An Analysis of Map-Based Abstraction and Refinement

Nathan Sturtevant and Renee Jansen

Department of Computing Science, University of Alberta, Edmonton, Alberta, T6G 2E8, Canada
{nathanst|maaike}@cs.ualberta.ca

Abstract. A variety of techniques have been introduced over the last decade for abstracting search graphs and then using these abstractions for search. While some basic work has been done to predict the value of an abstraction mechanism, the results have not been validated in practice. In this paper we analyze a variety of old and new abstraction mechanisms in a pathfinding testbed and show that the work done in abstraction-based refinement-style search can be predicted by the diameter and size of abstract nodes.

1 Introduction

Search is a widely studied task in artificial intelligence, due to the fact that many problems such as pathfinding and scheduling can be solved using search techniques. The traditional and widely used search algorithm is A* [1], which uses a heuristic function to guide its search. Unfortunately, this algorithm is computationally expensive: the amount of work required can be exponential in the length of the solution.

One way to deal with this problem is by abstracting the search space. In particular, it is possible to create a simpler search graph by representing a number of nodes of the original search graph by a single node in an abstract search graph. Abstract edges are then added based on the edges that exist in the original search graph. This can be done recursively, giving a hierarchy of abstractions. An approximate solution can be found by a search in the abstract graph, which can then be *refined* to a solution in the original search space.

Abstraction methods have been studied in a number of places. For example, Holte *et al.* proposed the STAR abstraction, in which a node and all nodes within some predefined radius are abstracted together [2]. Botea *et al.* developed an abstraction method specifically designed for gridworlds, which divides the grid into square clusters [3]. Another abstraction method, devised by Sturtevant and Buro, takes cliques in the original search graph and abstracts them into a single node in the abstract graph [4]. Abstraction has been used in other domains such as robotics [5] and planning [6].

Holte *et al.*'s STAR abstraction was developed as a result of a theoretical analysis of the amount of work required to find a solution to a search problem using abstractions of the search space [2]. The analysis shows that, in order to minimize the total amount of work done, it is desirable to minimize the maximum length of a path between any two nodes inside an abstract node, while maximizing the number of nodes that are combined into a single abstract node.

The goal of this paper is to verify the analysis done by Holte *et al.* of the amount of work done during a search which uses search space abstractions. We will first compare

the amount of work done with a variety of both new and old abstraction methods, and then show that the analysis of Holte *et al.* holds in practice.

1.1 Problem Definition

A search problem can be formally defined as the tuple (S, A, c, s_0, S_g), where S denotes the set of states in the environment, A denotes the set of actions, $c(s, a)$ is the cost of taking action $a \in A$ in state $s \in S$, $s_0 \in S$ is the start state, and $S_g \subseteq S$ is the set of goal states. The search space can be represented as a graph $G = (V, E)$, where V is the set of vertices (nodes), representing the set of states S, and E is the set of edges, representing the actions A. The weight of an edge is defined as the cost of performing action a in state s. In this paper, we will assume that the edges of the search graph are undirected, implying that the cost of going from state s to state s' is equal to the cost of going from state s' to s.

We look specifically at problems from the pathfinding domain. That is, underlying the graph representation of the world there is a grid-based map. Cardinal moves on the map (N, S, E, W) have cost 1 while diagonal moves have cost $\sqrt{2}$. We choose this focus, because pathfinding is currently a widely studied and well-motivated area, with applications in areas including robotics and computer games.

2 Abstraction Mechanisms

We first define an abstraction formally, and then present 5 different abstraction mechanisms which we will use experimentally to test the predictions made by Holte *et al.* [2]. The radius and clique abstractions have been described elsewhere; the other abstraction mechanisms are presented for the first time here.

2.1 Automatic State Abstraction

Formally, an abstraction is a graph homomorphism ϕ from a graph G_1 to a graph G_2 which maps nodes from G_1 to nodes in G_2. An edge e is added between two abstract nodes s_1 and s_2 whenever there is at least one edge e' between s'_1 and s'_2 such that $\phi(s'_1) = s_1$ and $\phi(s'_2) = s_2$. We can abstract the search graph recursively, giving an abstraction hierarchy of α levels, where level 0 is the original search graph and level α is the topmost abstract level. We will introduce a variety of such homomorphisms.

While edge costs are well-defined in the original problem space, we define the location of a node s in abstract space as the average location of all the nodes abstracted by s. This means that we can use the location of abstract nodes as a heuristic for searching the abstract graph. This heuristic will be admissible in abstract space, but not in the original problem space. In domains other than pathfinding, edges can have uniform cost.

2.2 Clique Abstraction

The clique abstraction (CA) was initially introduced by Sturtevant and Buro [4]. The idea behind this abstraction, as the name suggests, is to abstract cliques in each level of

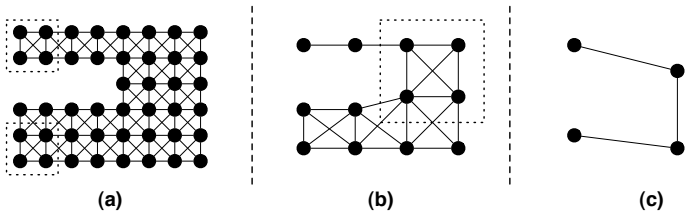

Fig. 1. Clique abstraction.

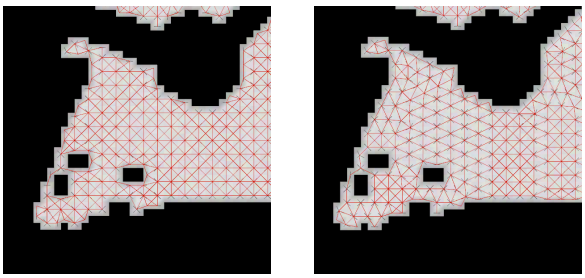

Fig. 2. Uniform (left) and non-uniform (right) abstractions.

the abstraction. Every node in a clique will be no more than one edge away from every node, which is a desirable property, according to the theoretical evaluation discussed in the next section. In a two-dimensional, octile-connected map, the maximum clique size is 4 nodes, which makes the clique-finding problem tractable.

We illustrate the clique-abstraction process in Figure 1. The initial graph is shown in the portion of this figure labeled (a). Two sample cliques are indicated with a dotted line. The middle figure, (b), shows one possible way the first graph can be abstracted. Note that where four-cliques cannot be abstracted, smaller cliques are removed instead. This abstraction mechanism can be applied once more to (b) to obtain the graph in (c). In (b), the marked clique is removed first, followed by the only other four-clique. The remaining pairs of nodes will be abstracted together. Depending on the order in which nodes are considered and the policy for abstracting nodes with only a single neighbor, the final graph will either take one or two steps to abstract until it is represented by just a single node.

In this paper we will use two different approaches for building a clique abstraction. The first method, CA(n) [*normalized*], builds the abstract graph in a more uniform manner by relying on knowledge of the underlying map. CA(n) first abstracts 4-cliques in a uniform manner across open areas of the map before considering the rest of the map. The second implementation of the clique abstraction, CA(i) [*irregular*], does not rely on knowledge of an underlying map and thus builds less uniform abstractions. We demonstrate the difference between a uniform and non-uniform abstraction in Figure 2. The lines in this figure represent edges, with nodes implicit at the ends of edges. In the

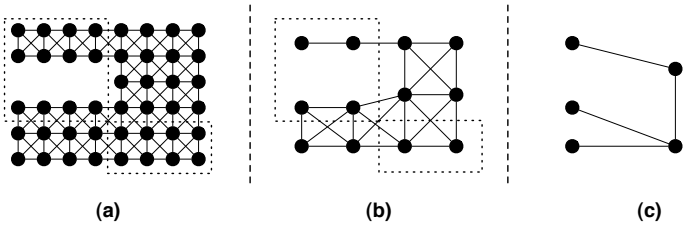

Fig. 3. Sector abstraction.

left portion of the figure, most of the map has been abstracted in uniform squares. Only the edges of the map are less uniform. Because the underlying topology is not known in the right portion of this figure, the abstraction is much less uniform.

There will be some cases where the abstraction procedure fails to find a clique among the remaining unabstracted nodes. In this case, these nodes can be passed through to the next abstraction level instead of being abstracted with their neighbors. In the case of nodes with only a single neighbor, we choose to abstract them into their neighbor regardless of whether they form a non-trivial clique or not.

2.3 Sector Abstraction

The sector abstraction (SA) is inspired by the abstraction used by HPA* [3], and is limited to grid-based maps. The sector abstraction is parameterized by a fixed sector size, k. At the first level of abstraction, sectors of size $k \times k$ are overlaid onto the map. Within each sector, a breadth-first search is used to determine connected components, each of which becomes an abstract node. At the ith level of abstraction, sectors of size $k^i \times k^i$ are used. Note that with an empty map and a sector size of 2, the clique abstraction and sector abstraction will be identical.

We demonstrate this abstraction mechanism with a sector size of 2 in Figure 3. In this example, the clique abstraction and sector abstraction both abstract the initial graph in the same manner. In graphs (a) and (b), two of the sectors (4×4) used for the building graph (c) pass are marked by dotted lines. Because only nodes which form a connected component within a single sector can be abstracted together, the top left sector becomes two separate nodes when abstracted. This results in one extra node in the most abstract graph on the right, (c). If we were to apply one more level of abstraction, the entire graph would be immediately abstracted into a single node, because all nodes in this graph are connected within an 8×8 sector.

2.4 Radius Abstraction

Holte *et al.* suggested an abstraction mechanism they called the STAR abstraction [2]. We use what is essentially the same mechanism, but refer to it as the radius abstraction (RA), which we feel is a more evocative description. The radius abstraction works by first selecting an unabstracted node. All neighboring nodes within a fixed radius, r, of

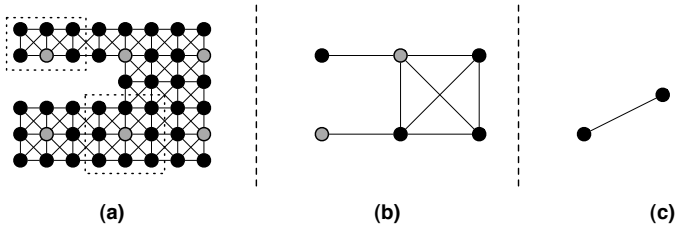

Fig. 4. Radius one abstraction.

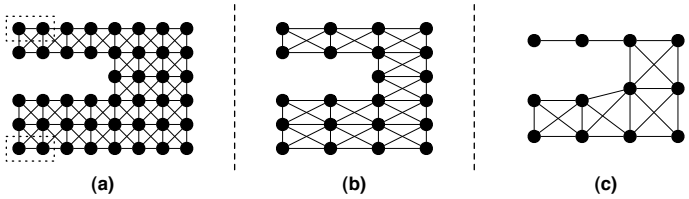

Fig. 5. Line abstraction.

this node are then abstracted together into the same abstract node. The radius, r, is the depth limit (in edges) on a breadth-first search which finds the neighbors to abstract. The radius abstraction procedure is simple and can be applied to any graph. We demonstrate the radius abstraction in Figure 4.

Our implementation of the radius abstraction chooses the next node to abstract at random; however, in this example we choose the nodes to be abstracted quite carefully. In the left part of Figure 4, (a), we mark in gray the nodes which are selected for abstraction. The immediate neighbors ($r = 1$) of these nodes are then abstracted together. In the first abstract graph, (b), there are only 6 nodes, and again, we mark in gray the nodes which are selected to drive the abstraction process. The resulting graph, (c), has two nodes, and will be fully abstracted at the next level of abstraction. The radius abstraction will remove more nodes in each step than the clique abstraction. A radius 1 abstraction can be quite similar to a sector abstraction with $k = 3$.

2.5 Line Abstraction

The line abstraction (LA) finds sequences of nodes length k, and abstracts them together. In this paper we experiment with two variants of the line abstraction. One variant abstracts the graph uniformly, as we will do in the example below. The other variant just selects nodes to abstract at random and is much less uniform. We vary the maximum length of the abstracted line between 2 and 6.

We demonstrate the line abstraction with $k = 2$ in Figure 5. First, we attempt to abstract each node with its neighbor to the right. This takes the original graph, (a) and transforms it into the graph (b). In the next step we attempt to abstract each node with

its neighbor below. This results in the graph on the far right, (c), which is identical to the graph produced by clique-abstraction in a single step. When done uniformly, the line abstraction proceeds in this manner, first abstracting horizontally and then vertically.

2.6 Node-Limit Abstraction

The node limit abstraction (NLA) has a single parameter k, the number of nodes to abstract together. Given an initial node, a breadth-first search is performed until k unabstracted nodes have been visited. These nodes are then abstracted into a single abstract node. The node-limit abstraction and line abstraction are the same when $k = 2$. If k is defined dynamically as the number of neighbors within a radius r, the node-limit abstraction will be the same as the radius abstraction. In this paper we use a fixed k for all nodes.

3 Abstraction Analysis

In this section we analyze the complexity of using an abstraction hierarchy to find a path through a search graph. For the moment we will consider using an algorithm which first finds a path at the highest possible level of abstraction, and then successively refines this path until a path is found in the original graph. This analysis is originally due to Holte *et al.* [2] and we follow their derivation closely here.

During the refinement process, a node s at some abstract level i is replaced by a series of nodes in level $i-1$. This is done by finding a path p in level $i-1$ consisting of nodes which are mapped to s. In particular, if we let the neighbours of s along the path at level i be t and u, the first and last nodes on path p must have neighbours t' and u' such that $\phi(t') = t$ and $\phi(u') = u$. We say refinement is *monotonic* if no backtracking needs to be done across levels. This will be the case throughout this paper.

The total amount of work done in finding a solution consists of the refinement costs at each level. If we assume that every abstract graph is strictly smaller than the graph it abstracts, the solution in the abstract graph at level α is trivial since this graph will only have a single node (one node per connected component in the original graph). In each refinement step, every node in the solution at level i is replaced by a sequence of nodes at level $i-1$. If we let the length of the path at level i be denoted by λ_i, and the work required to replace a node at level i by a sequence of nodes at level $i-1$ by ω, then the total work done in refining from level i to level $i-1$ is $\omega\lambda_i$. The *expansion factor* χ is defined to be λ_i/λ_{i-1}, giving $\lambda_i = \chi\lambda_{i-1} = \chi^{\alpha-i}$.

If we let ω and χ be the worst cases, this gives a bound on the total work done to find a solution:

$$Total\ Work \leq \omega \sum_{i=1}^{\alpha} \chi^{\alpha-i}$$

This is equivalent to:

$$Total\ Work \leq \omega \sum_{i=0}^{\alpha-1} \chi^{i}$$

Furthermore, we know that χ is upper-bounded by d, the maximum diameter of any abstract state. The *diameter* is the maximum distance between any two states in that abstract state. (This distance is defined as the number of *nodes* on a path.) This gives:

$$Total\ Work \leq \omega \sum_{i=0}^{\alpha-1} d^i$$

The sum over d^i can be can be represented by the closed form formula $(d^\alpha - 1)/(d-1)$. If the diameter is at least 2, we can replace this with d^α. We will assume that $n \geq 2$ and n is the same for all states. After replacing α with $\log_n N$, where n is the number of states mapped to a single abstract state and N is the number of states in the original space, the bound on the total work can then be expressed as:

$$Total\ Work \leq \omega n^{(\ln N)/(\ln d)}$$

This function is symmetric in n and N, so we can swap them, giving:

$$Total\ Work \leq \omega N^{(\ln n)/(\ln d)}$$

This shows that if we ignore ω, the total work can be minimized by minimizing d and maximizing n, *i.e.,*by making the maximum distance between any two nodes in an abstract state as small as possible and making the number of states which map to an abstract state as large as possible.

Consider the case where we choose the d as small as possible, *i.e.,* $d = 2$. In this case there is an edge between any pair of nodes that make up the abstract node: the nodes form a clique. Based on the above analysis we need to maximize the number of nodes that are abstracted together; we want cliques that are as large as possible.

Whereas Holte *et al.* were interested in generating an upper bound on the amount of work required to do refinement, we are also interested in finding a measure that will be predictive of the actual work performed. We note that measuring the maximum diameter of the abstraction, d, can be a poor estimate of the cost required to refine a path through a node in practice. For abstractions that are relatively symmetric, such as the clique abstraction, d is likely a good estimate of the cost of refinement. On the other hand, the line abstraction will have $d \approx k$, but there are many short paths through an abstract node, so the average path length through an abstract node will be shorter than d. (This can be verified by Table 1 later in the paper.)

We therefore propose a different measure of d, which we will call d_E. Instead of measuring the maximum distance between states abstracted within a node, we measure the expected cost given that we enter the node from a random edge e_1 and exit from a different edge e_2. For instance, in Figure 6 there are 11 edges at level i by which we can enter or exit the node when doing refinement from level $i+1$ to level i. To enter the edge marked e_1 and exit the edge marked e_2 we would have to traverse a single internal edge. We also add the cost of entering on edge e_1, so this path has cost 2. (Assuming uniform edge costs.) The expected cost to refine the abstract node given that we enter on edge e_1 is $\frac{1\times2+8\times2}{10} = 1.8$. To compute d_E in this example we would perform the same computation for all edges at level i which are external to the abstract node at level $i+1$.

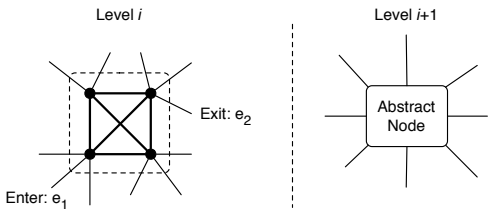

Fig. 6. Measuring the expected diameter of an abstract node.

Finally, we note that this discussion has ignored the ω term, which is the work required to refine a single node. This is directly related to n, so that although we want to increase n to minimize the total work, this has a secondary effect which increases work, so increasing n may not be as effective as decreasing d.

4 Experimental Results

One goal of this paper is to experimentally analyze the previous theoretical results. We begin by measuring properties from the previous section for the different abstractions over 116 maps extracted from Baldur's Gate II (a role-playing game) and Warcraft III (a real-time strategy game). Two examples of these maps can be found in Figure 7. For our experiments we scaled the maps to 512×512 so they are all similarly sized. In Table 1 we list the different abstraction types and the parameters used for each. We measured each of the theoretical properties averaged over all nodes at all levels of the maps.

The values which describe for each abstraction type are not surprising. For instance, the line abstraction ($k = 2$) abstracts, on average, just under two nodes at a time. The uniform abstractions abstract more nodes than the non-uniform abstractions, as seen in Figure 2. The third column in Table 1, Avg. Diameter, is the maximum length of the shortest path between any two states abstracted into a node. For the clique, line ($k = 2$), and sector ($k = 2$) abstractions this value can be at most 1, but is 0 in the cases where a single node cannot be abstracted with its neighbors, resulting in an average value just below 1. We might expect the abstraction with the most nodes, RA(2), to have a diameter of 4, twice the radius. However, since this abstraction is not performed in a uniform manner, its average is lower than the maximum possible.

The final column is the expected number of edges that would be traversed given that we randomly select an incoming and outgoing edge and then measure the number of edges needed to traverse through the node, including the incoming edge. Because we include the incoming edge, we expect this value to be at most 1 larger than the maximum diameter. This value is more indicative of the cost of traveling through a node than the maximum diameter of a node.

Consider, for instance, two nodes abstracted by the line abstraction ($k = 2$). If they both have the same number of edges, we expect that a random path will pass through a single node half the time (cost 1), and pass through both nodes half the time (cost 2). Thus, the expected diameter would be 1.5. The expected diameter is slightly higher

Abstraction Type	Abbrev.	Avg. Nodes (n)	Avg. Diameter (d)	Expected Diameter (d_E)
Clique (Non-uniform)	CA(i)	3.60	0.96	1.72
Clique (Uniform)	CA(n)	3.71	0.97	1.73
Sector 2	SA(2)	3.70	0.98	1.74
Sector 3	SA(3)	7.51	1.89	2.29
Radius 1	RA(1)	8.15	2.36	2.31
Radius 2	RA(2)	10.17	2.80	2.53
NodeLimited 3	NLA(3)	2.57	1.00	1.61
NodeLimited 5	NLA(5)	3.96	1.65	1.86
NodeLimited 6	NLA(6)	4.78	1.84	2.00
Line 2 (Non-uniform)	LA(2, i)	1.86	0.86	1.46
Line 3 (Non-uniform)	LA(3, i)	2.59	1.27	1.67
Line 4 (Non-uniform)	LA(4, i)	3.18	1.65	1.80
Line 5 (Non-uniform)	LA(5, i)	3.58	1.82	1.87
Line 6 (Non-uniform)	LA(6, i)	3.78	1.87	1.87
Line 2 (Uniform)	LA(2, n)	1.97	0.97	1.52
Line 3 (Uniform)	LA(3, n)	2.89	1.86	1.90
Line 4 (Uniform)	LA(4, n)	3.78	2.72	2.22
Line 5 (Uniform)	LA(5, n)	4.50	3.35	2.45
Line 6 (Uniform)	LA(6, n)	5.20	3.97	2.66

Table 1. The abstraction properties for each abstraction type: average nodes abstracted, the average of the max diameter and the average of the expected diameter of an abstract node.

for the LA(2, n) abstraction because we do not consider the possibility of entering and exiting from the same edge, as this can never be part of a refined path. So there is a slightly higher chance of traveling through a node than entering and exiting it directly.

4.1 Abstraction-Based Search Algorithms

In this paper, we use the PRA* algorithm [4] to find paths in each of the abstraction hierarchies. PRA* stands for Partial-Refinement A*, although we use the PRA*(∞) variant, which does full refinement of paths. Given start and goal nodes, PRA* successively maps these nodes into the next higher abstract graph until the abstract start and goal are connected by a single edge. If this occurs at level ℓ in the abstraction hierarchy, PRA* then uses A* at level $\ell/2$ to find a path between the abstract start and goal nodes. A* is used to find paths at successively lower levels of abstraction as well, except that it is constrained to a corridor. The corridor at level ℓ includes all nodes which abstract into either the solution at level $\ell + 1$ or which abstract into a node which neighbors the solution at level $\ell + 1$. PRA* returns paths that are very close to optimal both because it begins planning at the middle of the abstraction hierarchy, and because it widens the corridor used for search.

As a comparison, we use a simple refinement algorithm which follows the theoretical derivation from the last section more closely. This simple refinement algorithm is

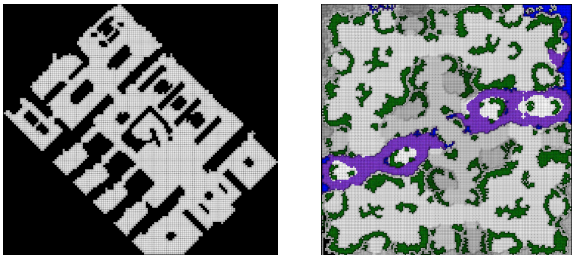

Fig. 7. Two of the maps used in the experiments.

like PRA* except that it starts at the top of the abstraction hierarchy, and does not expand the corridor during the refinement portion of the search. This generally decreases the work that must be done, but results in lower-quality (*i.e.,* longer) paths.

4.2 Search Costs

Given that we have measured the properties of our abstraction, we are interested in measuring the cost of pathfinding using these abstractions and comparing them to see if there is a correlation between the number of abstracted nodes, n, and the diameter of the abstraction, d or d_E.

Our experiments were conducted as follows. We used the same 116 maps as above. On each map, we chose random paths from length 1 to length 512, and placed 10 paths in each of 128 buckets. The first bucket has paths length $(0, 4]$, while the 128th bucket has paths length $(508, 512]$. In total, we have 114,131 paths over all the maps. We then computed a path between these nodes using PRA* and the simple refinement variant of PRA* previously described. We did this for each of the abstractions in Table 1.

In Table 2 we report the average number of nodes expanded in the last bucket (paths of length 508-512) over all maps as well as the total suboptimality over all paths and all maps. Suboptimality is measured as $100 \times (\frac{actual\ path\ length}{optimal\ path\ length} - 1)$, *i.e.,* the percentage difference in length. While there is some correlation between the nodes expanded by simple refinement and by PRA*, there are some differences as well. For instance, the sector abstraction expands fewer nodes (relative to the other abstractions) with simple refinement than with PRA*. This is due to the fact that simple refinement tends to produce paths with less suboptimality in the sector abstraction, and shorter paths are cheaper to refine. This is compared to the non-uniform line abstraction, which produces paths which are up to 80% longer than optimal.

Due to space concerns, we cannot reproduce the full graphs of the nodes expanded for all abstraction mechanisms here. However, we show the 5, 50 and 95th percentile curves for nodes expanded using PRA* using six of the different abstraction mechanisms in Figure 11. Some abstractions, like SA(3), produce a significantly wider spread of best-case and worst-case paths than the clique abstraction. These are percentile curves, so they are different than the average value which we report above. We also show optimality for several abstraction, comparing PRA* and simple refinement in Fig-

	PRA*(∞)		Simple Refinement	
	suboptimality	nodes	suboptimality	nodes
CA(n)	0.04	5301	12.04	1840
CA(i)	0.19	5228	15.81	2062
SA(2)	0.04	5416	6.85	1716
SA(3)	0.02	6232	6.19	1826
RA(1)	0.23	6790	14.76	2540
RA(2)	0.19	6930	11.18	2574
NLA(3)	0.48	5712	40.38	2316
NLA(5)	0.37	5627	27.43	2158
NLA(6)	0.34	5533	23.13	2137
LA(2, n)	0.50	6598	19.05	2378
LA(3, n)	0.95	6092	22.62	2238
LA(4, n)	0.61	6119	22.60	2359
LA(5, n)	0.60	6188	27.62	2473
LA(6, n)	1.07	6764	30.23	2693
LA(2, i)	0.35	7291	81.84	3248
LA(3, i)	0.38	6075	68.33	2261
LA(4, i)	0.39	5914	62.01	2215
LA(5, i)	0.49	5871	60.55	2213
LA(6, i)	0.56	5936	58.42	2269

Table 2. A comparison of suboptimality with PRA*(∞) and simple refinement.

ure 12. These graphs show the worst quality path found, as well as the quality of the 99.5, 98, 95 and 50th percentile. The 50th percentile line is not visible for PRA* and falls just above the x-axis for simple refinement.

4.3 Predicting Total Work

We can now address the main point of our experimental results, which is to test whether the average number of nodes expanded, n, and the diameter of an abstraction, d or d_E, can be used to predict the total work needed to compute a path using an abstraction. For the moment, we just consider d_E.

First, recall that we are trying to minimize d_E and maximize n. In order to simplify our analysis, we look instead at the problem of minimizing both d_E and $\frac{1}{n}$. We normalize d and $\frac{1}{n}$ over all abstractions to the range $0 \ldots 1$ and plot them in Figure 8. The size of the point for each algorithm scaled by the number of nodes expanded by PRA* on this abstraction. The figures are the same, the left one just includes labels of the data points. By looking at sequences of algorithms, one can see the effect of the abstraction parameters. The algorithms which abstract fewer nodes are found in the bottom right corner, while the algorithms which abstract more nodes are a time are in the top-left of the figure.

Because the total work is predicted to be correlated with both measures, we use the distance from each point in this chart to the origin of the graph (which attempts

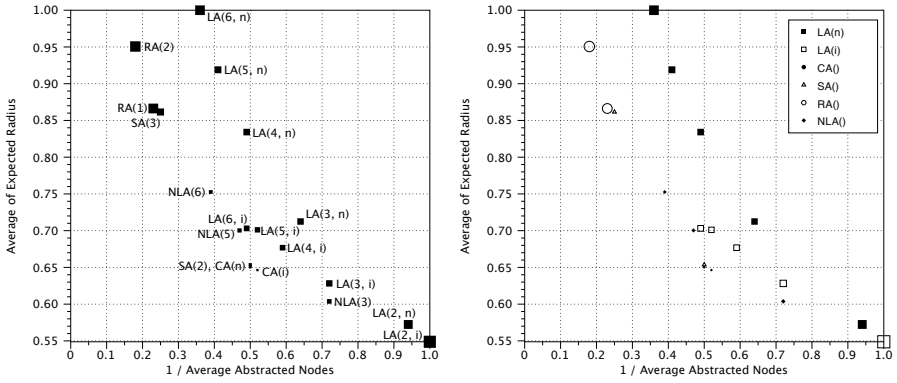

Fig. 8. The trade-off between the size of each abstraction and the diameter.

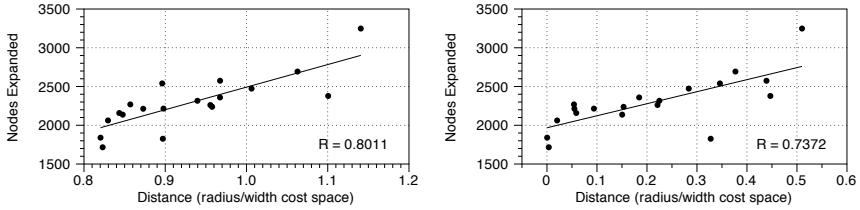

Fig. 9. The correlation between abstraction parameters and work done using simple refinement. The graph on the left measures the distance between the abstraction parameters relative to the origin, while the graph on the right measures the distance relative to the CA(n) parameters.

to minimize both measures equally) as a measure of the abstraction parameters. Since the parameters may not be equally weighted, we also considered the distance from the clique abstraction parameters as a secondary measure.

We plot these distances against the nodes expanded by simple refinement in Figure 9 and against the nodes expanded by PRA* in Figure 10. If these values are well-correlated, then the measures of n and d are accurate in predicting total work. The best-fit line for search by the simple refinement algorithm has a correlation coefficient 0.80 using the distance from origin metric and 0.74 using the distance from the clique abstraction parameters. The best-fit line for work done by PRA* has a correlation of 0.81 when using the distance from the origin metric, and 0.92 when using the distance from the clique abstraction parameters.

Returning to the issue of whether it is better to use the maximum diameter, d, or the expected diameter, d_E, we ran the same predictions using both definitions of the diameter and distance. We found that the correlation between the best-fit line and the data was more accurate when using the expected diameter instead of the maximum diameter for both algorithms and distance measures.

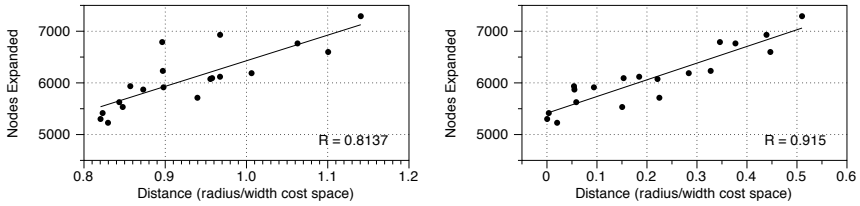

Fig. 10. The correlation between abstraction parameters and work done using PRA*. The graph on the left measures the distance between the abstraction parameters relative to the origin, while the graph on the right measures the distance relative to the CA(n) parameters.

5 Conclusions

We have shown that the total work done when using a graph abstraction for search and refinement can be predicted by two parameters, the diameter of abstract nodes, d, and the total number of nodes abstracted into each abstract node, n. Thus, the theoretical predictions of Holte *et al.* [2] are useful in practice as well. We have also shown that the clique abstraction's parameters are well suited for minimizing computation, particularly for PRA*.

Additionally, we have introduced a new method for computing the expected diameter of an abstract node, d_E, and have shown that this is more accurate than using the maximum diameter, d. Our analysis has also highlighted how small changes in how an abstraction is built can influence the abstraction measures, n and d. This was shown in the differences between uniform and non-uniform clique and line abstractions.

There are two specific areas for future research. First, we would like to expand these results beyond the pathfinding domain. Secondly, we would like to better understand the manner in which suboptimality is affected by the choice of abstraction and how it influences the predictions of how much work must be done. While we have seen that there is some influence, we have yet to describe this influence in detail.

References

1. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4(2)** (1968) 100–107
2. Holte, R.C., Mkadmi, T., Zimmer, R.M., MacDonald, A.J.: Speeding up problem solving by abstraction: A graph oriented approach. Artificial Intelligence **85** (1996) 321–361
3. Botea, A., Müller, M., Schaeffer, J.: Near optimal hierarchical path–finding. Journal of Game Development **1(1)** (2004) 7–28
4. Sturtevant, N.R., Buro, M.: Partial pathfinding using map abstraction and refinement. In: AAAI. (2005) 1392–1397
5. Fernandez., A., Gonzalez, J.: Multi-Hierarchical Representation of Large-Scale Space. Kluwer (2001)
6. Yang, Q., Tenenberg, J., Woods, S.: On the implementation and evaluation of ABTweak. Computational Intelligence Journal **12** (1996) 295–318

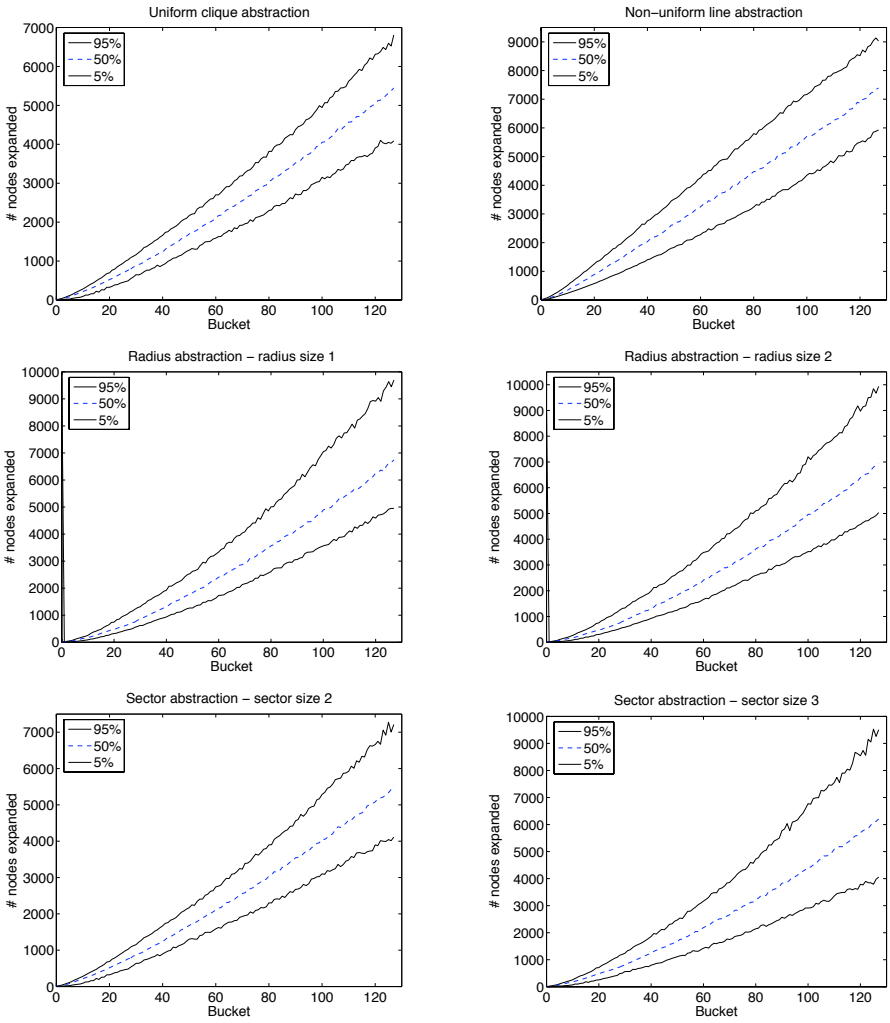

Fig. 11. Nodes expanded by 6 of the different abstraction mechanisms.

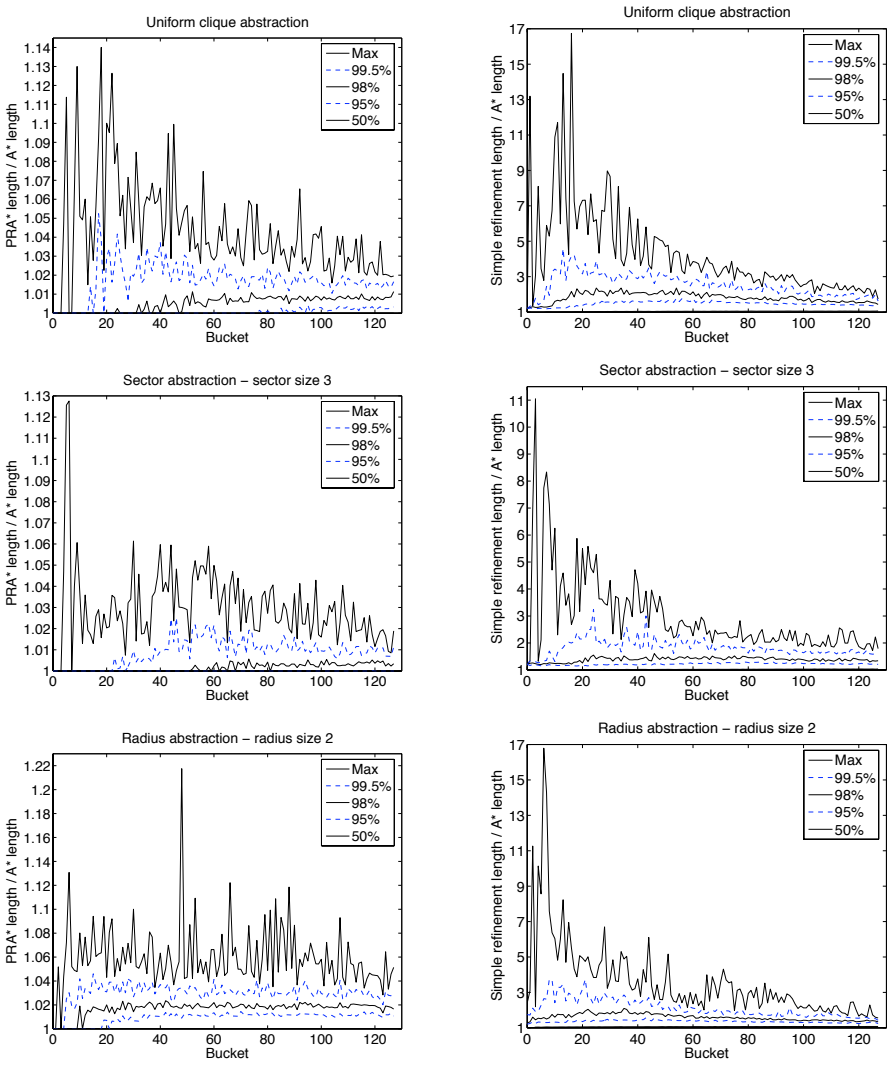

Fig. 12. Path optimality using 3 of the different abstraction mechanisms and PRA*(∞) [left] versus simple refinement [right].