent initially assumes a completely open map. As it senses its local environment, the map is updated with obstacle information. In light of this, the authors state that they implemented Local Repair A* [Stout 1996] as opposed to classic A* for the lowest-level search.

The freespace assumption also means the abstraction hierarchy is dynamically updated as search progresses. The authors state that when an obstacles in sensed in the path, movement halts and re-planning begins. They note that such graph updates involve removing states and edges, not inserting them, so heuristics remain admissible.

The authors conducted a set of different experiments with different objectives:

—The authors analysed their algorithm against others by an examination of dominating factors. They define an algorithm as dominant when its average performance for some measure is better than some other algorithm. Their algorithm was compared against A* and LRTA*.

—They also examined the effects of using an abstraction hierarchy on individual performance measures. The authors chose three diverse settings for their algorithm and compared the results against each other to investigate the effect of the settings on performance.

—The authors analysed how their algorithm's performance scales with increasing problem size, with problem size expressed as cost of the optimal path.

The test bed for all of these experiments were 3000 search problems on 6 maps taken from popular computer games [Corp et al. 1998; Entertainment 2002]. The problems were uniformly distributed in optimal cost from 50 to 100.

The authors claim that for any single measure of performance in isolation, A* or LRTA* appears to be the best choice (their algorithm being the exception when looking at convergence planning). But when comparing antagonistic statistics plotted on a graph, their algorithm is the dominant. From this they claim that their algorithm is the most efficient in balancing these antagonistic measures. They also claim to have found that as the number of abstraction levels in the hierarchy is increased, suboptimality is increased, while convergence measures are lowered. Finally, they claim their trend for increasing problem size is an increase in suboptimality.

In addition, the authors presented a proof that their algorithm is complete (that is, that it will always find a solution if one exists), and more importantly that planning is constant-bounded.

Bulitko et al. assert that PR LRTS is the first real-time algorithm at that time to use any kind of automatic abstraction mechanism. Prior algorithms, they claim, are not real-time because must plan a complete path (even if only at an abstract level), and thus are not constant bounded. Other real-time algorithms make no use of abstraction and are dominated by the use of an abstraction hierarchy.

2.1.6 Summary.

Table I provides a quick breakdown of the major contributions using clique-based hierarchical abstractions.

Table I. Summary of Research on Clique-Based Hierarchical Algorithms

Year	Title	Author(s)	Contribution
1996	Hierarchical A*: Searching Abstraction Hierarchies Efficiently	R.C. Holte, M.B. Perez, R.M. Zimmer, A.J. MacDonald	Introduced Hierarchical A*, first hierarchical path-finding algorithm. Uses STAR abstraction technique.
2005	Partial Pathfinding Using Map Abstraction and Refinement	Nathan Sturtevnt and Michael Buro	Introduced PRA* algorithm, clique-based abstraction basis for subsequent algorithms
2006	Efficient Triangulation-Based Pathfinding	Douglas Demyen and Michael Buro	Introduced TRA* algorithm. Abstractions represent 'corridors' of polygon space
2007	Graph Abstraction in Real-time Heuristic Search	Vadim Bulitko and Nathan Sturtevant and Jieshan Lu and Timothy Yau	Analysis of effectiveness of hierarchical algorithms. Introduces PR LRTS, claims to be first Hierarchical *real-time* pathfinding algorithm

2.2 Sector-Based Hierarchies

2.2.1 Overview.

In contrast to clique based hierarchies, another approach is to define a regular partitioning and group all nodes in a partition into a single state. This was introduced by [Botea et al. 2004]. Figure 3 illustrates this.

[Harabor and Botea 2008] applied sector-abstraction to their algorithm as well.

This style lends itself to pre-computation of many smaller solutions. The sectors have well-defined entry and exit points.

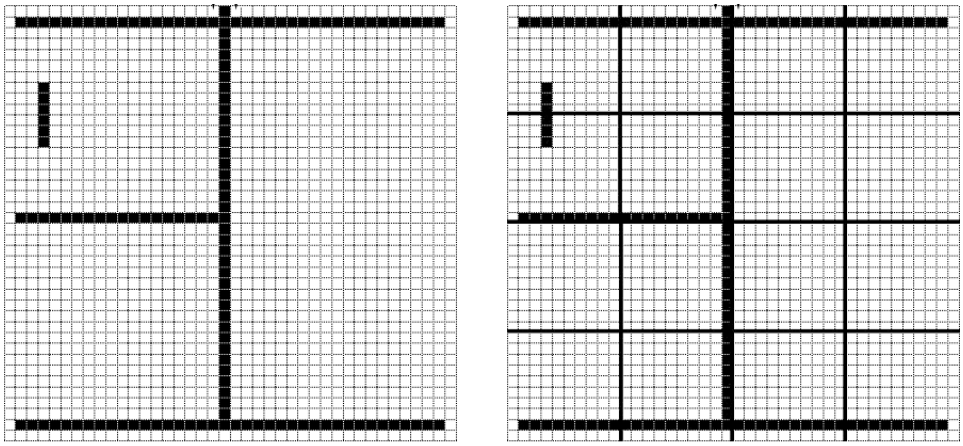

Fig. 3. An illustration of the sector-abstraction, (from [Botea et al. 2004])

2.2.2 Hierarchical Path-finding A* (HPA*).

Botea et al. were concerned with the need for highly efficient path-finding algorithms used in computer games, which is a demanding domain.

Rabin [Rabin 2000a] described a two-level hierarchy, which the authors state only has a "high-level presentation of the approach". By the same author, the *visibility graph* in [Rabin 2000b] is described, which they note as being especially suited for interior game maps due to the nature of the geometry favouring large convex obstacles, but is less suited for outdoor environments. *Navigation Meshes* are described as a way of reducing unblocked potions of a 2D environment into "a minimal set of convex polygons".

The authors mention the method in [Tozour 2002] as a fast way to produce a navigation mesh. Also described are some hierarchical approaches applied by the robotics community. Mentioned are the use of *quadtrees* in [Samet 1988] to reduce a map to unblocked square cells. The authors claim that agents follow suboptimal paths by always moving to the middle of cells. Referred to as an improvement on this are *framed quadtress* from [Chen et al. 1995]. The authors describe this approach as a way to improve solution quality while consuming much more memory.

Hierarchical A* from [Holte et al. 1996] is described by the authors as also using a hierarchical representation of a search space, but it concerned instead with using the hierarchy to generate heuristic functions.

The authors note that while visibility graphs can produce good quality paths, the technique produces complex graphs in game maps involving many small obstacles or concave shapes, reducing its usefulness. A forest scene is given as an examples where graph complexity would quickly grow. Quadtrees, they note, do not produce optimal paths because search agents are directed to seek out the centre of a cell as way-points along a path. Framed quadtrees address this specific problem, but at the cost of using much more memory to produce the framing cells.

The authors describe their technique as a process that occurs in three steps when a search is performed, which they term an *online search*. The first step is to perform a search from the beginning location to the border of the location's neighbourhood, followed by a search across the abstract graph of all neighbourhoods to the neighbourhood containing the goal location using the A* approach of [Stout 1996]. Finally, local search is again performed inside the goal neighbourhood.

To build these neighbourhoods, the authors describe breaking up a map into rectangular areas called *clusters* of equal size. Transition cells are locations in adjacent clusters that can be traversed. Inside a cluster, costs are stored for the optimal path between two transition points. If no such path exists without searching into another cluster, the cost is not defined. The transition cells and associated edges form an abstract graph of the game world. The authors say that this process can be repeated again on the abstract graph to produce another.

Botea et al. describe optional enhancements to this approach. First, refinement can be performed by running a set of small searches between each set of transition cells inside a cluster to find an optimal path. They suggest that these paths can be cached if appropriate. The author's second suggestion is the smoothing of paths by replacing sections with straight lines where possible.

To test their approach, the authors claim to have run 100 searches on a set of 120 different maps taken from the Baldur's Gate series of computer games [Corp et al. 1998; 2000] with maps varying between a size of 50×50 to 320×320 cells. Two and three-level hierarchies were created with a variety of cluster sizes.

In their analysis, the authors report that a A* returned a solution sooner than their HPA* implementation for problems with a small optimal solution. The authors attribute this to "the overhead of HPA*...larger than the potential savings that the algorithm could achieve". They also report that A* outperforms on "straight line" problems, as "using the Euclidean distance as heuristic provides perfect information, and A*expands no nodes other than those that belong to the solution". However the authors report that HPA* outperforms low-level A* for problems larger than the overhead associated with their abstraction approach. The authors also discuss the additional memory involved to support the abstraction hierarchy.

The authors state that they have presented a hierarchical path-finding algorithm that is domain independent and works for different kinds of map topologies. They further claim that it is suitable for large scale problems and is capable of handling dynamic changes to a search environment, and claim it does so while beating the

classic A* algorithm in running time and producing solutions that are near-optimal.

2.2.3 HPA* Enhancements.

Jansen and Buro took the Hierarchical Pathfinding A* (HPA*) algorithm introduced in [Botea et al. 2004] and addressed shortcomings of this particular algorithm, proposing remedies to produce an updated version.

The authors note that others have examined the use of abstraction for speeding up path-finding. Recent work includes PRA* [Sturtevant and Buro 2005] and TRA* [Demyen and Buro 2006], both of which were designed to deal with computer game maps. The authors however have focused on the earlier HPA* algorithm from [Botea et al. 2004].

Jansen and Buro identify two shortcomings of HPA*. They note that HPA* usually does not produce an optimal path at the ground level, so the original implementation uses a smoothing technique is applied to straighten the path where possible to shorten it. They describe the original implementation as a series of ray casts along each node, and adjusting the path where suitable path intersections are found, which works but is computationally expensive method. They also claim that the paths found between each pair of entrance nodes for a given sector could have an improved worst-case running time.

Though other hierarchical algorithms were cited, this paper focuses on the shortcomings of the HPA* algorithm. Any shortcomings of similar algorithms are not discussed.

The authors have proposed a set of three improvements to the HP* algorithm. First, they propose a new means of path smoothing to address the shortcoming described earlier. They propose to restrict the amount of ray-casting performed by limiting the ray-cast to a bounding box of some size centred on the node the rays are cast from. The authors claim this results in a minor reduction of path quality but with a significant reduction in running time.

Secondly, they mention that HPA* uses A* to determine costs E entrance nodes in a given sector of size $L \times L$. Jansen and Buro propose instead using the Dijkstra single-source shortest path algorithm as a better alternative.

Lastly, they propose that in dynamic domain that updating the abstraction hierarchy should use a *lazy computation* scheme, as a given sector may change several times before it is needed in a search query. In this way the amortised cost of recomputing the hierarchy should be lower.

For a sector of size $L \times L$, the authors note the worst-case running time for A* to build a path between two sector entrances is $O(L^2)$, compounded by searching for all pairs. Because a single pass of Dijkstra will find all shortest-paths for a given entrance in a single pass they reason that it has a lower worst-case complexity than A*.

In support of these proposed enhancements the authors claim to have performed an experiment comparing HPA* with and without these enhancements to show their effectiveness. 116 maps from popular computer games [Corp et al. 1998; 2000; Entertainment 2002] and 80 artificial generated maps.

The authors claim to have found the following results from their experiments:

- The use of small bounding boxes to restrict the time spent ray casting led to a

large decrease in running time with a minor reduction in smoothing quality.

- They claim that the use of Dijkstra's algorithm over A* can be up to twice as fast.
- Lazy computation of edge weights led to a much faster initial build of the abstraction hierarchy.

The authors state that they have shown that their enhancements to the HPA* algorithm are promising. They also suggest that their use of lazy computation may be applicable to other hierarchical algorithms to deal with "many moving objects, changing topologies, and incremental terrain discovery".

2.2.4 Hierarchical Annotated A* (HAA*).

Harabor and Botea begin this paper with the claim that many current path-finding algorithms plan for agents of varying sizes and movement capabilities, but do not account for this. They assert that current algorithms work assuming all agents are of homogeneous size or capability, whereas in modern computer games agents can be heterogeneous. They claim their new algorithm focuses on this problem using a annotation hierarchy.

The authors refer to the work of [Botea et al. 2004], [Sturtevant and Buro 2005] and [Bulitko et al. 2007] as hierarchical path-finding methods relevant to their work. Harabor and Botea also cite the Brushfire algorithm from [Latombe 1991] as using a similar technique to denote obstacle proximity. The Corridor Map Method [Geraerts and Overmars 2007] and the Triangulation A* and Triangulation Reduction A* methods [Demyen and Buro 2006] are identified by the authors as other path-finding techniques that are capable of planning for multiple-sized agents since they incorporate obstacle clearance planning.

The authors assert that the majority of existing path-finding algorithms assume agents are of a uniform size or assume that they are all equally capable of traversing the same types of terrain. They state that this is a limitation of the kinds of problems they are capable of solving and could fail in a domain with agents of different sizes and/or capabilities.

Harabor and Botea also introduce a *clearance value* metric, notating for each tile the upper bound on size for an agent to legally occupy the tile. These values are calculated for every tile/capability pair.

These annotations are incorporated into A* search as additional parameters, producing a variant called Annotated A* (AA*). AA* evaluates search nodes as blocked if their annotation does not include the capability or size of the agent. They note that these annotated gridmaps allow a reduction to a simplified *canonical* problem, being a grid of blocked and unblocked cells and an atomic agent i.e. a homogeneous representation.

To improve efficiency the authors extended AA* with a hierarchical grid representation. They build upon the technique from [Botea et al. 2004] which constructs square cluster with entrances defined between two maximally spaced points along the border. They extend this into a *set* of entrances to accommodate different composite capabilities. Intra-sector edges for the abstract layer are constructed by running AA* over all capability/sector pairs.

To test the effectiveness of AA* and HAA*, the authors used the same set of 120 game maps used in [Botea et al. 2004], along with modified versions of each for a total of 720 different maps. Two different agent sizes were used, taking up 1×1 and 2×2 tiles. The derivative maps incorporate tiles traverse-able by only some agents, which they refer to as *soft obstacles*. 100 problems were created for each map for a total of 144,000 problems. Concerns were the abstract-graph size, path quality, and search effort.

Harabor and Botea report that in all instances of their experiments the map abstraction uses less than half the memory of the original search graph, with one instance having only 2.0% the number of nodes and 0.9% the number edges of the original. Larger cluster sizes in the abstraction generally produced smaller abstract graphs.

Path quality was measured by comparing the lengths of the AA* and HAA* paths produced for a given problem. They claim that the more complex the map from soft obstacles, the lower the error introduced by abstraction. They also claim a trend of error increasing with agent size. Harabor and Botea suggest that larger agents are forced to a single location to transition between two given abstraction clusters, producing suboptimal paths.

For search effort (the number of node expansions to find a solution), the authors claim to have found that there is a trade-off point favouring HAA* once the problems exceed a certain length, depending on other factors like the size of the clusters.

Harabor and Botea state that their new algorithm shows that incorporating consideration for heterogeneity can be done without sacrificing path quality. They claim to have shown that their analysis shows HAA* does this while still outperforming A*.

2.2.5 Summary.

Table II provides a quick breakdown of the major contributions using clique-based hierarchical abstractions.

Table II. Summary of Research on Sector-Based Hierarchical Algorithms

Year	Title	Author(s)	Contribution
2004	Near Optimal Hierarchical Path-finding	Adi Botea, and Martin Müller and Jonathan Schaeffer	Introduced HPA*, first sector-based hierarchical path-finding algorithm.
2007	HPA* Enhancements	M. Renee Jansen and Michael Buro	Described several changes to HPA* to reduce running time for a slight quality trade-off
2008	Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments	Daniel Harabor and Adi Botea	Introduced HAA*. Claims to be first attempt to explicitly account for multiple-factor search agents in a hierarchical path-finding algorithm.

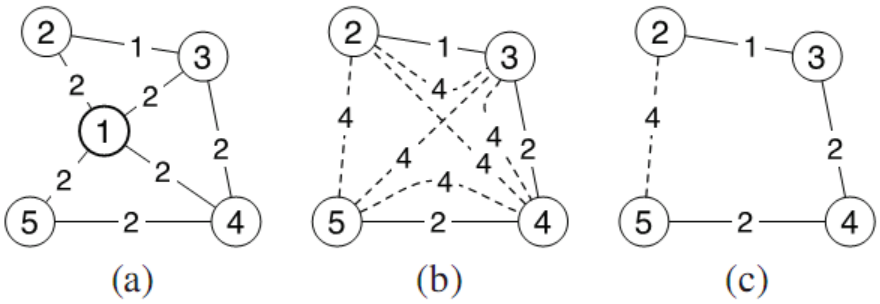

Fig. 4. An illustration of contracting a graph (taken from [Sturtevant and Geisberger 2010], Figure 5). Dashed edges are edge shortcuts derived from contraction.

2.3 Contraction-Based Hierarchy

2.3.1 Overview. A new approach is radically different from the previous. [Geisberger et al. 2008] describe a hierarchy that is more properly described as a long chain of transformations. Every level of the hierarchy is the abstraction of a single node by replacement with edges.Figure 4 illustrates this.

This is a relatively new approach to pathfinding in computer games, with only a single paper found introducing the techniques to the game AI community at the time this survey was compiled.

2.3.2 Contraction Hierarchies (CHs).

In this paper, Geisberger et al. looked at the problem of planning an optimal route on a road network.

This paper does not refer to the other papers examined in this survey, but does refer to work in the area of road networks. The authors refer to their work as an extreme case of highway-node routing [Schultes and Sanders 2007; Schultes 2008]. Geisberger et al. state their original motivation was to eliminate the need for complicated Highway Hierarchies (HHs) [Sanders and Schultes 2005; 2006].

Geisberger et al. did not reference any particular shortcomings of any of the other papers examined in this survey. They make the claim that HHs rely on a subroutine called Edge Reduction that even with improvements [Goldberg et al. 2007; Bauer and Delling 2008] was an expensive process, which they claim is unnecessary given the results of their new approach.

GGeisberger et al. contribute a new path-finding technique with a data structure they call a *Contraction Hierarchy*, which can then be queried to find a route between two different locations. The authors provide a description of constructing such a hierarchy and the query method.

To construct a contraction hierarchy, nodes of the target graph are ordered according to a given importance function to define a queue. They state the given important of a node to the contraction can change as the graph is processed, so a lazy update scheme is used. Contracting a node involves replacing it with 'shortcut'

edges that describe the cost of traversing the node to its respective neighbors. Each such abstraction is a level in the contraction hierarchy.

Querying the contraction hierarchy for a path involves a bi-directional Dijkstra search that terminates when the ends meet. Shortcuts are unpacked in a recursive fashion.

The authors present a lemma that proves the bi-directional search produces a shortest path between two locations.

An empirical experiment was conducted to evaluate CHs against what they describe as the fastest current variant of Highway Node Routing (HNR) [Schultes and Sanders 2007]. For their road network, the authors tested against a model of Western Europe comprised of over 18 million nodes and 42 million edges. Edge weights were derived from estimated travel times based on real road conditions. The authors compared statistics like average query time, memory consumption, and pre-processing time. A variety of different heuristics to determine node ordering were also considered.

The authors claim to have found that CHs have greatly improved query times over HNRs across all the heuristic they tried for constructing the hierarchy. For example, they claim that using a particular ordering parameters results in query times four times faster than HNR. The results they present show that initial construction of the hierarchy can take longer (dominated by the time to order the graph), but this is a preprocessing step and only performed once.

The authors state that their new algorithm has been shown to outperform the current leader in road network routing. They further claim that the simplicity of their approach suggests that further improvements are likely.

2.3.3 Analysis of Contraction Hierarchies.

Sturtevant and Geisberger presented a post-mortem on the abstraction technique employed during the development of a recent computer game called Dragon Age: Origins (DA:O). They note that no analysis was performed at the time to measure the effectiveness of the two-level abstraction technique applied in the game.

The authors note that popular planning techniques in games are often based on traversable terrain abstractions like navmeshes [Tozour 2002] or waypoint graphs [Lidén and Valve-Software 2002], which may be created by hand, abstraction which are created automatically [Botea et al. 2004].

While the authors do not address the shortcomings of a particular algorithm, they note two general shortcomings of abstract search spaces. They note that abstractions still grow with the size of a problem, so given a large enough search space the abstracted space may be so large that path-finding still takes an unacceptable amount of time. This can be mitigated by using a coarser (therefore smaller) abstraction, they claim, but this second issue is that larger abstractions lower the quality of any paths produced as low-level features are lost.

As part of a product launch post-mortem, the authors contribute a comparative analysis of the two level abstraction hierarchy that was implemented in the DA:O game. They compare this implementation against alternative approaches like developing a better search heuristic, and employed a new abstraction called a contraction hierarchy (CH).

Sturtevant and Geisberger describe the abstraction in DA:O as a 16×16 sector abstraction of a grid-based map. These sectors are then subdivided into contiguous regions with each represented by a single node. This is the same technique described in [Sturtevant and Jansen 2007]. To adequately handle larger maps or interior maps, another layer of abstraction was applied to produce a two-layer abstraction hierarchy.

The authors describe the use of improved heuristics as an alternative means of improving search results. They examined the use of a *differential heuristic* called ALT [Goldberg and Harrelson 2005], which they briefly describe as deriving distances using the triangle inequality and reference points called *landmarks*.

The authors then provide a description of a contraction hierarchy as a contrast to their approach, and propose several means of reducing CH storage overhead. They did this by leveraging a data structure designed for mobile systems [Sanders et al. 2008], which they claim compressed the storage for a CH by 70%, but doubling the query time.

Sturtevant and Geisberger conducted an experiment to compare the performance of the three techniques. They describe their testbed as 10,000 randomly selected problems on 120 maps from the Baldur's Gate game [Corp et al. 1998]. The measured statistics were memory usage, planning cost and path refinement cost.

From a baseline average of 28 kilobytes of storage for the maps, the authors claim to have found that using a two-level sector abstraction averaged from 30 to 35k. The CH averaged from 25k to 111k based on the heuristic used. Using a differential heuristic required on average 55k of storage.

For planning cost the authors examined the number of nodes expanded, and the query time in μs. They claim that all 3 techniques outperformed a basic A* search as expected. Sector-abstraction and CH reported similar results and outperformed the heuristics. For example, they claim differential heuristics took $209\mu s$ on average for the longest 10%, compared to $64.3\mu s$ and $68.6\mu s$ for the best implementations of Sector-abstraction and CH respectively.

The authors come to the conclusion that with regards to the abstractions developed for DA:O, a larger sector size for the higher level of abstraction would have resulted in improved performance were it implemented.

The authors say that the relative strengths and weaknesses of either approach means it is difficult to state that one is superior to the other. Abstraction hierarchies and easy to understand and implement and 'recursively similar' and able to handle dynamic changes, but admit suboptimal paths. Contraction hierarchies offer high performance for difficult problems, noting the fact that long paths are actually faster to compute in a CH than shorter ones. They recommend to the reader that anyone building a high-quality motion planner familiarise themselves with both techniques.

2.3.4 Summary. Table III provides a quick breakdown of the major contributions using clique-based hierarchical abstractions.

Table III. Summary of Research on Contraction-Based Hierarchical Algorithms

Year	Title	Author(s)	Contribution
2008	Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks	Robert Geisberger and Peter Sanders and Dominik Schultes and Daniel Delling	Introduced Contraction Hierarchy (CH) paradigm based on research in road networks.
2010	A Comparison and High-Level Approaches for Speeding Up Pathfinding	Nathan Sturtevant and Robert Geisberger	Conducted comparison between sector and contraction based hierarchies, and offered improvements for CH for computer games.

3. REFERENCES

REFERENCES

Bauer, R. and Delling, D. 2008. Sharc: Fast and robust unidirectional routing. In *ALENEX*, J. I. Munro and D. Wagner, Eds. SIAM, 13–26.

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

Bulitko, V., Sturtevant, N. R., Lu, J., and Yau, T. 2007. Graph abstraction in real-time heuristic search. *J. Artif. Intell. Res. (JAIR) 30*, 51–100.

Chen, D. Z., Szcerba, R. J., and Urhan Jr., J. J. 1995. Planning conditional shortest paths through an unknown environment: A framed-quadtree approach. *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and System Human Interaction and Cooperation 3*, 33–38.

Corp, B., Studios, B. I., and Entertainment, I. 1998. Baldur's gate.

Corp, B., Studios, B. I., and Entertainment, I. 2000. Baldur's gate II: Shadows of amn. CD-ROM.

Demetrescu, C., Ed. 2007. *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*. Lecture Notes in Computer Science, vol. 4525. Springer.

Demyen, D. and Buro, M. 2006. Efficient triangulation-based pathfinding. In *AAAI*. AAAI Press.

Entertainment, B. 2002. Warcraft III: Reign of chaos. [CD-ROM].

Gaschnig, J. A problem similarity approach to devising heuristics: First results. *IJCAI'79*, 301–307.

Geisberger, R., Sanders, P., Schultes, D., and Delling, D. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, C. C. McGeoch, Ed. Lecture Notes in Computer Science, vol. 5038. Springer, 319–333.

Geraerts, R. and Overmars, M. H. 2007. The corridor map method: Real-time high-quality path planning. In *ICRA*. IEEE, 1023–1028.

Goldberg, A. V. and Harrelson, C. 2005. Computing the shortest path: search meets graph theory. In *SODA*. SIAM, 156–165.

Goldberg, A. V., Kaplan, H., and Werneck, R. F. F. 2007. Better landmarks within reach. See Demetrescu [2007], 38–51.

Guida, G. and Somalvico, M. 1978. A method for computing heuristics in problem-solving. In *AISB/GI (ECAI)*. 115–121.

Harabor, D. and Botea, A. 2008. Hierarchical path planning for multi-size agents in heterogeneous environments. In *Proceedings of the IEEE Symposium on Computational Intelligence in Games (CIG 08)*.

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

Holte, R. C., Perez, M. B., Zimmer, R. M., and MacDonald, A. J. 1996. Hierarchical a*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*. 530–535.

Jansen, M. R. and Buro, M. 2007. Hpa* enhancements. See Schaeffer and Mateas [2007], 84–87.

Kallmann, M. 2005. Path planning in triangulations. In *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*. Edinburgh, Scotland.

Koenig, S. 2001. Agent-centered search. *AI Magazine 22,* 4, 109–132.

Korf, R. E. 1990. Real-time heuristic search. *Artif. Intell. 42,* 2-3, 189–211.

Latombe, J.-C. 1991. *Robot Motion Planning*. Kulwer Academic Publishers.

Lidén, L. and Valve-Software. 2002. *Strategic and tactical reasoning with waypoints*. AI Game Programming Wisdom. Charles River Media, 211–220.

Pearl, J. 1984. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley.

Prieditis, A. and Davis, R. 1995. Quantitatively relating abstractness to the accuracy of admissible heuristics. *Artif. Intell. 74,* 1, 165–175.

Rabin, S. 2000a. *A* Aesthetic Optimizations.* Game Programming Gems. Charles River Media, 264–271.

Rabin, S. 2000b. *A* Speed Optimizations.* Game Programming Gems. Charles River Media, 272–287.

Samet, H. 1988. An overview of quadtrees, octrees, and related hierarchical data structures. *NATO ASI Series, vol. F40.*

Sanders, P. and Schultes, D. 2005. Highway hierarchies hasten exact shortest path queries. In *ESA*, G. S. Brodal and S. Leonardi, Eds. Lecture Notes in Computer Science, vol. 3669. Springer, 568–579.

Sanders, P. and Schultes, D. 2006. Engineering highway hierarchies. In *ESA*, Y. Azar and T. Erlebach, Eds. Lecture Notes in Computer Science, vol. 4168. Springer, 804–816.

Sanders, P., Schultes, D., and Vetter, C. 2008. Mobile route planning. In *ESA*, D. Halperin and K. Mehlhorn, Eds. Lecture Notes in Computer Science, vol. 5193. Springer, 732–743.

Schaeffer, J. and Mateas, M., Eds. 2007. *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference, June 6-8, 2007, Stanford, California, USA*. The AAAI Press.

Schultes, D. 2008. Route planning in road networks. In *Ausgezeichnete Informatikdissertationen*, D. Wagner, Ed. LNI, vol. D-9. GI, 271–280.

Schultes, D. and Sanders, P. 2007. Dynamic highway-node routing. See Demetrescu [2007], 66–79.

Stout, B. October/November 1996. Smart moves: Intelligent pathfinding. *Game Developer Magazine*.

Sturtevant, N. R. 2007. Memory-efficient abstractions for pathfinding. See Schaeffer and Mateas [2007], 31–36.

Sturtevant, N. R. and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *AAAI*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, 1392–1397.

Sturtevant, N. R. and Geisberger, R. 2010. A comparison of high-level approaches for speeding up pathfinding. In *AIIDE*, G. M. Youngblood and V. Bulitko, Eds. The AAAI Press.

Sturtevant, N. R. and Jansen, M. R. 2007. An analysis of map-based abstraction and refinement. In *SARA*, I. Miguel and W. Ruml, Eds. Lecture Notes in Computer Science, vol. 4612. Springer, 344–358.

Tozour, P. 2002. *Building a Near-Optimal Navigation Mesh.* AI Game Programming Wisdom. Charles River Media, 171–185.

Valtorta, M. 1984. A result on the computational complexity of heuristic estimates for the A* algorithm. *Inf. Sci. 34,* 1, 47–59.