Evaluating FastMap and Hybrid Heuristics for Fast Grid Pathfinding

Masoud Jafaripour
MSc Student, CS of UofA
jafaripo@ualberta.ca

Instructor: Dr. Nathan Sturtevant
Professor, CS of UofA
nathanst@ualberta.ca

Abstract

We present a comparative evaluation of heuristic strategies for grid-based pathfinding with A* search, focusing on FastMap and hybrid methods that integrate FastMap with Differential Heuristics (DH). FastMap constructs low-dimensional Euclidean embeddings of graph nodes, enabling admissible heuristics based on ℓ_1 distances. Although compact and efficient, FastMap alone may underperform in complex environments. Recent work addresses this limitation by combining FastMap with DH to improve heuristic accuracy and reduce node expansions. Building on this, we implement and evaluate several strategies, including additive FM+DH combinations and enhanced variants that use heuristic error (HE)-based pivot selection to diversify embeddings. Experiments on benchmark maps from the MovingAI dataset show that these hybrid heuristics significantly reduce node expansions compared to standalone DH and FastMap. Our findings support the effectiveness of hybrid embeddings for scalable and optimal pathfinding.

1 Introduction

Efficient path planning is a fundamental challenge in artificial intelligence, robotics, and game development. Algorithms like A* [8] are widely adopted due to their optimality and generality, but their efficiency depends heavily on the choice of heuristic used to guide the search. In large grid-based domains, strong admissible heuristics can dramatically reduce node expansions and computation time.

A large body of research has developed heuristics that approximate shortest-path distances while remaining fast and space-efficient. Differential Heuristics (DH) [6, 14] use precomputed landmark distances and the triangle inequality to estimate cost-to-go. More recently, FastMap [3], adapted from the data mining domain [5], has emerged as an embedding-based alternative. FastMap projects nodes into a low-dimensional Euclidean space using shortest-path distances to selected pivot pairs, yielding admissible and consistent ℓ_1 heuristics.

Although FastMap offers compact representations and supports downstream tasks like clustering or multi-agent coordination, its standalone performance often lags behind DH in complex grid environments. Mashayekhi et al. [10] addressed this by introducing hybrid strategies—such as FM+DH (additive fusion) and max fusion (e.g., $\max\{\text{FM+DH}, \text{DH}\}$)—that combine the strengths of both methods. To further improve heuristic quality, they proposed a pivot selection technique based on heuristic error (HE), which chooses pivots that expose underestimations in the current heuristic, thereby increasing diversity across embedding dimensions.

In this project, we implement and evaluate a range of heuristic strategies within the A* framework across various grid-based domains. These include FastMap with embedding-distance (ED) and heuristic-error (HE) pivoting, classical DHs, and multiple hybrid methods with additive or max-based fusion. We adopt the Generalized Metric Embedding (GME) framework to unify these strategies and analyze their behavior in terms of node expansion, preprocessing cost, and generalization.

By systematically evaluating FastMap and its hybrid extensions, our study offers practical insights into heuristic design for scalable pathfinding. Our results confirm that hybrid heuristics—especially max-fusion with ED pivots—consistently reduce node expansions. While trends align with [10], we observed stronger performance from ED pivots. These findings highlight the complementary strengths of geometric and landmark-based embeddings across diverse grid domains.

1.1 Related Work

Heuristic-guided search methods such as A* [8] have long relied on informative and admissible heuristics to reduce node expansions and improve runtime, particularly in large graphs like grid maps. A key area of development in this context has been the use of abstraction-based and embedding-based heuristics.

Abstraction and refinement techniques have been central to improving the scalability of search. Holte et al. [9] introduced graph-based abstractions that reduce problem size by collapsing similar states, showing that abstract solutions can guide real problem solving efficiently. One prominent hierarchical approach is Hierarchical Path-Finding A* (HPA*) [2], which partitions grid maps into fixed-size blocks and connects them via portals (entrance points). This abstract graph significantly reduces search depth while retaining near-optimality and is widely used in robotics and games. Sturtevant and Buro [13] proposed dynamic refinement of abstract paths at runtime, and later work by Sturtevant [12] developed memory-efficient representations for these abstractions to support large-scale grids.

Although abstraction methods can offer speed and compactness, they often rely on domain-specific assumptions (e.g., portal placement in HPA*) and require complex refinement strategies. In contrast, our focus in this project is on embedding-based heuristics that are lightweight, domain-agnostic, and easy to precompute.

Differential Heuristics (DH) [14, 6] estimate the shortest path by leveraging precomputed distances to a set of landmarks. The heuristic $h(a, b) = \max_p |d(p, a) - d(p, b)|$ is admissible and consistent under the triangle inequality. DHs are especially well-suited for grid environments due to their low memory footprint, linear preprocessing time, and compatibility with uniform-cost assumptions.

FastMap is a more recent technique adapted from data mining [5] for shortest path computations [3]. It embeds the nodes of a graph into a low-dimensional Euclidean space, where ℓ_1 distances between points approximate the original shortest-path distances. This embedding is constructed by iteratively selecting pairs of pivot nodes and updating residual distances. The resulting heuristic is admissible and consistent, making it suitable for A*. Unlike DH, FastMap produces an explicit geometric embedding of all nodes, which can be reused for operations like computing the median or centroid—useful in problems such as multi-agent meeting [1]. FastMap also uses less memory than full pairwise distance storage and avoids dependence on domain-specific layouts.

While promising, FastMap's performance varies depending on domain complexity. In their extensive empirical analysis, Mashayekhi et al. [10] showed that FastMap is often weaker than DH in challenging grid domains. They introduced a hybrid method, FM+DH, which combines FastMap for the initial $k-1$ embedding dimensions and DH for the final one. This hybrid captures more residual costs and leads to stronger heuristics in both single-agent and multi-agent pathfinding. The paper also proposes a new pivot selection strategy based on heuristic error (HE), which avoids selecting similar pivots across multiple FastMap embeddings. This diversification improves heuristic coverage and is particularly helpful when using ensembles of FastMap heuristics. Their results indicate that, while FastMap alone may underperform on harder maps, the FM+DH and FM+HE variants significantly improve performance—often closing the gap with, or surpassing, traditional DHs in terms of node expansions while maintaining low computational overhead.

Compared to abstraction methods like HPA*, FastMap offers a simpler and more flexible preprocessing strategy. It avoids the need to explicitly construct abstract graphs or define block boundaries and portals. Yet, its embeddings implicitly capture global structural information that can support scalable

search. Techniques like sector-based decomposition [13] or hierarchical representations [12] offer precise control in constrained domains, but FastMap's general-purpose formulation makes it suitable for a wide range of applications, including maps with irregular or dynamic layouts.

This project builds directly on the insights from Mashayekhi et al. [10], evaluating FastMap and FM+DH heuristics across a set of grid maps using A* search. We aim to replicate key findings from their work and assess the effectiveness and generalizability of FastMap-based heuristics in domains where both abstraction and embedding approaches have previously been applied.

2 Problem Definition

We consider the standard grid-based pathfinding problem in 2D environments, where the agent must find a cost-optimal path from a start cell to a goal cell while avoiding obstacles. The environment is represented as a uniform-cost grid, with each cell either walkable or blocked. Movements are allowed in 8 directions: four cardinal (N, S, E, W) with cost 1, and four diagonals (NE, NW, SE, SW) with cost $\sqrt{2}$. To enforce realistic motion, diagonal moves are only permitted when both adjacent cardinal neighbors are also walkable (i.e., corner-cutting is disallowed).

Formally, we model the grid as an undirected graph $G = (V, E)$, where V is the set of walkable cells and E contains edges between valid adjacent walkable cells according to these movement constraints. Given a start node $s \in V$ and a goal node $g \in V$, the objective is to find a path $\pi = \{v_0 = s, v_1, \ldots, v_n = g\}$ that minimizes the total cost:

$$\sum_{i=0}^{n-1} c(v_i, v_{i+1}),$$

where $c(v_i, v_{i+1}) \in \{1, \sqrt{2}\}$ depending on movement direction.

The benchmark maps used in this project are taken from the MovingAI Lab [11], a widely adopted testbed in pathfinding research. Maps follow the octile format, where '.' and 'G' represent walkable terrain and '@', 'T', 'O' denote impassable or out-of-bounds cells. Scenario files specify start and goal coordinates and the optimal cost, assuming the aforementioned movement model. A sample 8-connected neighborhood and corner-cutting restriction are shown in Figure 1.

Map Format. Maps follow the MovingAI format:

```
type octile
height y
width x
map
```

The ASCII grid uses characters such as '.' and 'G' to denote passable terrain, and '@', 'T', and 'O' for impassable or out-of-bounds terrain. Scenario files specify queries with start and goal locations and the known optimal path cost.

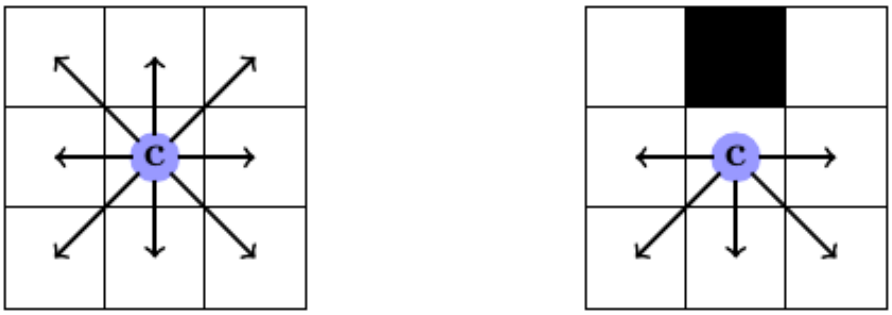
Connectivity and Valid Paths. Each walkable cell may have up to 8 neighbors as shown in Figure 1a. Corner-cutting rules are illustrated in Figure 1b. A path is valid if all segments conform to connectivity rules and do not violate corner-cutting.

Path Validity. For a query (s, g), a valid path must be a sequence of nodes $\pi = \{s, v_1, ..., v_k, g\}$ such that every segment (v_i, v_{i+1}) follows a valid direction and obeys the corner-cutting constraint. For example:

- Moving from a to b in Figure 2 is valid (three NE diagonal steps).
- Moving from a to c is invalid due to a mix of directions.
- Moving from c to d is invalid due to blocked corner cells.

If the start and goal are the same node, the path must be empty, and its cost must be 0.

[1]Adapted from `https://github.com/gppc-dev/startkit-classic/blob/master/Problem_Definition.md` and `https://movingai.com/benchmarks/formats.html`.



(a) 8-connected neighbors (b) Invalid diagonal moves due to corner cutting

Figure 1: Connectivity rules and corner-cutting constraints.[1]

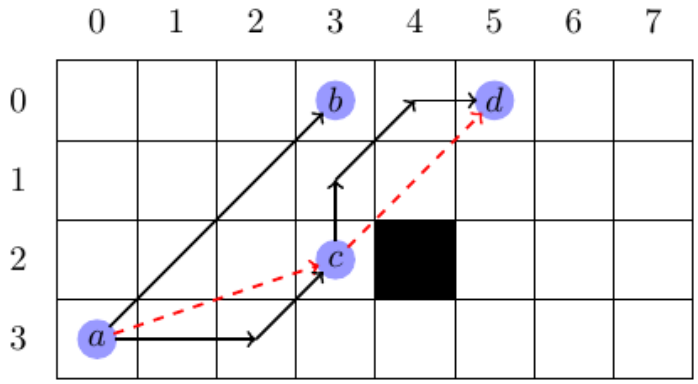


Figure 2: Valid and invalid path segments[1]

2.1 Heuristic Construction via Metric Embeddings

To accelerate A* search in large graphs, heuristic functions are often constructed using precomputed distance information. Given a weighted undirected graph $G = (V, E)$ with cost function $c : E \to \mathbb{R}_{\geq 0}$, a heuristic $h : V \times V \to \mathbb{R}_{\geq 0}$ is admissible if $h(u, v) \leq d(u, v)$ for all $u, v \in V$, where $d(u, v)$ is the cost of the optimal path between u and v under c.

Differential Heuristics (DH) [14] estimate $d(u, v)$ by using a set of landmark nodes P, such that

$$h_{\mathrm{DH}}(u, v) = \max_{p \in P} |d(p, u) - d(p, v)|. \tag{1}$$

DH is simple and effective in grid environments, requiring a small memory footprint and fast evaluation.

FastMap [3] constructs a K-dimensional metric embedding by iteratively selecting pivot pairs and computing coordinate projections for each node based on shortest-path distances. The resulting vector $\mathbf{x}(v) \in \mathbb{R}^K$ defines the heuristic:

$$h_{\mathrm{FM}}(u, v) = \sum_{k=1}^{K} |x_k(u) - x_k(v)|. \tag{2}$$

The pivot pairs can be chosen using either the classic *embedding distance* (ED) strategy or the more informative *heuristic error* (HE) method [10], which selects pivots based on discrepancies between true and estimated costs to improve embedding diversity.

Hybrid heuristics such as FM+DH or $\max\{\text{FM+DH}, \text{DH}\}$ have also been proposed to combine the strengths of both methods. This project evaluates all these strategies by comparing their impact on A* search efficiency over benchmark grid domains.

3 Methods

This section describes the implementation of our heuristic-guided pathfinding framework, which unifies differential heuristics and metric embeddings under a generalized embedding-based paradigm. Inspired by prior work on FastMap and landmark-based heuristics [3, 10, 14], our framework constructs and evaluates a family of admissible heuristics that share a common structure: a pivot selection strategy, a embedding function, and an aggregation operator. At runtime, A* search uses these heuristics to guide node expansions, supporting variants such as additive fusion and max fusion for improved performance across different grid environments.

3.1 Heuristic Construction

We implement and evaluate a set of admissible heuristics for grid-based A* search. These heuristics are constructed using either landmark distances or low-dimensional metric embeddings. At runtime, the selected heuristic function guides the node expansion order during A* search. Our system supports switching between multiple modes including FM, DH, FM+DH, and max-fused hybrids with different pivot selection strategy.

We unify the heuristics under the Generalized Metric Embedding (GME) framework (see Algorithm 1 in [10]), where each heuristic $h(u, v)$ is parameterized by:

- A **pivot selection strategy** (e.g., farthest-pair or heuristic-error),
- A **embedding function** (e.g., projection-based for FastMap, or absolute difference for DH),
- An **aggregation operator** (e.g., max or sum).

This abstraction allows us to describe all heuristics (FM, DH, hybrids) under a unified form while supporting flexible combinations of geometric and landmark-based strategies.

Pipeline Overview. All heuristics are precomputed in the PREPAREFORSEARCH phase (Algorithm 1). This includes generating the FastMap embedding, computing DH vectors, and selecting pivots via either the classical farthest-pair method or heuristic error (HE) strategy. Once constructed, all heuristic information is cached for use by the A* planner.

To increase embedding diversity, we allow construction of multiple FastMap embeddings using independently chosen pivot pairs. This stacking of embeddings enables the heuristic to better approximate distances across varied directions of the state space and reduces systematic underestimations in sparse or obstacle-heavy regions.

Differential Heuristics (DH). Given a set of landmark nodes P, we compute shortest-path distances $d(p, v)$ for all $v \in V$ using Dijkstra's algorithm (Algorithm 5, in the `GETFARTHESTPAIR()` function). The heuristic is then computed at runtime by maximizing the absolute distance difference across all pivots:

$$h_{\mathrm{DH}}(u, v) = \max_{p \in P} |d(p, u) - d(p, v)|.$$

To ensure robustness, we fallback to octile distance when precomputed values are unavailable or inconsistent.

Algorithm 1 PrepareForSearch: Unified Heuristic Embedding (FM + DH Variants)

function PREPAREFORSEARCH($bits$, $width$, $height$, $filename$)
▷ Step 1: Extract walkable grid and build graph
    Extract walkable cells → `cellToIndex`, `indexToCell`
    Build 8-connected graph → `edges`, $n \leftarrow$ number of walkable cells
▷ Step 2: Choose FM variant (Single or Hybrid)
    $K \leftarrow$ number of dimensions (e.g., 5), $\epsilon \leftarrow 10^{-2}$
    if mode == Hybrid **then**
        `embedding` $\leftarrow$ FASTMAP_HYBRID($edges$, n, K, ϵ)
    else
        `embedding` $\leftarrow$ FASTMAP($edges$, n, K, ϵ)
    end if
▷ Step 3: Construct FMHD embedding with HE pivots
    $K_{HE} \leftarrow 5$, $\epsilon_{HE} \leftarrow 10^{-2}$
    BUILDFMEMBEDDING_HE($width$, $height$, K_{HE}, ϵ_{HE})
▷ Step 4: Compute DH vectors
    COMPUTEDHVECTORS($k = 5$) ▷ Standard ED landmarks
▷ Step 5: Record heuristic computation time
    return heuristic context ctx with precomputed data
end function

Algorithm 2 FastMap

function FASTMAP($edges$, n, $Kmax$, ϵ)
    Initialize residual weights $w_0 \leftarrow edges$
    Initialize empty embedding for n nodes
    while $Kmax > 0$ **do**
        $G_0 \leftarrow$ BUILDADJLIST(w_0)
        $(a, b) \leftarrow$ GETFARTHESTPAIR(G_0)
        $d_a \leftarrow$ DIJKSTRA(G_0, a), $d_b \leftarrow$ DIJKSTRA(G_0, b)
        if $d_a[b] < \epsilon$ **then break**
        end if
        for all v **do**
            $x_k(v) \leftarrow \frac{d_a[v]+d_a[b]-d_b[v]}{2}$
            Append $x_k(v)$ to embedding
        end for
        for all edges (u, v) **do**
            $w(u, v) \leftarrow w(u, v) - |x_k(u) - x_k(v)|$
        end for
        $Kmax \leftarrow Kmax - 1$
    end while
    return embedding
end function
function GETFARTHESTPAIR(G)
    Choose random $s \in V$
    $d \leftarrow$ DIJKSTRA(G, s)
    $a \leftarrow \arg\max_v d[v]$
    $d' \leftarrow$ DIJKSTRA(G, a)
    $b \leftarrow \arg\max_v d'[v]$
    return (a, b)
end function

FastMap Embedding (FM). FastMap constructs a K-dimensional embedding $\mathbf{x}(v) \in \mathbb{R}^K$ for each node v based on shortest-path distances between selected pivot pairs (a_k, b_k) (Algorithm 5, lines 1–20). Each coordinate is computed as:

$$x_k(v) = \frac{d(a_k, v) + d(a_k, b_k) - d(b_k, v)}{2}.$$

In our FastMap implementation, we iteratively construct low-dimensional embeddings by projecting node positions using selected pivot pairs. After computing each embedding dimension, we update the graph's edge weights to reflect the residual distance not yet captured by the current embedding. Specifically, for each edge (u, v), the cost is reduced by the amount explained by the new dimension:

$$w_{k+1}(u, v) = \max(0,\ w_k(u, v) - |x_k(u) - x_k(v)|).$$

This ensures admissibility by maintaining non-negative weights, and mirrors the residual cost update procedure described in Algorithm 1 of [10], where each dimension incrementally reduces the unexplained portion of the distance metric.

While our generalized embedding framework (Section 3.1) abstracts heuristics in terms of pivot selection, embedding function, and aggregation, it does not explicitly model cost updates as a component of the embedding interface. However, in practice, this residual update step is implemented internally within our FM algorithms to maintain theoretical guarantees and match the behavior described in prior work [10]. The final FM heuristic is the ℓ_1 distance in embedding space:

$$h_{\mathrm{FM}}(u, v) = \sum_{k=1}^{K} |x_k(u) - x_k(v)|.$$

Algorithm 3 FastMap with ED Pivots + DH Final Dimension

function FASTMAP_DHFINAL($edges$, n, $Kmax$, ϵ)
  Initialize residual weights $w_0 \leftarrow edges$ and empty embedding for n nodes
  ▷ Step 1: Standard FastMap using ED pivots
  for $k = 1$ to $Kmax - 1$ **do**
    $G_0 \leftarrow$ BUILDADJLIST(w_0)
    $(a_k, b_k) \leftarrow$ GETFARTHESTPAIR(G_0, n)
    $d_a \leftarrow$ DIJKSTRA(G_0, a_k), $d_b \leftarrow$ DIJKSTRA(G_0, b_k)
    if $d_a[b_k] < \epsilon$ **then break**
    end if
    for all $v \in V$ **do**
      if $d_a[v] = \infty$ or $d_b[v] = \infty$ **then**
        $x_k(v) \leftarrow 0$
      else
        $x_k(v) \leftarrow \frac{d_a[v] + d_a[b_k] - d_b[v]}{2}$
      end if
      Append $x_k(v)$ to `embedding[`v`]`
    end for
    for all edges (u, v) **do**
      $w(u, v) \leftarrow w(u, v) - |x_k(u) - x_k(v)|$
    end for
  end for
  ▷ Step 2: Append DH[0] from residual graph as final coordinate
  $p \leftarrow$ random node from V
  $d_p \leftarrow$ DIJKSTRA(G_0, p)
  for all $v \in V$ **do**
    Append $d_p[v]$ as last coordinate to `embedding[`v`]`
  end for
  return `embedding`
end function
function GETFARTHESTPAIR(G, n)
  $a \leftarrow$ random node from V
  for $t = 1$ to τ **do**
    $d_a \leftarrow$ DIJKSTRA(G, a)
    $b \leftarrow \arg\max_v d_a[v]$
    $d_b \leftarrow$ DIJKSTRA(G, b)
    $a \leftarrow \arg\max_v d_b[v]$
  end for
  return (a, b)
end function

Algorithm 4 FastMap with HE Pivots + DH Final Dimension

function FASTMAP_HE_HYBRID($edges$, n, $Kmax$, ϵ)
  Initialize residual edge weights $w_0 \leftarrow edges$
  Initialize empty embedding for n nodes
  Define base heuristic $h(\cdot, \cdot) \leftarrow$ OctileDistance
  ▷ Step 1: FastMap embedding using HE pivots
  for $k = 1$ to $Kmax - 1$ **do**
    $G_0 \leftarrow$ BUILDADJLIST(w_0)
    $(a_k, b_k) \leftarrow$ GETFARTHESTPAIR_HE(G_0, n, h)
    $d_a \leftarrow$ DIJKSTRA(G_0, a_k), $d_b \leftarrow$ DIJKSTRA(G_0, b_k)
    if $d_a[b_k] < \epsilon$ **then break**
    end if
    for all $v \in V$ **do**
      if $d_a[v] = \infty$ or $d_b[v] = \infty$ **then**
        $x_k(v) \leftarrow 0$
      else
        $x_k(v) \leftarrow \frac{d_a[v] + d_a[b_k] - d_b[v]}{2}$
      end if
      Append $x_k(v)$ to `embedding[`v`]`
    end for
    for all edges (u, v) **do**
      $w(u, v) \leftarrow w(u, v) - |x_k(u) - x_k(v)|$
    end for
  end for
  ▷ Step 2: Append DH[0] as final dimension
  $p \leftarrow$ random pivot from V
  $d_p \leftarrow$ DIJKSTRA(G_0, p)
  for all $v \in V$ **do**
    Append $d_p[v]$ as final coordinate to `embedding[`v`]`
  end for
  return `embedding`
end function
function GETFARTHESTPAIR_HE(G, n, h)
  $t \leftarrow$ random node from V
  $d_t \leftarrow$ DIJKSTRA(G, t)
  $p_1 \leftarrow \arg\max_v (3 \cdot d_t[v] - 2 \cdot h(t, v))$
  $d_{p_1} \leftarrow$ DIJKSTRA(G, p_1)
  $p_2 \leftarrow \arg\max_v (3 \cdot d_{p_1}[v] - 2 \cdot h(p_1, v))$
  return (p_1, p_2)
end function

Algorithm 5 Differential Heuristic

function COMPUTEDHVECTORS(k)
  Select k farthest landmarks using octile distance
  for all landmarks p **do**
    $dh[p] \leftarrow$ DIJKSTRAFROM(p)
  end for
end function

Pivot Selection Strategies. Our framework supports two pivot selection strategies for constructing FastMap embeddings and selecting DH landmarks:

- **Embedding Distance (ED):** Selects the farthest pair of nodes using Dijkstra distances from a random source. This classical strategy captures global geometry and is implemented in Algorithm 5 (`GetFarthestPair`).

- **Heuristic Error (HE):** Iteratively selects pivot pairs by maximizing the expression $3d - 2h$, where d is the true distance from a reference node and h is an existing heuristic (e.g., octile distance). This approach encourages embeddings that expose residual underestimations and guide more informative coordinate directions. The full routine is shown in Algorithm **??** (`GetHEPair`).

While ED captures long-range structure in a static manner, HE adapts to the current residual graph and the geometry of the prior embedding, yielding more complementary dimensions. Both strategies can be used independently or in combination, depending on the desired diversity and tightness of the heuristic.

3.2 Hybrid Heuristics.

In addition to standalone heuristics, we implement composite methods that combine FM and DH (see Algorithm 7):

- **FM+DH (Additive):** Adds FM and DH values from the same pivot configuration.
- **FM+DH (HE):** Uses HE-based FM and DH embeddings for improved diversity.
- **Max-Fusion:** Returns $\max\{h_{\mathrm{FM+DH}}, h_{\mathrm{DH}}\}$, selecting the tighter lower bound at each query.

This design is motivated by the complementary strengths of the two embedding types. FastMap constructs low-dimensional global embeddings that approximate shortest-path distances through geometric projection, while Differential Heuristics capture local structure by encoding landmark-to-node distances. Their strengths are often complementary—FM provides compact, general guidance, whereas DH offers tight, localized estimates—making their combination particularly effective.

Hybrid heuristics such as FM+DH and max fusion align naturally with the Generalized Metric Embedding (GME) framework: we use a shared or independently selected pivot set and aggregate their contributions using either summation or maximization. For example, FM+DH combines the ℓ_1 distance in FM space with the maximum difference from DH pivots. Both components are admissible and often encode different aspects of the underlying distance space. The max fusion variant selects the tighter lower bound between FM+DH and an additional DH embedding, which can improve performance in cluttered maps or narrow corridors.

Algorithm 6 A* Search with FM, DH, and Hybrid Heuristics

function ASTAR($start$, $goal$, ctx, $selected\_mode$)
  Initialize open list with $start$
  $g[start] \leftarrow 0$, $f[start] \leftarrow h(start, goal)$
  $get\_h \leftarrow$ SELECTHEURISTIC($selected\_mode$)
  while open list is not empty **do**
    $current \leftarrow$ node with lowest f in open list
    if $current = goal$ **then return** reconstructed path
    end if
    for all neighbors n of $current$ **do**
      $tentative\_g \leftarrow g[current] + cost(current, n)$
      if n not visited or $tentative\_g < g[n]$ **then**
        $g[n] \leftarrow tentative\_g$
        $f[n] \leftarrow g[n] + get\_h(n, goal)$
        Add n to open list
      end if
    end for
  end while
  return failure
end function

We implement both ED-based and HE-based hybrid embeddings to investigate how pivot diversity affects performance.

All combinations are implemented using runtime function binding in the A* planner (Algorithm 6), via the `SelectHeuristic` function (Algorithm 7). This modular design enables clean switching between heuristic modes with minimal overhead.

A* with Heuristics. During online planning, A* expands nodes in increasing order of $f(v) = g(v) + h(v)$ (Algorithm 6). The heuristic function h is selected using a mode parameter (see Algorithm 7) and applied at each neighbor expansion. This modular setup enables controlled comparisons of heuristic effectiveness across identical maps and scenarios.

Algorithm 7 SelectHeuristic: FM, DH, and Hybrid Heuristics

function SELECTHEURISTIC($mode$)
  if $mode =$ `FM` **then**
    return ℓ_1 distance between `fmEmbedding[a]` and `fmEmbedding[b]` (fallback: Octile)
  else if $mode =$ `DH` **then**
    return $\max_p |d(p,a) - d(p,b)|$ from `dhVectors` (fallback: Octile)
  else if $mode =$ `FM+DH (ED, FinalDim)` **then**
    $h \leftarrow \sum_{i=1}^{k} |x_i(a) - x_i(b)| + |x_k(a) - x_k(b)|$ using `fmEmbedding`
    return h (fallback: Octile)
  else if $mode =$ `FM+DH (HE, FinalDim)` **then**
    $h \leftarrow \sum_{i=1}^{k} |x_i(a) - x_i(b)| + |x_k(a) - x_k(b)|$ using `fmEmbedding_HE`
    return h (fallback: Octile)
  else if $mode =$ `MAX[FM+DH (ED), DH]` **then**
    $h_{\text{FM+DH}} \leftarrow \sum_{i=1}^{k} |x_i(a) - x_i(b)| + |x_k(a) - x_k(b)|$ from `fmEmbedding`
    $h_{\text{DH5}} \leftarrow \max_p |d(p,a) - d(p,b)|$ from `dhVectors`
    return $\max(h_{\text{FM+DH}}, h_{\text{DH5}})$
  else if $mode =$ `MAX[FM+DH (HE), DH]` **then**
    $h_{\text{FM+DH}} \leftarrow \sum_{i=1}^{k} |x_i(a) - x_i(b)| + |x_k(a) - x_k(b)|$ from `fmEmbedding_HE`
    $h_{\text{DH5}} \leftarrow \max_p |d(p,a) - d(p,b)|$ from `dhVectors`
    return $\max(h_{\text{FM+DH}}, h_{\text{DH5}})$
  else if $mode =$ `MAX[DH5, FM5]` **then**
    $h_{\text{FM}} \leftarrow \ell_1$ from `fmEmbedding`
    $h_{\text{DH}} \leftarrow \max_p |d(p,a) - d(p,b)|$ from `dhVectors`
    return $\max(h_{\text{FM}}, h_{\text{DH}})$
  else
    return Octile distance
  end if
end function

Embedding Effectiveness. To better understand the practical impact of different FastMap configurations, we visualize the number of node expansions for a representative map under each heuristic variant in Figure 3 (similar to Figure 4 in [10]).

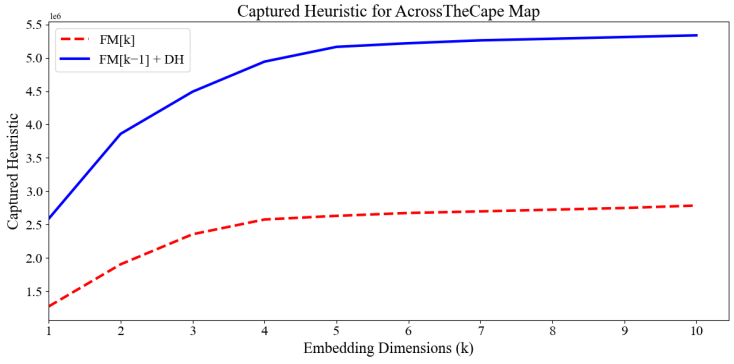


Figure 3: Node expansions under different FastMap and hybrid heuristic configurations on a sample map. Lower is better.

The figure shows the total edge costs captured in the complete embedding. The more edge cost is captured, the smaller the residual becomes, leading to larger heuristic estimates. The FM+DH approach captures distance information more effectively than FastMap alone, especially when DH is added in the final dimension. As shown, FastMap embeddings combined with DH consistently result in higher captured heuristic values, making them stronger and more informative heuristics.

4 Experiments

This section outlines the experimental setup, implementation details, and evaluation methodology for our heuristic-guided A* pathfinding framework. Our primary goal is to evaluate the impact of different heuristic strategies on search efficiency, measured by node expansions and planning time, across a diverse set of grid environments. Inspired by prior work on FastMap-based heuristics [10], we compare classical differential heuristics, metric embeddings, and their hybrid variants.

4.1 Implementation Environment

Our implementation builds upon the official GPPC Starter Kit for the Classic Track [7], which provides a standardized C++ interface for evaluating grid-based pathfinding algorithms. The system is written in C++17 and compiled using GCC (tested with versions 11.4 and 14.2.0) on Ubuntu 22.04 (kernel 5.x). We extended the kit by modifying `main.cpp` to log detailed results for each scenario and export them to `.csv` files.

The core logic is implemented in `Entry.cpp`, in accordance with the GPPC infrastructure. Specifically, we defined:

- `PrepareForSearch`: Performs preprocessing to construct heuristics including FastMap embeddings and DH vectors.
- `GetPath`: Executes A* search using the selected heuristic (FM, DH, or hybrid).

For local testing and benchmarking, we used the following GPPC commands:

- `./run -check <map> <scen>`: Validates query results against the ground truth.
- `./run -run <map> <scen>`: Executes queries and records results in `result.csv`.

All search logic and heuristic functions are implemented in `Entry.cpp`, while `main.cpp` was only modified to support result logging. All runtime I/O (e.g., `run.stdout`, `run.stderr`, `run.info`) remains compatible with the GPPC leaderboard evaluation pipeline.

The full implementation is available at `https://github.com/Masoudjafaripour/SAS_A3_Project`. To ensure reproducibility, we have also attached the modified source files—`main.cpp`, `Entry.cpp`, and `Entry.h`—along with this report.

4.1.1 Hardware Specifications

Experiments were conducted on two machines:

- **Workstation:** Intel Core i9-13980HX (32 cores @ 2.2 GHz), 32 GB RAM, 1TB SSD.
- **Server:** AMD EPYC 7452 (32 cores), 503 GB RAM, running Ubuntu 22.04.

All computations were performed on CPU with no GPU acceleration.

4.2 Benchmark and Scenarios

We use a subset of maps from the MovingAI benchmark suite [11], including game-based maps such as Dragon Age: Origins (DAO) and Starcraft (SC1), as well as artificial maps including random, room, and maze environments. Each map is preprocessed to extract walkable terrain and construct an 8-connected grid graph, where diagonal movements incur a cost of $\sqrt{2}$ and corner-cutting is disallowed.

Each benchmark scenario includes multiple start-goal queries provided in the associated `.scen` file. We evaluate all heuristics using the same set of queries to ensure consistency and comparability across runs. Heuristic values are computed once during the `PrepareForSearch` phase.

4.3 Heuristic Variants Compared

We compare the following heuristic strategies:

- **FM (ED):** FastMap with farthest-pair pivot selection.
- **FM (HE):** FastMap with heuristic-error-based pivot selection.
- **DH:** Differential Heuristic with 1–10 landmarks selected using octile distance.
- **FM+DH (Additive):** Combines FM and DH from the same pivot strategy.
- **FM+DH (HE):** Combines FM and DH embeddings selected using HE pivots.
- **Max-Fusion:** $\max\{h_{\mathrm{FM+DH}}, h_{\mathrm{DH}}\}$ to maximize admissible lower bounds.

All variants are implemented within a unified interface (Algorithm 6) using runtime function binding via `SelectHeuristic` (Algorithm 7).

4.4 Results and Discussion

The results of our experiments are presented in two subsections, each corresponding to one of the evaluation metrics.

We evaluate heuristic performance using the following metrics:

- **Node Expansions:** The total number of nodes expanded by A* (with and without tie-breaking strategy) across a fixed set of queries. For each heuristic, we report the mean, median, and 95% confidence interval across all solved problems.
- **Heuristic Time:** The wall-clock time (in milliseconds) required to compute the heuristic during the `PrepareForSearch` phase, before any node expansions. We report the average per-query heuristic computation time for each domain.

4.4.1 Node Expansion

Due to time constraints in conducting experiments for this project and the varying number of problems across domains, we evaluated the first 20,000 problems from each domain. We report the mean, median, and 95% confidence interval of mean of node expansions in Table 1 for the DAO, SC1, and Random domains, and in Table 2 for the Rooms and Mazes domains. In contrast, the reference paper [10] solved a different number of problems per domain—generally fewer than the total available, but still exceeding 20,000 in each case. We believe this difference does not significantly affect the results. Our choice of evaluating 20,000 problems is further justified in Appendix 6.1.

Table 1: Node expansions by A* for DAO, SC1, and Random maps across different heuristics. Each group reports the median (Md), mean (Mean), and 95% confidence interval (C). HE = Heuristic Error pivoting, ED = Embedding Distance, FAR = Farthest landmark

Heuristics	**Pivot**	**DAO**			**SC1**			**Random**		
		Md	Mean	C	Md	Mean	C	Md	Mean	C
DH10	FAR	2955	4227	55	12528	18431	234	5602	**8390**	120
FM10	ED	3059	5600	91	13734	32840	504	10129	32067	633
FM9+DH	ED	2689	3667	53	9834	22407	410	6923	30921	648
FM9+DH	HE	3464	5429	79	14267	31151	501	9515	34641	667
max[DH5,FM5]	ED	2842	3696	46	11957	20329	310	5670	10693	188
max[FM4+DH,DH5]	ED	2782	**3637**	46	10197	**16296**	247	5369	10681	198
max[FM4+DH,DH5]	HE	3091	4303	61	12658	23152	375	6029	11688	210

Table 2: Node expansions by A* for Rooms and Mazes benchmarks using different heuristic combinations. Each group reports the median (Md), mean (Mean), and 95% confidence interval (C). HE = Heuristic Error pivoting, ED = Embedding Distance, FAR = Farthest landmark.[2]

Heuristics	**Pivot**	**Rooms**			**Mazes**		
		Md	Mean	C	Md	Mean	C
DH10	FAR	10044	**15597**	214	17085	25299	294
FM10	ED	21914	43642	661	16613	27512	382
FM9+DH	ED	12993	38367	673	10312	**15401**	182
FM9+DH	HE	22108	47047	730	10890	15930	185
max[DH5,FM5]	ED	9953	18193	285	13092	19085	225
max[FM4+DH,DH5]	ED	13369	30685	533	13197	19124	221
max[FM4+DH,DH5]	HE	11176	19758	343	13702	19536	223

In addition, only 10-dimensional embeddings (including DH10) were evaluated in our experiments. Due to time constraints, 24-dimensional embeddings were not explored.

Discrepancy in Node Expansions. Before diving into a detailed comparison and analysis of trends across heuristics and domains, we begin with a rough comparison between our results in Table 1 and Table 2, and Table 1 and Table 2 in the original paper [10]. While the overall patterns are similar, our node expansion counts are consistently about 2 to 2.5 times higher, even for the baseline DH10 and FM10 approaches, suggesting that the discrepancy is likely unrelated to the specific implementation of different heuristic methods. This discrepancy is unlikely to be caused by the reduced number of problems we solved as well. As shown in Appendix 6.1, solving all problems in the DAO domain yields results that are very close to the 20K-problem subset.

Given the similarity in heuristic trends and domain-level patterns between our results and those in [10], we believe the differences in *absolute expansion counts* stem from implementation-level factors in our A* algorithm. In particular, we identified that our original implementation lacked an effective *tie-breaking strategy* for nodes with equal $f = g + h$ cost. Without such a bias, A* tends to expand nodes in a broader, often circular frontier—especially in grid domains with uniform cost and weak heuristics—leading to significantly more expansions. This behavior has been well documented in the literature on grid-based A* search [4].

Table 3: Node expansions by **A* with tie-breaking** for DAO, SC1, and Random maps across different heuristics. Each group reports the median (Md), mean (Mean), and 95% confidence interval (C). HE = Heuristic Error pivoting, ED = Embedding Distance, FAR = Farthest landmark

Heuristics	**Pivot**	**DAO**			**SC1**			**Random**		
		Md	Mean	C	Md	Mean	C	Md	Mean	C
DH10	FAR	1171	2253	37	5619	**13300**	223	4033	**7036**	110
FM10	ED	1555	5072	99	9826	32819	549	10135	32751	649
FM9+DH	ED	1010	2195	50	3460	18973	441	5871	30672	657
FM9+DH	HE	1535	3548	79	9122	26564	491	9652	35991	682
max[DH5,FM5]	ED	1166	**2107**	37	6309	14339	241	4561	9943	187
max[FM4+DH,DH5]	ED	1233	2274	39	6386	15396	287	4526	9910	191
max[FM4+DH,DH5]	HE	1370	2380	40	8640	23753	461	5143	11184	204

After adding a standard heuristic-based tie-breaking term to the A* algorithm (i.e., $f = g + h + \varepsilon \cdot h$ with $\varepsilon = 10^{-5}$), the number of node expansions dropped substantially overall. The updated results

[2]It should be noted that the cutoff for the number of problems solved was based on complete scenario files—we did not stop partway through any map. As a result, the number of problems solved is not exactly 20,000 for each domain. Specifically, we evaluated 22,400 problems for DAO, 24,280 for SC1, 20,470 for Random, 21,570 for Rooms, and 24,540 for Mazes.

Table 4: Node expansions by **A* with tie-breaking** for Rooms and Mazes benchmarks using different heuristic combinations. Each group reports the median (Md), mean (Mean), and 95% confidence interval (C). HE = Heuristic Error pivoting, ED = Embedding Distance, FAR = Farthest landmark.[3]

Heuristics	**Pivot**	**Rooms**			**Mazes**		
		Md	Mean	C	Md	Mean	C
DH10	FAR	6886	**12192**	187	13546	20684	255
FM10	ED	22005	43187	663	16625	27469	393
FM9+DH	ED	10253	38149	699	7459	**11744**	153
FM9+DH	HE	19273	49245	816	8013	12337	156
max[DH5,FM5]	ED	7227	15117	260	10420	15839	200
max[FM4+DH,DH5]	ED	7250	18384	380	12089	17385	210
max[FM4+DH,DH5]	HE	9438	21706	429	12222	17432	210

for node expansions across different heuristics and domains are provided in Table 3 and Table 4. The most affected domain is DAO, showing a 40–50% reduction in node expansions. As the domain becomes more complex, this effect diminishes due to increased variability in heuristic effectiveness and path ambiguity.

Among the heuristic methods, `FM10` shows the least reduction in expansions. This is because `FM10`, with ten continuous-valued embedding dimensions, already provides fine-grained guidance, naturally breaking most f-value ties without extra help. In contrast, `DH10`, although similarly strong, uses a max operator over a set of discrete landmark distances, often producing tied f-values in open or symmetric regions. These flat plateaus are where tie-breaking becomes most helpful, and thus `DH10` benefits more from the additional $\varepsilon \cdot h$ term. The same holds for the two max-fusion variants based on ED, which combine coarser heuristic components.

Adding tie-breaking (e.g., $f = g + h + \varepsilon \cdot h$) reduces the *mean* number of node expansions because it resolves f-value ties in favor of nodes closer to the goal, avoiding unnecessary exploration. However, it does not always reduce the 95% confidence interval on the mean because the variance in node expansions across different problems may remain high. This is due to outlier maps where even tie-breaking cannot overcome poor heuristic guidance or complex geometry. In some cases, the added numerical noise may slightly increase the variance, even if the average improves.

These observations confirm that the lack of tie-breaking was one of the primary causes of our higher expansion counts compared to [10]. Although less likely, factors such as data structure choices or priority queue behavior may also have contributed. Therefore, we attribute the remaining discrepancy to implementation variance and minor optimizations that were not detailed in the original paper. Overall, the relative ranking and performance trends of heuristics remain consistent with [10], supporting the correctness of our implementation pipeline.

Baselines: DH and FM The first two rows in Table 1 and Table 2 (and similarly Table 3 and Table 4) present the results for DH10 and FM10, which serve as the baseline heuristics in this project. Consistent with the findings in the reference paper [10], DH generally outperforms FM by yielding fewer node expansions across all statistical metrics. However, in the original paper, FM showed competitive median performance to DH in the DAO domain. In contrast, our results show a more pronounced performance gap: DH performs more than twice as well as FM in most domains, and in the Random domain, it achieves nearly four times fewer node expansions. This discrepancy aligns with the paper's observation that FM struggles with a tail of hard problems and performs better in low-dimensional or linear environments such as DAO. In more nonlinear domains like SC1 or Random, DH heuristics more effectively handle difficult scenarios, which explains the sharper performance gap observed in our experiments.

A Single 10-Dimension Embedding Next, we compare the overall performance of single 10-dimensional heuristic embeddings using both ED and HE pivot selection strategies. Similar to the findings in [10], `FM9+DH` with ED pivots consistently outperforms `FM10` across all domains. The most significant improvement appears in the Maze domain, highlighting the benefit of replacing the final FM embedding dimension with a DH component. Once again, aligning with the results reported

in [10], FM9+DH with HE pivots did not perform as well as its ED counterpart, and with the exception of the Maze domain, it did not show a significant improvement over FM10.

Two 5-Dimension Embeddings Finally, we compare the results of using two 5-dimensional embeddings and taking their maximum. Following [10], the baseline in this setting is the maximum of a single 5-dimensional FastMap embedding and a 5-dimensional DH embedding. The FastMap component can also be replaced with a hybrid FM4+DH embedding, constructed using either ED or HE pivot selection.

Consistent with the original paper, all three variants—max[FM5, DH5], max[FM4+DH, DH5] (ED), and max[FM4+DH, DH5] (HE)—achieve comparable performance across domains. However, in contrast to the paper's findings where HE pivoting gave the best results in DAO and Rooms domains, our experiments show that max[FM4+DH, DH5] with ED pivots consistently outperformed the others in DAO and SC1. This suggests that in our implementation, HE-based pivot selection did not yield as strong of an improvement as reported in [10].

4.4.2 Heuristic Lookup Speed

We measure heuristic computation time within the PrepareForSearch function, before any node expansion or actual search is performed. Specifically, we time the total duration required to construct the heuristic for all 20k problems solved, and report the average computation time per problem. This isolates the cost of heuristic construction from the rest of the search process. The results are shown in Table 5 for each heuristic methods.

Table 5: Mean heuristic computation time (in milliseconds) across benchmark domains.[4]

Algorithm	DAO	SC1	Random	Rooms	Mazes
DH10	3.85	39.16	10.20	11.54	2.63
FM10	1.23	7.23	6.30	9.28	1.90
FM9+DH (ED)	6.49	34.19	17.09	18.30	3.82
FM9+DH (HE)	5.99	31.00	13.93	14.11	3.19
max[DH5,FM5]	4.32	22.75	11.24	12.86	2.85
max[FM4+DH,DH5](ED)	6.09	34.09	13.74	14.54	3.03
max[FM4+DH,DH5](HE)	6.08	32.83	14.53	13.81	3.02

In contrast to the reference paper [10], we recorded heuristic computation times across all benchmark domains—not just SC1. The most notable discrepancy appears in the SC1 domain, where our recorded times are over 100× larger than those reported in the original paper (see their Table 3, reproduced in Figure 7 in the Appendix). This difference may stem from inefficiencies in our implementation and data structures. Additionally, our results show that the computation time for FastMap is approximately 4× lower than for other heuristics, which is particularly interesting because this contrast is minimal in their reported results across different heuristic.

An additional observation from Table 5 is that FM10 has the lowest heuristic computation time, while FM9+DH with ED pivot selection has the highest. The result that adding just a single DH dimension to FM10 leads to significantly longer computation—exceeding even HE-based methods that require iterative maximization—is somewhat counterintuitive and thought-provoking.

Moreover, the variation in heuristic computation time across domains is also notable: SC1 has the highest average time, while Maze has the lowest. This discrepancy is primarily influenced by the size and complexity of the maps in each domain, which directly affects the performance of Dijkstra's algorithm used during preprocessing.

[4]Only the heuristic computation times for A* without tie-breaking are reported in Table 5, as we observed no theoretical or empirical differences in computation time due to tie-breaking.

5 Conclusion

This project implemented and evaluated a suite of heuristic strategies for efficient A* search on large grid maps, with a focus on FastMap and its hybrid variants. FastMap embeds the graph into a compact Euclidean space, enabling fast and admissible ℓ_1 heuristics. However, when used in isolation, FastMap often underperforms in complex environments. Motivated by the findings in [10], we investigated hybrid strategies such as additive FM+DH and max-fusion combinations, alongside advanced pivot selection methods based on heuristic error (HE). These strategies aim to combine the global structure captured by FastMap with the tight local bounds provided by Differential Heuristics.

Our experiments across five domains from the MovingAI benchmark reveal that the relative ranking of heuristic strategies closely mirrors those in [10], despite differences in absolute node expansion counts caused by implementation factors. These discrepancies were partially addressed through the introduction of a tie-breaking strategy. Nevertheless, the overall patterns and trends across domains and heuristics remain consistent and similar to the original paper. We find that hybrid heuristics generally outperform standalone FastMap and DH methods in terms of node expansions. Notably, `FM9+DH` with ED pivot selection consistently outperforms `FM10`, and the `max[FM4+DH, DH5]` heuristic with ED pivots achieves the best overall performance in the DAO and SC1 domains. This slightly contrasts with the reference paper, where HE pivoting yielded stronger results in DAO and Rooms. A key difference in our findings is that HE-based pivot selection was not as effective as reported in [10].

In terms of heuristic computation time, our results differ significantly from [10], particularly in the SC1 domain, where our recorded times are over 100× higher. While `FM10` is consistently the fastest to compute, hybrid methods like `FM9+DH (ED)` require substantially more time. Interestingly, adding just one DH dimension to `FM10` results in longer computation times than HE-based methods that involve iterative pivot selection—an unexpected outcome.

Overall, aside from minor differences, our findings align with those reported in [10]. The results reinforce the effectiveness of combining FastMap embeddings with DH vectors—particularly through max-fusion strategies—and highlight the importance of pivot selection and implementation details in shaping both performance and efficiency. These observations confirm that our project successfully replicates and validates the core contributions of the original paper [10]. As future work, we aim to explore alternative techniques, more efficient data structures, and optimized programming practices to reduce the absolute number of node expansions—bringing them closer to the results in [10]—as well as to improve heuristic computation times.

References

[1] Dor Atzmon, Ariel Felner, Roni Stern, and Nathan R. Sturtevant. Multi-agent meeting algorithms: Topics and new directions. *AI Communications*, 33(3):389–407, 2020.

[2] Adi Botea, Martin Muller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 2004.

[3] Arik Cohen, Iddo Rabinovich, and Ofer Strichman. The fastmap algorithm for shortest path computations. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1385–1391. IJCAI, 2018.

[4] Augusto B Corrêa, André Grahl Pereira, and Marcus Ritt. Analyzing tie-breaking strategies for the a* algorithm. In *IJCAI*, pages 4715–4721, 2018.

[5] Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995.

[6] Meir Goldenberg, Ariel Felner, Roni Stern, Jonathan Schaeffer, and Robert C. Holte. Improved heuristics for optimal path-finding on game maps. In *Proceedings of the 7th International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 9–14, 2007.

[7] Grid-Based Path Planning Competition. Grid-Based Path Planning Competition (GPPC). `https://gppc.search-conference.org/`, 2025. Accessed: 2025-04-15.

[8] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[9] Robert C Holte, Taieb Mkadmi, Robert M Zimmer, and Alan J MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence*, 85(1-2):321–361, 1996.

[10] Saeed Mashayekhi, Nathan R. Sturtevant, and Adi Botea. Analyzing and improving the use of the fastmap embedding in pathfinding tasks. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023.

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

[12] Nathan Sturtevant. Memory-efficient abstractions for pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2007.

[13] Nathan Sturtevant and Michael Buro. Partial pathfinding using map abstraction and refinement. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2005.

[14] Nathan R. Sturtevant, Ariel Felner, Mordechai Barer, Jonathan Schaeffer, and Neil Burch. Differential heuristics. *Artificial Intelligence*, 174(12-13):848–861, 2010.

6 Appendix

6.1 Node Expansion on all problem sets

To investigate the effect of evaluating all problems versus a subset of 20,000 problems on node expansions, we solved all queries in each benchmark domain using the DH10 heuristic. The results are reported in Table 6, with the total number of problems in each domain indicated in parentheses. Compared to the first row of Table 1 and Table 2, which show node expansions for DH10 over the 20k problems, the differences are not significant. This observation supports the validity of our 20k selection, at least within the scope of this project.

Table 6: Node expansions for the DH10 heuristic across all problems in each benchmark domain. Each group reports the median (Md), mean (Mean), and 95% confidence interval (C). The total number of problems per domain is indicated in parentheses.

Heuristics	**Pivot**	**DAO** (311,240)			**SC1** (251,140)			**Random** (311,500)		
		Md	Mean	C	Md	Mean	C	Md	Mean	C
DH10	FAR	2216	4029	19	11352	19660	95	5708	8674	31
		Rooms (168,700)			**Mazes** (506,820)					
		Md	Mean	C	Md	Mean	C			
DH10	FAR	9271	14909	76	14682	26388	85			

In comparison to the results of DH10 reported in Table 6, and those from the original reference paper [10] (specifically, the first rows of their Tables 1 and 2, see Figures 5 and 6 in Appendix), we observe that solving all problems in each domain—despite the fact that [10] did not solve all problems either—did not significantly improve alignment between our results and theirs. Nevertheless, the relative ratio of node expansions across domains in our results remains almost consistent with the trends observed in [10].

6.2 Tables and Figures of Referenced Paper

In this section, we include screenshots of selected tables and figures from our reference paper [10] to facilitate direct comparison with our results. For further details, please refer to the original paper [10].

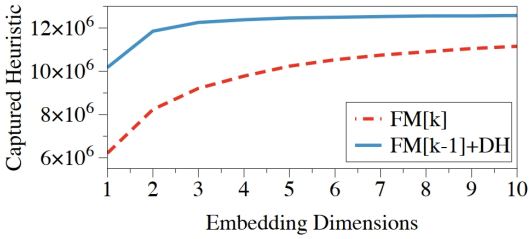


Figure 4: Captured edge costs in embeddings. FM[k] refers to an embedding with k dimensions.

Figure 4: Figure 4 of the paper [10] showing captured cost in embedding

Heuristics	Pivot	DAO (157,905)			SC1 (198,230)			Random (146,220)			Rooms (78,840)		
		Md	Mean	C	Md	Mean	C	Md	Mean	C	Md	Mean	C
DH10	FAR	561	1,581	13	2,219	7,453	58	2,984	**5,345**	32	4,098	**7,700**	66
FM10	ED	516	2,172	21	4,199	18,548	144	5,917	12,455	84	9,471	19,271	165
FM9+DH	ED	381	1,034	11	1,576	15,036	136	4,053	11,069	83	5,783	16,778	163
FM9+DH	HE	370	1,038	12	1,460	13,961	129	4,049	11,089	83	6,219	16,960	161
max[DH5,FM5]	ED	467	1,290	11	2,277	9,029	74	3,452	6,118	36	4,467	8,809	75
max[DH5,FM4+DH]	ED	406	1,036	9	1,511	7,834	70	3,139	5,773	35	3,632	8,030	73
max[FM4+DH,DH5]	HE	399	**1,002**	8	1,441	**7,227**	64	3,079	5,717	34	3,466	**7,687**	70
max[DH5,FM4+DH]	HE	400	**995**	8	1,446	7,399	66	3,025	5,628	34	3,355	**7,598**	70
DH24	FAR	518	1,483	13	1,520	5,678	46	2,387	4,469	28	3,077	5,882	53
FM24	ED	401	1,471	16	1,983	16,120	138	5,257	11,923	83	8,059	18,130	163
FM23+DH	ED	349	887	10	1,198	14,783	135	3,562	10,838	83	5,437	16,471	163
max[DH12+FM12]	ED	355	772	6	916	4,758	46	2,191	4,461	29	2,658	5,930	57
max[DH12,FM11+DH]	ED	327	**646**	5	784	4,314	45	2,011	**4,256**	29	2,152	5,513	57
max[DH12,FM11+DH]	HE	326	**641**	5	773	**4,240**	44	1,995	**4,229**	29	2,127	5,222	53
max[8xFM2+DH]	ED	641	2,036	19	4,924	18,415	145	5,376	11,962	83	7,654	18,200	166
max[8xFM2+DH]	HE	473	1,518	16	1,694	11,337	117	2,638	5,533	37	1,900	**5,139**	56

Table 1: Results reporting expansions by A* including the median node, mean, and the 95% confidence interval on the mean.

Figure 5: Table 1 of the paper [10] showing node expansion

Heuristics	Pivot	Mazes (586,370)		
		Md	Mean	C
DH10	FAR	6,554	10,090	27
FM10	ED	8,898	15,311	44
FM9+DH	ED	4,668	**7,067**	19
FM9+DH	HE	4,679	**7,084**	19
max[DH5,FM5]	ED	6,630	10,036	26
max[DH5,FM4+DH]	ED	6,270	9,477	24
max[FM4+DH,DH5]	HE	6,252	9,449	24
max[DH5,FM4+DH]	HE	6,231	9,422	24
DH24	FAR	4,806	7,493	21
FM24	ED	5,557	10,512	32
FM23+DH	ED	3,348	**4,873**	13
max[DH12+FM12]	ED	4,377	6,622	17
max[DH12,FM11+DH]	ED	4,449	5,409	23
max[DH12,FM11+DH]	HE	4,164	6,179	16
max[8xFM2+DH]	ED	10,122	15,047	39
max[8xFM2+DH]	HE	7,313	11,712	32

Table 2: Results on mazes extending Table 1.

Figure 6: Table 2 of the paper [10] showing node expansion

Algorithm	Median	Mean	95%
FM	271	242	40
FMDH	263	231	37
DH	260	233	37

Table 3: Average total time in microseconds to compute heuristics on 75 SC1 maps.

Figure 7: Table 3 of the paper [10] showing computation time of heuristic