The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

FAHR: Focused A* Heuristic Recomputation

Matthew McNaughton and Chris Urmson

Abstract— **In this paper we introduce Focused A* Heuristic Recomputation (FAHR), an enhancement to A* search that can detect and correct large discrepancies between the heuristic cost-to-go estimate and the true cost function. In situations where these large discrepancies exist, the search may expend significant effort escaping from the "bowl" of a local minimum. A* typically computes supporting data structures for the heuristic once, prior to initiating the search. FAHR directs the search out of the bowl by recomputing parts of the heuristic function opportunistically as the search space is explored. FAHR may be used when the heuristic function is in the form of a pattern database. We demonstrate the effectiveness of the algorithm through experiments on a ground vehicle path planning simulation.**

I. INTRODUCTION

Good heuristic estimates are necessary for effective search. One typically computes a function $h(n)$ based on the problem instance that approximates the cost from n to the goal. The heuristic function can take a variety of forms. It may be static with respect to all problem instances. For example, straight-line distance is often used as a heuristic for the shortest path through a Euclidean space. It can also be computed on a per-problem basis. In the case of path planning, for example, this can be done by computing a single-source shortest path table through a lower-dimensional search space formed by a simplification of vehicle kinematics or dynamics. In either case, once the heuristic data structure is computed, its $h(\cdot)$ values typically do not change until a solution is found or declared not to exist. In this paper we show that it can be beneficial to recompute the heuristic function $h(\cdot)$ using information obtained about the search space while the search is underway, re-evaluate the open list using the new $h(\cdot)$, and continue the search with the refined heuristic. We validate our approach using a path planning simulation for a ground vehicle.

The organization of this paper is as follows. In Section II we discuss related work in heuristic search and path planning. In Section III we describe the FAHR algorithm. In Section IV show how the technique is applied in a vehicle motion planning context. Section V gives experimental results, and Section VI concludes with a discussion and ideas for future research.

II. RELATED WORK

Culberson and Schaeffer[1] used the term *pattern database* to describe a heuristic that takes the form of a table containing the shortest cost-to-go of any state in a subspace of the problem. They applied their technique to the 15-puzzle and Checkers. Korf[10] applied it to the Rubik's cube. These works precomputed large databases that could be used with all problem instances. The work we present in this paper may be seen as an attempt to recompute pattern databases online for a single problem instance based on information obtained during the search.

Holte et al. describe Hierarchical A*[7] and IDA*[6], which implement the estimated cost-to-go $h(\cdot)$ by searching for the true cost-to-go $h^*(\cdot)$ in an abstracted version of the state space. A side-effect is the creation of a pattern database tuned to a particular problem instance which can be reused for other problem instances, depending on the abstraction. The present work can be seen in contrast as re-evaluating existing entries in the pattern database based on information obtained at lower levels of the abstraction.

Zhou and Hansen[14] define the *focused memory-based heuristic* as a heuristic table with entries that are computed for just the nodes explored by A* in solving the search problem. They use a multi-resolution heuristic approach to focus effort into computing more accurate $h(\cdot)$ values for states that are more likely to be on or near the optimal path. Our work differs in attempting to focus effort on regions of the heuristic table that may direct search effort to unproductive parts of the search space.

Other authors have addressed the phenomenon FAHR is concerned with, namely that an A* search may spend a lot of time expanding all nodes within a bowl of search space, in an effort to prove that there is no way through the drain. Junghanns and Schaeffer[8] introduce *relevance cuts* to address the issue that in a combinatorial problem with multiple subgoals, the search algorithm will try all transpositions of moves that advance independent subgoals. Relevance cuts attempt to characterize which moves are independent from each other and enforce that only moves that are affected by the previous one should be considered. They demonstrated results using the Sokoban puzzle, but optimality is guaranteed only with IDA*. Our work differs in that pathfinding for a single vehicle does not break down into independent subproblems, so we look instead for areas of the search space where the heuristic function is highly misleading. Our work also differs in that it allows edge costs to be non-uniform, whereas results for combinatorial games often depend on a uniform cost of 1 for all actions.

Incremental replanning techniques such as Dynamic A*(D*)[13] and D* lite[9] were developed in the context of path planning, which often involves repeated searches to the same goal on similar graphs. They re-use data from the open and closed lists from previously completed searches

Matthew McNaughton and Chris Urmson are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA mmcnaugh@ri.cmu.edu, curmson@ri.cmu.edu

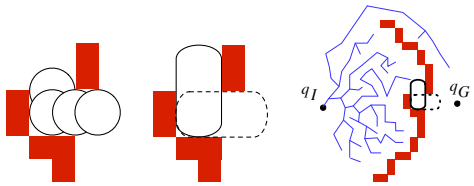

Fig. 1. A gap (a) that the ball heuristic suggests may be passable, but (b) cannot be navigated by the real vehicle, and (c) the "bowl" of search space explored as a result

on similar problem instances. These algorithms depend on the assumption that if the search space has only changed a little, then much of the computation should be reusable. We develop our work in the context of path planning, but focus on accumulating changes in the heuristic table during a single search.

In the next section we formally describe the problem we are addressing in this work, and introduce our proposed algorithm.

III. Approach

A. Problem Description

We term the *bowl effect* in heuristic search as the problem that a large region, or "bowl" of configuration space must sometimes be explored before all states within the bowl have been examined, so that the search can overflow the bowl and continue towards the goal.

Figure 1 illustrate how an A* search in vehicle path planning can encounter a bowl effect if the heuristic used suggests that it is possible to navigate a narrow gap, when in fact it is not. The A* search must examine all states within the bowl in order to prove that there is no way through the gap. In this case, the heuristic is based on a ball approximating the vehicle. Later in the paper we will show how this heuristic is constructed and demonstrate that it is efficient to compute and admissible for A* search. The bowl is caused by a large discrepancy between the heuristic function and the true cost function. Our proposed method detects the presence of these bowls early in the search, and modifies the heuristic to direct the search to more promising areas of the space.

In the next section we formalize search on a graph and describe the property that we exploit to detect bowls.

B. Formulation

An optimal search proceeds to find the lowest-cost path from a given start node q_I to a given goal node q_G on a graph $G = (V, E)$ with edge costs $c : V \times V \rightarrow \mathbb{R}^+$. We define $c(u, v) = \infty$ for edges $(u, v) \notin E$, indicating that the edge cannot be part of a valid path.

To achieve a provably optimal result in A* and derived algorithms such as D*, the heuristic function $h(n)$ estimating the cost from node n to the goal node q_G must be *admissible*, meaning that $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost of the minimum path from the node n to the goal. A* examines nodes it encounters in order of increasing $f(n) = g(n) + h(n)$, where $g(n)$ is the proven minimum cost to reach n from q_I. This makes $f(n)$ an under-estimate of the total cost of the shortest path from q_I to q_G that passes through n.

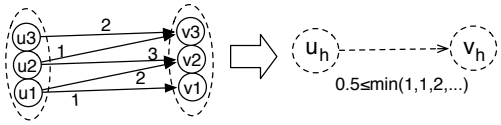

Fig. 2. Projection of the search graph to the heuristic graph

For some problems, the graph is a discrete subset of a continuous metric space. For example, in robot path planning, the search graph is a discretization of $\mathbb{R}^2$. In these cases the heuristic can be a function derived directly from the metric space, rather than indirectly through the search graph. For example, the straight-line Euclidean distance $d(n, q_G)$ from n to the goal q_G is independent of the graph chosen to discretize the space for the search.

One can also use a direct relaxation of the search graph G as a heuristic. For example, one can construct a smaller heuristic graph $G_h = (V_h, E_h)$, where $p : V \rightarrow V_h$ for a chosen V_h with $|(V_h)| \leq |(V)|$, and

$$E_h = \{(p(u), p(v)) : (u, v) \in E\}.$$

The cost $c_h(u, v)$ must satisfy

$$c_h(u_h, v_h) \leq \min_{u,v \in V} \{c(u, v) : u_h = p(u), v_h = p(v)\}.$$

Figure 2 illustrates with a search graph. The full search graph consists of the sets of nodes $\{u_i\}$, $\{v_j\}$, and the edges between them. The projected heuristic graph contracts the $\{u_i\}$ and $\{v_j\}$ into the single nodes u_h, v_h with a single edge between them having a lower cost than any of the edge costs $c(u_i, v_j)$. In the figure, if $c(u_h, v_h) \lll \min_i c(u_i, v_i)$, then the heuristic is misleading, and is potentially creating a bowl.

In the next section we introduce FAHR, an algorithm to detect areas where this condition holds, with the purpose of recomputing the heuristic table to bring $c(u_h, v_h)$ closer to $\min_i c(u_i, v_i)$. Doing this should decrease the amount of search effort required to find the goal.

Increasing the value of $h(q)$ for some nodes q while retaining admissibility may reduce the number of nodes expanded by A* and can never increase it[12]. Modifications of $h(\cdot)$ come with a time cost, however. This cost must be less than the time saved by expanding fewer nodes in order for FAHR to yield a benefit. In a later section we show how we applied FAHR to a path planning problem, and demonstrate its efficiency.

C. The Focused A* Heuristic Recomputation Algorithm

The Focused A* Heuristic Algorithm (FAHR) is an extension to the standard A* algorithm[12] that finds bowls in the search space. Figure 3 illustrates the basic idea behind the algorithm using a small search tree, growing from the left to the right. In part (a), nodes y and v are on the open list (circles), and v is a successor of x, which is on the

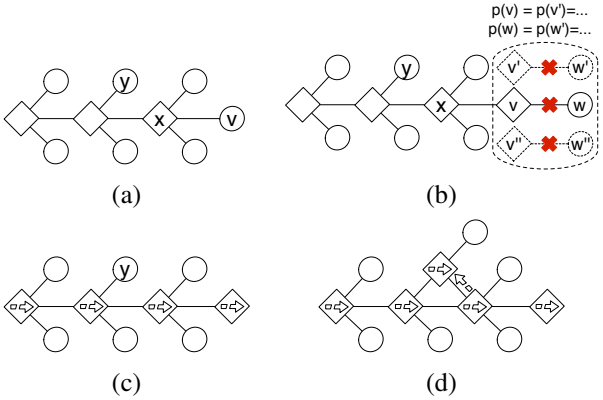

Fig. 3. Illustration of bowl detection in the A* search tree structure

closed list (diamonds). In (b), the node v happens to have the lowest f-value and is selected for expansion. The cost to reach its potential successor w is very high, contrary to expectation. Edges (v', w') and (v'', w'') are identified as all of the edges that project to the same edge in the heuristic table, i.e. $(p(v), p(w)) = (p(v'), p(w')) = (p(v''), p(w''))$. It is found that all of these edge transitions are actually much more costly than the heuristic estimate, and so it can be concluded that the cost of the edge $(p(v), p(w))$ as given by the heuristic is misleading. In (c), nodes leading from v up to the root node are marked as being part of a bowl, denoted by the broken arrow symbol. In (d), y is expanded, and one of its successors is x, which was already visited in the search and is known to be in a bowl. Consequently, y is marked as being part of the bowl, as are its predecessors up to the root. In this illustration, the immediate predecessor of y was already marked as being part of a bowl. Once the number of nodes in the bowl has grown beyond a chosen threshold, the state space in the neighborhood of v can be analyzed more deeply to find the cause of the bowl, and the heuristic can be recomputed so that the search is directed out of the bowl.

The FAHR algorithm itself is given in Figure 4. The main body of the algorithm is similar to A*, with the additions defining FAHR marked FAHR. Line 11 gives each node a pointer to its parent so that the chain of nodes in the bowl can be identified. It is common for A* implementations to keep this pointer for convenience in reconstructing the solution path, but FAHR requires it. Line 13 checks to see whether the heuristic is highly misleading for the child v' of v, the node being expanded. If this test succeeds, it means that $c(v, v')$ is much more expensive than the heuristic had predicted. The checkBowl() function examines all nodes that use the same edge in the heuristic graph, and if the heuristic severely underestimates the actual edge cost in all cases, then the given edge induces a bowl. Line 15 the child node v' of v is checked to see if it has already been expanded and been marked as a member of a bowl, in which case v and its parents are also marked in the bowl. If an edge is found to be blocked(Line 14), then the edge is recorded in blockedE, and incBowl() is invoked to mark all nodes from v up to the root of the search graph as being in the bowl. The nodes in E' are not added since they are not necessarily part of the search graph. In the final line of checkBowl(), the size of the bowl is checked against a tuned threshold, and the heuristic is recomputed if the threshold is exceeded. The heuristic recomputation step is highly problem-dependent. We will offer an example in the next section.

IV. Application of FAHR to Path Planning

In the previous section we described the general FAHR algorithm. In this section we describe how it can be applied to a path planning problem for a simulated ground vehicle.

For the problem of path-planning for a mobile robot, one possible search graph formulation is an 8-connected grid, where the robot may take 8 unique orientations at each grid position. See Figure 5(a). A widely used heuristic is to approximate the vehicle with the largest ball (in two dimensions, a circle) that fits inside the vehicle. Figure 5(b,c) illustrates. For this heuristic, the connectivity is simply that of an 8-connected grid. It is easy to see that any path that can be followed by a vehicle of arbitrary shape with arbitrary kinematics can also be followed by a ball that fits entirely inside the vehicle and can move in any direction. Therefore, a heuristic based on this projected graph is admissible. This "ball" heuristic is commonly used by path planning solvers for navigating through unstructured environments with obstacles[5], [3]. To construct the table, we first compute a distance transform on the occupancy grid of obstacles in the environment. This yields a table $d_{\mathcal{O}}(x, y)$ with the distance from each grid cell to the closest obstacle. An algorithm such as [4] can compute $d_{\mathcal{O}}$ in about 10 ms on a typical PC for an occupancy grid of size 400×400. From this table, a graph G_h is constructed, with one vertex per cell in the table. Cells are 8-connected and an edge between cells(vertices) is traversable if each of the cells (x, y) satisfies $d_{\mathcal{O}}(x, y) >= r_1$. Edges into cells where $d_{\mathcal{O}}(x, y) < r_1$ are given infinite cost. r_1 is the radius of the largest ball that fits entirely inside the vehicle. In this example, traversable horizontal and vertical edges have a cost of 1, and diagonal edges cost $\sqrt{2}$. Other works in path planning allow higher costs to be used in order to reflect that some terrain takes longer to traverse.

Finally, a single-source shortest path(SSSP) algorithm such as Dijkstra's[2] is run on G_h to get a table that gives the cost of the shortest path for a ball to travel from any node n to the goal q_G. We approximated edge lengths with integers, using 5 for horizontal and vertical edges and 7 for diagonal edges. This allowed us to use an $O(n)$ implementation of Dijkstra's algorithm. The heuristic table formed is still admissible.

Figure 5(b,c) show how the ball heuristic can allow edges into the heuristic graph where no edge is in the full search graph. In (b), the ball of radius r_1 does not collide with any obstacles, even though the vehicle cannot be placed in any orientation at that spot without collisions. In (c) the heuristic graph subsequently marks the lower-right circle as reachable.

Figure 6(a) shows the discretized kinematics of our sample vehicle in the full search space. The vehicle may move

```
FAHR(node s, node t, graph G = (V, E) ) : node
1:   ∀n, f(n)←∞; f(s)←0; OPEN = {s}
2:   while |OPEN| > 0
3:       v ← argmin_{v∈OPEN} f(v)
4:       if v = t then return A* ←v
5:       OPEN ← OPEN\ {v}
6:       CLOSED ← CLOSED ∪ {v'}
7:       for (v, v') ∈ E
8:           g' ← g(v) + c(v, v'), f' ← g' + h(v')
9:           if f' < f(v') then
10:              f(v') ← f', g(v') ← g'
11:              parent(v') ← v                          FAHR
12:              OPEN ← OPEN ∪ {v'}
13:          if f(v) ⋘ f(v') then                        FAHR
14:              checkBowl(v, v', g)                     FAHR
15:          if v' ∈ bowlV then                          FAHR
16:              incBowl(v)                              FAHR
       end for
   end while

checkBowl(edge (v, w), graph G=(V,E)) : void
   // All edges that project to the same edge as
   // (v, w) in the heuristic graph
17:  E' = {(v', w') ∈ E : p(v') = p(v), p(w') = p(w)}
18:  E'' = {(v', w') ∈ E' : c(v', w') ⋙ h(w') − h(v')}
19:  if E' = E'' then // h(·) misleading in all cases
20:      blockedE = blockedE ∪ E'
21:      incBowl(v)
22:      if |bowlV| > bowlCutoff
23:          recomputeHeuristic()

incBowl(node v): void
24:  if v ∉ bowlV
25:      bowlV = bowlV ∪ {v}
26:      incBowl(parent(v))

recomputeHeuristic(): void
   // Computation of Ĥ(blockedE) is problem-dependent.
   // blockedE is used to identify areas of
   // the heuristic graph for more analysis.
27:  h() ← Ĥ(blockedE)
28:  ∀v ∈ OPEN, f(v)←g(v) + h(v)
```

Fig. 4. A* search augmented with bowl detection

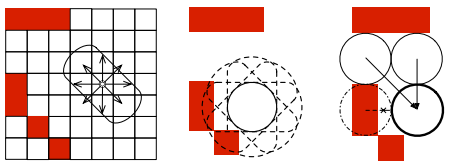

Fig. 5. (a): An eight-connected grid for vehicle motion planning. (b): A ball heuristic on the grid for vehicle motion planning. (c): Connectivity of the ball heuristic in the presence of obstacles.

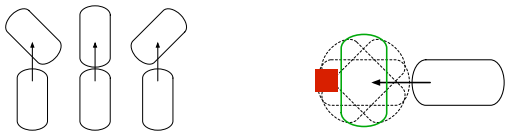

Fig. 6. (a) Simple kinematics of the simulated vehicle, and (b) a non-navigable set of edges in the full search space that has a valid corresponding edge in the heuristic graph.

forward in the direction it is currently facing and optionally rotate in either direction by 45 degrees.

A. Eliminating the Bowl

The recomputeHeuristic() function of the FAHR algorithm(Figure 4) requires that the heuristic be recomputed. We do this by analyzing in more detail the edges of G_h. Figure 6(b) shows an edge that would be marked traversable in G_h even though the vehicle cannot make this move according to the kinematics of the simulation specified in Figure 6(a). That is, $c(u_h, v_h) = 1 \leq \min_i c(u_i, v_i) = \infty$ as per Figure 2, where $(u_h, v_h) \in V(G_h)$ and $(u_i, v_i) \in V(G)$. The fact that $c(u_h, v_h) \lll \min_i c(u_i, v_i)$ is what induces the bowl. To fix this discrepancy, we analyze all edges (u_i, v_i) in G such that $(p(u_i), p(v_i)) = (p(u_h), p(v_h))$. Using these edges we compute $\hat{c}(u_h, v_h) = \min_i c(u_i, v_i)$, and reset the edge cost of $(u_h, v_h) = \hat{c}(u_h, v_h)$ in G_h to obtain a new $G_{\hat{h}}$. The SSSP algorithm is re-run on $G_{\hat{h}}$ and the f-values of the open list are reset using the new $h(\cdot)$. We look for edges (u_h, v_h) in a small radius around the edges of blockedE, since often there are more nearby than just the ones that have been directly encountered.

In the next section we present experiments that characterize the performance of FAHR on the vehicle path planning problem.

V. Experimental Results

We consider the factors that affect the performance of the search before moving on to experimental results. As with all heuristics and variations on optimal search algorithms, the benefit to be gained from using the technique varies depending on the conditions in which it is used. Following are some of the factors influencing the performance of FAHR.

- The size of the typical bowl encountered affects the time savings gained by recomputing the heuristic.
- The cost of computing the upgraded heuristic in the entire space before the search begins. This depends on the size of the entire search space and the cost of computing the upgraded heuristic in a unit area.
- Fixed costs of the heuristic recomputation, such as the SSSP algorithm, affect the ratio of the size of area of the heuristic grid to be upgraded to the size of the whole heuristic grid.

Assuming relatively small pattern databases, overall memory usage will tend to be lower whenever improved heuristics are used due to the decreased growth of the A* open list. In

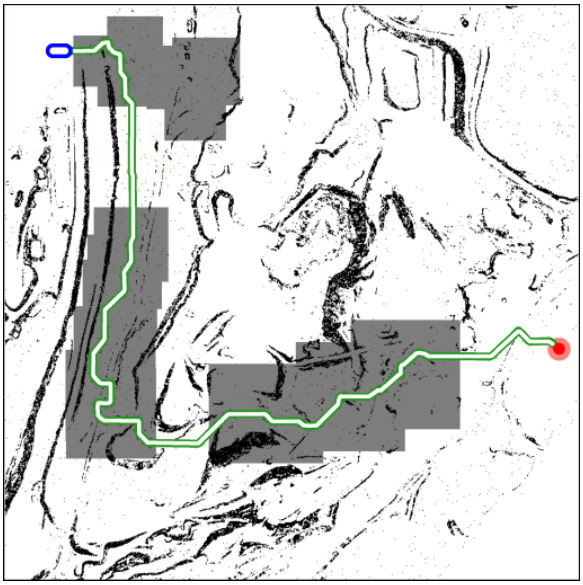

Fig. 7. Aerial view of map used for path-planning experiments. The grid is 2236x2236 cells. The grayscale values were thresholded to a binary obstacle map representation. An example of an optimal path and the areas recomputed by FAHR are also shown.

path planning for mobile robots, we are mainly concerned with overall CPU time used for the search, rather than memory usage. For other applications, we could expect that the potential for gain from using heuristic recomputation should increase with the running time of the search.

A. Description of Experiments

We obtained cost map data from an aerial image of land approximately 2 km^2 in size from the DARPA UPI Program, sampled at a resolution of 60 cm. The final cost map used in these experiments was a grid of size 2236×2236 cells. The terrain has a variety of roads which connect to open areas that tend to be scattered with small obstacles. Figure 7 gives a sketch of the landscape. White areas are traversable, and black areas are obstacles. We thresholded the grayscale image into a binary cost map. We used a vehicle model of 13 cells wide by 44 cells long. We randomly sampled 229 (start,goal) location pairs out of all occupiable cells on the map and executed three search algorithms in turn:

- PLAIN - Normal A* search with the basic ball heuristic. This serves as a baseline to compare the other approaches.
- FAHR - FAHR with bowlCutoff = 3000. A bowlCutoff value that is too small will result in frequent invocations of recomputeHeuristic() to remedy small bowls, and one that is too large will allow A* to expand too many of the nodes in the bowl before intervening, with a corresponding reduction in the potential savings. The area of the heuristic table upgraded in the call to recomputeHeuristic() was a square of size 350×350. The side length of the area recomputed at each call was increased by a factor of 1.1. These values were arrived at by brief experimentation. We shall explore these values in a future work. Since binary obstacles were used where the cost of an obstacle collision is infinite, the "$\lll$" relation on Line 13 of the algorithm (Figure 4) holds iff $f(v') = \infty$. Future work will address tuning the "$\lll$" relation in the case of non-binary costs.
- PRECOMPUTE - Normal A* search, with the upgraded heuristic used by the FAHR search algorithm simply precomputed over the entire space.

TABLE I

MEANS AND STANDARD DEVIATIONS OF TIMES TAKEN AND NODES EXPANDED OVER ALL 229 TEST CASES BY THE THREE ALGORITHMS.

Algorithm	μ time (s)	σ time (s)	μ nodes	σ nodes
PLAIN	10.0	12.4	1.4×10^6	1.9×10^6
FAHR	7.1	5.7	1×10^5	1.4×10^5
PRECOMPUTE	16.2	0.7	5×10^4	8×10^4

Table I summarizes the mean and standard deviations of times taken by the these three algorithms. PRECOMPUTE is consistently slow since computing the upgraded heuristic is slow, but searching with it is fast. PLAIN is faster on average than PRECOMPUTE but has widely variable times. FAHR is fastest on average and the variability is significantly lower than PLAIN.

Figure 7 shows the optimal path found in a sample case. The grey boxes show regions of the heuristic that were recomputed by FAHR. The heuristic is initially recomputed in regions at the top left, as the unenhanced ball heuristic leads the search directly to the right of the start position. This creates a bowl in the search space since the gap through the wall is not traversable. Once FAHR recomputes the heuristic in the area, the search proceeds downwards and occasionally recomputes the heuristic in areas of scattered obstacles and additional non-navigable gaps. In this example, FAHR recomputed the heuristic 9 times and took 13 seconds versus 16 for plain A*.

The approximate time taken by each step of the search process on the sample grid shown in Figure 7 is as follows

1) Distance transform - 0.5s. This time is not counted in the results.
2) Kinematic analysis for upgraded heuristic - 16s for the entire grid. Varies by obstacle density. This analysis is performed by FAHR on sections of the grid near blocked edges during the recompute step.
3) SSSP - 0.7s. This is performed by fahr at each recomputation.
4) A* search - highly variable.

Figure 8 shows a scatter plot of the times taken by FAHR versus PLAIN on the test cases. Points below the diagonal line are cases where FAHR was faster, and points above the line are slower. On very easy cases, FAHR costs little to nothing extra since recomputation is not invoked, but on hard cases, FAHR shows significant benefit. FAHR rarely takes significantly longer than PRECOMPUTE.

Figure 9 shows the number of times recomputation was performed by FAHR, against the total time taken. The relationship is approximately linear, possibly because the work

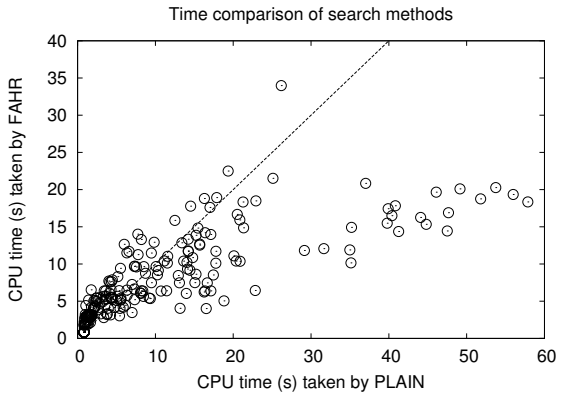

Fig. 8. Scatter plot of time taken by FAHR compared to PLAIN on 229 test cases.

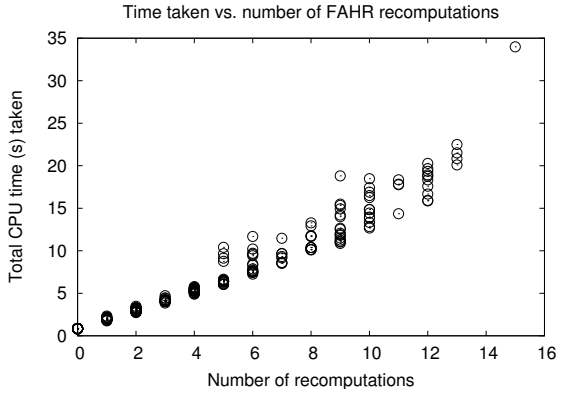

Fig. 9. Scatter plot of time taken by FAHR, versus number of recomputations performed by FAHR.

performed by each recomputation is roughly constant. The SSSP always takes the same amount of time. The area of the heuristic that is recomputed is grown by a factor of 1.1 on each call to recomputeHeuristic(), which may explain the slight super-linear trend. The upgraded heuristic is very effective so most of the time taken by the A* search loop to expand nodes is consumed by filling up bowls.

VI. Discussion and Future Work

In this paper we contributed the Focused A* Heuristic Recomputation (FAHR) algorithm for identifying local minima, or "bowls" in the search space induced by highly misleading heuristics. We also contributed an observation of the relationship between the lattice search space used in vehicle path planning, and the simplified lattice used as a heuristic.

Future work in this area leads in several directions. We can

- Identify additional problem domains that can benefit from dynamic heuristic recomputation. This can include not only higher-dimensional domains in vehicle motion planning, but any A* search problem that can use a pattern database as a heuristic, such as multiple sequence alignment[11]. As the dimensionality of a planning problem increases, the benefit of identifying and closing non-navigable gaps in the heuristic should increase. However, the number of edges in G that project to the same edge in G_h will also increase exponentially.
- Apply the heuristic structure to replanning problems such as path planning with D*[13]. Replanning involves multiple searches performed sequentially in similar obstacle fields. Implementations typically recompute $h(\cdot)$ each time new data is obtained. Regions of the heuristic identified in earlier searches for closer kinematic analysis can be analyzed immediately, preserving the reduction of effort across replanning runs.
- We shall investigate the effect of bowlCutoff values on planning efficiency, as well as the size of the region of interest that should be formed around the edges in blockedE.
- Experiment with parallel implementations - recomputeHeuristic() can run in parallel with the main search, and the new heuristic used when it is ready.

VII. Acknowledgments

This work was made possible by a grant from General Motors Corp. We thank David Silver and the DARPA UPI Program for providing obstacle map data for our experiments. We also thank the anonyomous reviewers for their thoughtful comments.

References

[1] J. Culberson and J. Schaeffer. Searching with pattern databases. In *CSCSI '96 (Canadian AI Conference), Advances in Artificial Intelligence*, pages 402–416. Springer-Verlag, 1996.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[3] D. Dolgov, S. Thrun, , M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*. AAAI, June 2008.

[4] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical Report TR2004-1963, Cornell Computing and Information Science, 2004.

[5] D. Ferguson, T. Howard, and M. Likhachev. Motion planning in urban environments: Part ii. In *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems*, September 2008.

[6] R. C. Holte, J. Grajkowski, and B. Tanner. *Abstraction, Reformulation and Approximation*, volume 3607 of *Lecture Notes in Computer Science*, chapter Hierarchical Heuristic Search Revisited, pages 121–133. Springer, 2005.

[7] R.C. Holte, M.B. Perez, R.M. Zimmer, and A.J. MacDonald. Hierarchical a*: Searching abstraction hierarchies efficiently. In *AAAI*, pages 530–535, 1996.

[8] A. Junghanns and J. Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1-2):219 – 251, 2001.

[9] S. Koenig and M. Likhachev. D* lite. In *Proceedings of the AAAI Conference of Artifical Intelligence (AAAI)*, pages 476–483, 2002.

[10] R. E. Korf. Finding optimal solutions to rubik's cube using pattern databases. In *AAAI-97*, pages 700–705, 1997.

[11] M. McNaughton, P. Lu, J. Schaeffer, and D. Szafron. Memory-efficient a* heuristics for multiple sequence alignment. In *Proc. Eighteenth National Conference on Artificial Intelligence (AAAI)*, July 2002.

[12] N. J. Nilsson. *Principles of Artificial Intelligence*. 1980.

[13] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 1994.

[14] R. Zhou and E. Hansen. Space-efficient memory-based heuristics. In *19th National Conference on Artificial Intelligence (AAAI-04)*, 2004.