

Hierarchical Path-Finding for Navigation Meshes (*HNA**)

Nuria Pelechano, Carlos Fuentes

Universitat Politècnica de Catalunya

Abstract

Path-finding can become an important bottleneck as both the size of the virtual environments and the number of agents navigating them increase. It is important to develop techniques that can be efficiently applied to any environment independently of its abstract representation. In this paper we present a hierarchical NavMesh representation to speed up path-finding. Hierarchical path-finding (*HPA**) has been successfully applied to regular grids, but there is a need to extend the benefits of this method to polygonal navigation meshes. As opposed to regular grids, navigation meshes offer representations with higher accuracy regarding the underlying geometry, while containing a smaller number of cells. Therefore, we present a bottom-up method to create a hierarchical representation based on a multilevel k-way partitioning algorithm (*MLkP*), annotated with sub-paths that can be accessed online by our Hierarchical NavMesh Path-finding algorithm (*HNA**). The algorithm benefits from searching in graphs with a much smaller number of cells, thus performing up to 7.7 times faster than traditional A* over the initial NavMesh. We present results of *HNA** over a variety of scenarios and discuss the benefits of the algorithm together with areas for improvement.

Keywords: path-finding, hierarchical representations, navigation meshes

1. Introduction

Most video games are required to simulate thousands or millions of agents who interact and navigate in a 3D world and show capabilities such as chasing, seeking or intercepting other agents. Path-finding provides characters with the ability to navigate autonomously in a virtual environment. The most well known path-finding algorithm is A*, which explores the nodes of a graph while balancing the accumulated cost with a heuristic to find an optimal path quickly. Throughout the years many algorithms have been proposed to further speed up the basic A* algorithm, but the cost of these algorithms is still strongly dependent on the size of the graph. Hierarchical path-finding aims to reduce the number of nodes that need to be explored when computing paths in large terrains. The reduction in the number of nodes for higher levels of the hierarchy significantly decreases the execution time and memory footprint when calculating paths.

Current hierarchical techniques may result in unbalanced abstractions. For example, top-down hierarchies are created by splitting the environment into large square clusters, where all the clusters contain the exact same number of lower level grid cells. The main disadvantages of such constructions are that the resulting higher level of the hierarchy may have an uneven number of edges between nodes and also an uneven number of walkable cells (since there may be some clusters with a large percentage of the grid cells being occupied by obstacles).

Navigation meshes represented by polygons provide closer representation of the geometry with a lower number of cells than regular grids. Since having a smaller number of cells can greatly accelerate path-finding, it is therefore necessary to extend the concept of hierarchical path-finding to a more general

representation of navigation meshes with polygon based cells. Moreover it would also be beneficial to have a hierarchical representation with a balanced number of polygons per node and portals between nodes.

In this paper we present a new hierarchical path-finding solution for large 3D environments represented with polygonal navigation meshes. The presented solution works with navigation meshes where cells are convex polygons, and thus it also includes triangular representations. Our hierarchical graph representation is based on a multilevel k-way partitioning algorithm annotated with sub-path information. Our method presents a flexible approach in terms of both the number of levels used in the hierarchy and the number of polygons to merge between levels of the hierarchy. We evaluate the gains in performance when using our hierarchical path-finding, and discuss the trade-offs between the number of merged polygons and the number of levels employed for the search. We present a number of benchmarks that can help during the parameter fitting process to achieve the best speedups, as well as a quantitative analysis of the bounds on sub-optimality of the paths found with *HNA**. We also present an evaluation of the bottleneck that appears for certain configurations when inserting the start and goal positions in the hierarchical representation.

2. Related Work

A large amount of work to speed up path-finding focuses on enhancing the A* algorithm to reduce the computational time needed to calculate a path. This comes at the cost of finding sub-optimal paths or allowing a certain degree of error when searching for the optimal path and then allows the algorithm to

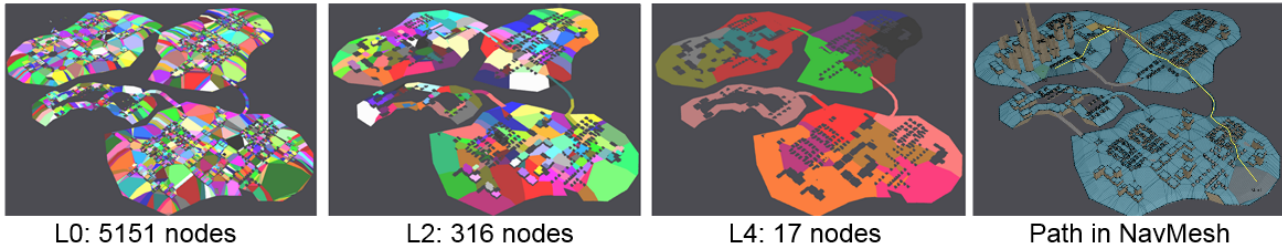


Figure 1: Hierarchical partition of a polygonal navigation mesh of over 5000 nodes at level 0 (each color identifies a node in the graph), 316 at level 2 and 17 at level 4, and the final path calculated with *HNA**.

61 repair those errors in future searches that are interleaved with
62 the execution.

63 The well known *A** algorithm [1] is a robust and simple
64 to implement method with strict guarantees on optimality and
65 completeness of solution. The *A** algorithm uses a heuristic
66 to restrict the number of states that must be evaluated before
67 finding the true optimal path and it guarantees to expand an
68 equal number or fewer states than any other algorithm using
69 the same heuristic. However *A** can be very time consuming
70 for large scenarios. Anytime Planning algorithms find the best
71 suboptimal plan and iteratively improve this plan while reusing
72 previous plan efforts. One of the most popular *A** is called
73 Anytime Repairing *A** (*ARA**) [2]. It performs a series of
74 repeated weighted *A** searches while iteratively decreasing a
75 loose bound (ϵ). It iteratively improves the solution by reduc-
76 ing ϵ and reusing previous plan efforts to accelerate subsequent
77 searches. However *ARA** solutions are no longer guaranteed
78 to be optimal.

79 *D** Lite [3] performs *A** to generate an initial solution and
80 repairs its previous solution to accommodate world changes by
81 reusing as much of its previous search efforts as possible. *D**
82 can correct "mistakes" without re-planning from scratch, but
83 requires more memory. Anytime Dynamic *A** (*AD**) [4] com-
84 bines the properties of *D** and *ARA** to provide a planning
85 solution that meets strict time constraints. It efficiently updates
86 its solutions to accommodate dynamic changes in the environ-
87 ment.

88 *DBA** algorithm [5] combines the memory-efficient sector
89 abstraction developed for [6] and the path database used by [7]
90 in order to improve space complexity and optimality. Huang [8]
91 presented a path planning method for coherent and persistent
92 groups in arbitrarily complex navigation mesh environments.
93 The group is modeled as a deformable and splittable area pre-
94 serving shape. The efficiency of the group search is determined
95 by three factors: path length, deformation minimization, and
96 spitting minimization.

97 Hierarchical graph representations have also been used for
98 visualization purposes of large data sets [9] [10]. The goal in
99 these applications is to offer an overview first, and then be able
100 to zoom and filter to offer details on demand.

101 Planning via hierarchical representation has been used to
102 improve performance in problem solving for a long time [11].
103 Holte et. al. [12] introduced hierarchical *A** to search in an
104 abstract space and use the solution to guide search in the origi-

105 nal space. There has also been work on abstraction based on
106 bottom-up approaches for general graphs [13][14] but without
107 considering balancing the number of nodes or minimizing the
108 edge-cut. Sturtevant and Jansen [15] extended the theoretical
109 work slightly and provided examples of a number of different
110 abstraction types over graphs. In this work graphs are created
111 from 2D grid-like structures by setting a node for each walkable
112 cell. Bulitko et al [16] showed that the quality of paths can de-
113 crease exponentially with each level of abstraction. Sturtevant
114 and Geisberger [17] studied the combination of abstraction and
115 contraction hierarchies to speed up path-finding. Abstraction
116 uses a top-down approach creating a 16x16 overlay across the
117 lower level regular grid. Contraction builds a higher level graph
118 using the concept of importance of nodes, which requires pri-
119 orities for the nodes to be set correctly as they will affect the
120 contraction algorithm.

121 Hierarchical representations have been used over 2D grid
122 representations [18]. In [19] an adaptive subdivision of the en-
123 vironment is proposed with efficient indexing, updating, and
124 neighbor-finding operations on the GPU which reduces the mem-
125 ory requirements. Another similar method based on *HPA**, but
126 taking into account the size of the agents and terrain traver-
127 sal capabilities, is Hierarchical Annotated *A** (*HAA**) [20]. It
128 presents an extension of *HPA** which allow multi-size agents
129 to efficiently plan high quality paths in heterogeneous-terrain
130 environments. Another interesting implementation is *DT-HPA**
131 [21] which uses a decision tree to create a hierarchical subdivi-
132 sion.

133 Jorgensen presented an automatic structuring method based
134 on a hierarchy that separated buildings into floors linked by
135 stairs and represents floors as rooms linked by doorsteps [22].
136 This method has a strict hierarchy and does not scale to large
137 outdoors environments such as the ones often presented in video
138 games. Zlatanova [23] presented a framework of space subdivi-
139 sion exclusively for indoor navigation, by identifying rooms
140 and corridors and including semantical information.

141 There are other approaches that focus on allowing agents
142 to be more environment-aware [24]. In this work planning is
143 based on an Anytime Dynamic *A**, and it is carried out sat-
144 isfying multiple special constraints imposed on the path, such
145 as: stay behind a building, walk along walls or avoid the line
146 of sight of other agents. In [25] a multi-domain anytime dy-
147 namic planning framework is presented which can efficiently
148 work across multiple domains by using plans in one domain to

149 accelerate and focus searches in more complex domains. It ex-
150 plores different domain relationships including the use of way-
151 points and tunnels. The different domains use only two rep-
152 resentations in terms of spacial subdivision, a 2D grid, and a
153 triangular mesh.

154 Hierarchical representations have been used to calculate agents
155 moving between two points at different levels of complexity
156 [26] [27]; from finding a route to animating 3D characters.
157 They have also been used to combine high level path-finding
158 with low level local motion [28]. When using triangular rep-
159 resentations, it is possible to optimize the data structures and
160 built in features such as clearance that can greatly improve per-
161 formance during path-finding [29] [30]. But it is not straight
162 forward to extend this implementation to polygonal meshes (i.e.
163 it would not be enough with a simple triangulation of the poly-
164 gons). There has been a recent technical report extending *HPA**
165 to triangular representations [31].

166 As most of the abstract representations for large 3D com-
167 plex environments employ polygon based representations (e.g:
168 NEOGEN [32], Recast [33], or navmeshes built from the med-
169 ial axis [34]), it is thus necessary to extend the concept of hi-
170 erarchical path-finding for general representations of navigation
171 meshes. Polygonal meshes have certain features and character-
172 istics that must be taken into account when evaluating the most
173 suitable hierarchical abstraction to be used.

174 3. Framework

175 Our framework consists of a pre-processing phase where
176 the hierarchy is created, and an adapted version of the basic A*
177 algorithm to perform searches online in this hierarchical repre-
178 sentation.

179 The pre-process phase starts with a polygonal navigation
180 mesh that represents an abstract partition of the 3D world. This
181 first navigation mesh is considered to be the lowest level in a
182 hierarchical tree. The rest of the levels in the hierarchy are
183 created by recursively partitioning a lower level graph into a
184 specific number of nodes. The partition is performed until the
185 graph of the highest level cannot be further subdivided. Thus, a
186 particular path planning search can be executed in any level of
187 this hierarchical tree. The higher the level of the hierarchy, the
188 fewer the number of nodes to search in. This approach allows
189 faster path-finding calculations than using a common A* with-
190 out any hierarchy. Although we have tested our results using the
191 basic A* algorithm, the method presented is general enough to
192 be used with improved versions of A* such as AD*, DBA*,
193 ARA* or D*.

194 The classic hierarchical path-finding algorithm (*HPA** [18])
195 for 2D grids consists of having the 2D grid as low level, and
196 builds a higher level by dividing the environment into squared
197 clusters connected by entrances, where all clusters have the
198 same number of low level grid cells. Clusters are connected
199 with inter-edges with cost 1.0 and the cost of intra-edges are
200 calculated with A* [1] algorithm searches inside each cluster,
201 for all pairs of abstract nodes that shared the same cluster.

202 Gravot et. at. [35] presented a top-down approach to com-
203 bine a 2D grid partition of large tiles, with a lower level nav-

204 igation mesh per tile. So each tile of 32x32 meters has its
205 own navigation mesh, which forces the number of cells to be
206 larger than when the polygon decomposition is generated di-
207 rectly from the original map. This 2-level representation im-
208 proves performance, but the misalignment between axis aligned
209 tiles and geometry causes inconsistencies in the pre-stored ta-
210 bles that force farther subsplitting of tiles.

211 In this work we propose a bottom-up approach that starts
212 with the initial navigation and it merges cells to obtain a higher
213 level of abstraction respecting the advantages of polygonal nav-
214 igation meshes. Grouping low level cells in a general navigation
215 mesh is not as straight forward as deciding to group squares of
216 $n \times n$ cells. The goal is to have a good graph partition with a bal-
217 anced size of components and a small number of edges running
218 between components, as this will reduce the costs of the hierar-
219 chical path-finding algorithm. We use a polygon mesh provided
220 by Recast [33] as our initial navigation graph and the multilevel
221 k -way partitioning algorithm (*MLkP*) [36] to create our hier-
222 archical representation. *MLkP* reduces the size of graph G_i to
223 create G_{i+1} by collapsing vertices and edges. This algorithm
224 has been proven to be faster than other multilevel recursive bi-
225 section algorithms, and produces high quality graphs.

226 3.1. Hierarchical representation

227 The first step is to build the framework for hierarchical searches
228 that is defined as a tree of graphs. We start to compute the
229 lowest graph of the hierarchy ($G_0 = (V_0, E_0)$) by searching the
230 polygons in the original navigation mesh. Each polygon be-
231 comes a node in the G_0 graph and edges are created between
232 polygons that share a border in the original mesh.

233 We define L_{max} as the maximum number of levels for the
234 hierarchical representation, and η as the number of nodes that
235 will be merged between levels of the hierarchy. Once the low-
236 est level graph G_0 is created, the upper levels of the hierarchy
237 $\{G_1, G_2, \dots, G_m\}$ are recursively built by partitioning each level
238 until it reaches the minimum number of the nodes in a graph or
239 $m = L_{max}$.

240 The *MLkP* algorithm starts with a *coarsening phase*, which
241 consists of creating a series of successively smaller graphs de-
242 rived from the input graph. Each graph is constructed from the
243 previous graph by collapsing together a maximal size set of ad-
244 jacent pairs of nodes. After the coarsening phase, a k -way par-
245 titioning of the smallest graph is computed (*initial partitioning*
246 *phase*). Next the *uncoarsening phase* begins by projecting the
247 partitioning of the smallest graph into the successively larger
248 graphs, refining the partitioning at each intermediate level. The
249 different phases of the multilevel paradigm are illustrated in Fig.
250 2.

251 In order to have a good partition the weight of a new node
252 should be equal to the sum of its previous nodes. In our case we
253 are interested in having a balanced number of polygons, there-
254 fore nodes in G_0 are initialized with weight=1. The new edges
255 created are the union of the edges from the previous nodes to
256 preserve the connectivity information in the coarser graph. The
257 coarsening phase ends either when the coarsest graph has a
258 small number of nodes or when the reduction in the size of suc-

cessively coarser graphs becomes smaller than a given threshold.

The *initial partitioning phase* is performed using a multilevel bisection algorithm [36]. Each partition contains roughly $|V_0|/k$ nodes' weight of the original graph. The division is done by KernighanLin (KL) partitioning algorithm [37] which finds a partition of a node into two disjoint subsets of equal size such that the sum of the weights of the edges between those subsets is minimized.

The uncoarsening phase initially projects the partition by assigning the same partition to the collapsed nodes. After each projection step, the partitioning is refined using various heuristic methods to iteratively move nodes between partitions as long as such movements improve the quality of the partitioning solution. The uncoarsening phase ends when the partitioning solution has been projected all the way to the original graph.

This multilevel partitioning process provides a hierarchy of graphs, where the lowest graph $G_0 = (V_0, E_0)$ corresponds to the original NavMesh of the environment, V_0 is the set $v_0^1, v_0^2, \dots, v_0^n$, where each v_0^j is a node representing a polygon of the NavMesh, and E_0 is the set of edges that correspond to portals between nodes of the original NavMesh. Therefore each graph $G_i = (V_i, E_i)$ consists of a set of nodes V_i where each node v_i^j represents a multinode collapsing several adjacent nodes of the lower graph G_{i-1} , i.e.: $v_i^j = \{v_{i-1}^1, v_{i-1}^2, \dots, v_{i-1}^y\}$.

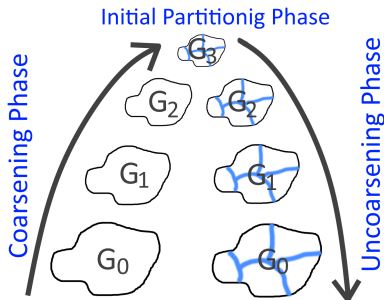


Figure 2: Multilevel k-way partitioning scheme [36].

The procedure allows us to have partitions which ensure high quality edge-cuts, where an edge-cut is defined as the number of edges whose incident nodes belong to different partitions.

The partition is carried out using the *METIS* software package [38], and after the first partition done from G_0 to G_1 all non accessible nodes returned from the NavMesh creation in Recast are eliminated from the hierarchy.

The iteration is done until either it reaches the maximum number of levels in the hierarchy or the graph cannot be further subdivided. The number of merged nodes per level to create a new partition is given by the user defined variable η , where $\eta \approx |V_0|/k$.

Once we have the partitions \mathcal{P} , the new nodes and edges between partitions are created. Edges between partitions are the inter-edges of the graph and contain the edges of the lower graph that join different partitions in the higher graph. Therefore, each partition \mathcal{P}_i has a set of inter-edges E_i which depends on the edges of E_{i-1} that connect nodes of V_{i-1} which fall in dif-

ferent partitions of \mathcal{P}_i . For each pair of inter-edges in a node v_i of the given partition \mathcal{P}_i , A^* is applied between them to calculate the cost of the shortest path and store it as an intra-edge for the given node. For all the graphs of the hierarchy, starting from G_1 and moving up to the highest level (note that G_0 does not contain intra-edges), the Hierarchical NavMesh Graph (*HNG*) is created as indicated in the following algorithm:

Algorithm 1 . Build HNG

```

procedure BUILDGRAPH( $G_i$ )
2:   for  $j \leftarrow 1, |\mathcal{P}_i|$  do
       for  $n \leftarrow v_i^1, |V_i^j|$  do
4:     for  $e \leftarrow 1, numEdges(n)$  do
            $m = neighbour(n, e)$ 
6:       if  $p[n] \neq p[m]$  then
            $G_{i+1}.addInterEdge(V_{i+1}(n), V_{i+1}(m))$ 
8:     for  $k \leftarrow 1, v_{i+1}^j.numEdges()$  do
           for  $l \leftarrow k + 1, v_{i+1}.numEdges()$  do
10:         $cost \leftarrow findPath(k, l)$ 
            $G_i.addIntraEdge(k, l, cost)$ 

```

Partitions will contain the intra-edges for each pair of edges within a node. Figure 3 shows a simple example with the partitions, inter-edges and intra-edges created. Figure 1 represents levels = 0, 2, 4 of the hierarchical partition for a map with 5,515 polygons, with $\mu = 5$ and $L_{max} = 5$.

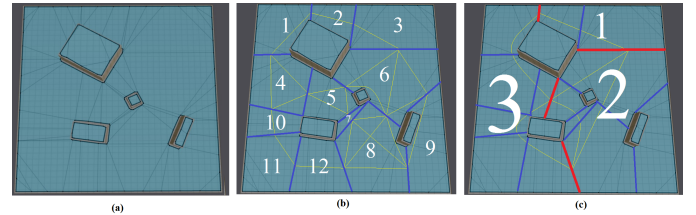


Figure 3: Hierarchical subdivision of a simple map, with $\mu = 5$ and levels = 5. Red lines in (c) represent inter-edges and yellow lines in (b) and (c) represent intra-edges. Partitions are shown with black (a), blue (b) and red (c) separation lines respectively. Level 0=76 nodes (a), Level 1=12 nodes (b), Level 2=3 nodes (c).

3.2. Hierarchical path-finding

Path-finding can be performed at any level of the hierarchy. For given starting and goal positions S and G we need to link this position to the *HNG* and then perform *HNA** in the temporally created graph (note that S and G are linked to the *HNG* and removed once the path is calculated). Note that the algorithm for hierarchical path-finding is conceptually similar to *HPA** [18] but has been adapted to the *HNG* introduced in the previous section. The algorithm proceeds through the following steps:

1. Insert the starting S and goal G positions at the desired level of the hierarchy and connect them to the higher level graph.
2. Search path between S and G at the highest level.
3. Extract intra-edges (optimal sub-paths).

329 4. Delete temporal nodes.

330 Algorithm 2 indicates the details of each step of the *HNA**
 331 algorithm. Note that currently the function *findPath()* simply
 332 calculates A* over the given graph at the level of the hierarchy
 333 indicated by the last parameter and heuristic based on Euclidean
 334 distance.

Algorithm 2 . Online *HNA**

```

procedure FINDPATHHNA*(S, G, L)
  //step 1. Insert and connect nodes S and G at level l
  3:   $n_l^S \leftarrow \text{getNode}(S, l)$ 
      $n_l^G \leftarrow \text{getNode}(G, l)$ 
     if  $l = 0$  then
  6:     $\text{path} \leftarrow \text{findPath}(n_l^S, S, n_l^G, G, 0)$ 
     return path
      $n_{aux}^S \leftarrow \text{linkStartToGraph}(S, n_l^S)$ 
      $n_{aux}^G \leftarrow \text{linkGoalToGraph}(G, n_l^G)$ 
  9:  //step 2. Path-finding between S and G at level l:
      $\text{tempPath} \leftarrow \text{findPath}(n_{aux}^S, S, n_{aux}^G, G, l)$ 
  12: //step 3. Extract sub-paths:
     for  $\text{subpath} \in \text{tempPath}$  do
      $\text{path} \leftarrow \text{getIntraEdges}(\text{subpath}, l - 1)$ 
  15: //step 4. Delete S and G:
      $\text{deleteTempNode}(n_{aux}^S)$ 
      $\text{deleteTempNode}(n_{aux}^G)$ 
  18: return path
  
```

335 **3.2.1. Inserting *S* and *G* and connecting to the graph**

336 The starting *S* and goal *G* positions are inserted in the ge-
 337 ometry at level 0 and then recursively looked up the hierarchy
 338 for the corresponding nodes at the highest level, *L* of the hier-
 339 archy. *S* and *G* are then temporally inserted in the higher level
 340 graph G_L as temporal nodes n_{aux}^S and n_{aux}^G respectively.

341 To connect the temporal node n_{aux}^S with the graph G_L we
 342 need to calculate the path from *S* to each of the inter-edges of
 343 higher level node n_L^S containing *S*. Inter-edges are the union
 344 of those edges from G_0 that connect a node n_0^i with a node n_0^j
 345 where $p_L[n_0^i] \neq p_L[n_0^j]$, (i.e nodes of level 0 that are neighbors
 346 but belong to different partitions of G_L).

347 The paths between *S* and each inter-edge, e_L^j , of n_L^S are cal-
 348 culated to create a temporal intra-edge linking n_{aux}^S to the higher
 349 level graph G_L . Similarly, temporal intra-edges are calculated
 350 linking the goal position *G* to the graph G_L (see figure 4a for an
 351 example of the temporal intra-edges used to connect *S* and *G*
 352 with the graph at the higher level).

353 The performance of this step depends on the computational
 354 cost of calculating each intra-edge for *S* and *G*. In the case of
 355 the starting position, it requires calculating paths between *S* and
 356 each edge e_L^j of the node n_L^S . The same applies to connecting
 357 the goal position *G* within its node.

358 The path-finding algorithm used to connect *S* and *G* is in-
 359 dependent of the algorithm used at the higher level, since the
 360 problem is quite different. In this case we are not finding a path
 361 between two points, but finding all the shortest paths between

362 one point (*S* or *G*) and many (all edges within the node). We
 363 have tested two algorithms, A* and Dijkstra [39].

364 A* is a faster algorithm than Dijkstra since it uses heuris-
 365 tics to expand less nodes. However in this particular scenario
 366 where several A* have to be performed, there will be a number
 367 of nodes explored multiple times for each search. Therefore,
 368 even though Dijkstra is meant to be slower in finding a single
 369 solution, when it comes to finding paths to multiple goals we
 370 may benefit from the fact that we only need to run the search
 371 once and stop as soon as it finds the last edge of the node.

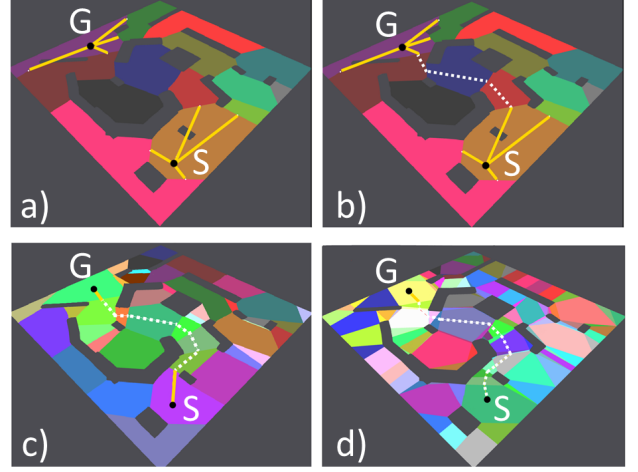


Figure 4: Path-finding computation: *S* and *G* are inserted and linked to their partitions at level 2 by calculating shortest paths to each portal in their respective node(a). Paths are calculated at level 2 (b), and then intra-edges are extracted from lower level 1 (c) and the final path is obtained for level 0 (d).

372 **3.2.2. Search path between *S* and *G* at the highest level**

373 Once the *S* and *G* are temporally connected to the higher
 374 level graph, path-finding is computed with the A* algorithm
 375 in the hierarchical navigation graph (*HNG*) formed by all the
 376 nodes in the higher level of the hierarchy and the connection to
 377 n_{aux}^S and n_{aux}^G . This path-finding at level *i* results in the following
 378 sequence:

$$ie(n_{aux}^S - v_i^1), v_i^1, v_i^2, \dots, v_i^m, ie(v_i^m - n_{aux}^G)$$

380 Note that $ie(n_{aux}^S - v_i^1)$ contains the sequence of nodes at level 0
 381 that belong to one of the temporal intra-edges added during the
 382 connection of *S* with the first high level node of the path v_i^1 , and
 383 similarly $ie(v_i^m - n_{aux}^G)$ contains the sequence of nodes at level
 384 0 between the last high level node of the path v_i^m and the goal
 385 position *G* (see figure 4b where the yellow lines indicate the
 386 temporal intra-edges created for *S* and *G*, and the white dotted
 387 lines the intra-edges to go through the highest level nodes of the
 388 graph).

389 The time execution of this path-finding at level *i* is signif-
 390 icantly faster than finding the path at level 0 due to the large
 391 reduction in the number of nodes.

392 **3.2.3. Extract intra-edges**

393 For the given sequence of high level nodes $\{v_i^1, v_i^2, \dots, v_i^m\}$
 394 belonging to the optimal solution for level *i*, the algorithm re-

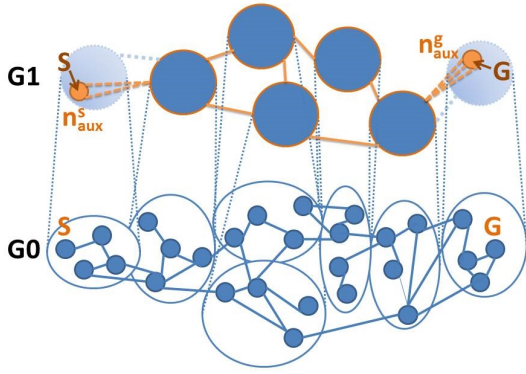


Figure 5: Example of *HNG* with two levels and $\mu = 4$. The orange links and circles represent the edges and nodes that belong to the temporal graph created after linking S and G to the *HNG*. This temporal graph is where the *HNA** is calculated.

395 cursively extracts the intra-edges for each lower node. The final
 396 sequence of intra-edges once level 0 has been reached is the actual
 397 path (sequence of polygons in the NavMesh) that the agents
 398 need to follow to move from S to G .

399 3.2.4. Delete temporal nodes

400 The final and simplest step consists of deleting the temporal
 401 nodes n_{aux}^s and n_{aux}^g from the graph, and all their temporal intra-
 402 edges. After this step, we recover the original *HNG* to perform
 403 future searches.

404 4. Results

405 For the evaluation of our method we have used several mul-
 406 tilayer 3D scenarios as shown in figure 6 with increasing num-
 407 bers of cells in the original NavMesh (see table 1 for details on
 408 the number of nodes in the map).

Table 1: For each map in figure 6, we show the number of triangles in the original geometry, and the number of nodes in the NavMesh depending on whether we use triangles or polygons.

Map Name	Geometry # Triangles	NavMesh # Triangles	NavMesh # Poly
Serpentine City (a)	135.1K	10,152	3,908
City Islands (b)	110.3K	14,551	5,515
Tropical Islands (c)	239.1K	29,499	12,666

409 We have calculated a large number of paths over each of
 410 these scenarios with increasing values of μ on increasing num-
 411 bers of levels in the hierarchy to determine which are the best
 412 configurations for hierarchical path-finding. Results show that
 413 we can achieve significant speedups for certain configurations,
 414 while we may get even worse results than A^* for other con-
 415 figurations. Therefore in this section we evaluate the overall
 416 performance of the algorithm, looking closely at the computa-
 417 tional time taken by each step of the *HNA** algorithm (see alg.
 418 2) to determine areas for improvement.

419 Figure 7 shows the reduction in the number of nodes as we
 420 increase the value of μ and the number of levels in the hierar-
 421 chy. The reduction for the first level is the largest one since
 422 we also remove unconnected polygons during the first step of
 423 the algorithm. From then on the reduction is due to collapsing
 424 nodes based on the value of μ . As we will see when we compute
 425 the overall performance of the algorithm, our experimental re-
 426 sults show that the most suitable configurations tend to happen
 427 when the number of polygons has been reduced around 12% for
 428 level 1 (with $\mu \approx 20$), and the second best configuration tends
 429 to happen when the number of polygons has been reduced to
 430 approximately 2.5% for level 2 (with $\mu \approx 6$).

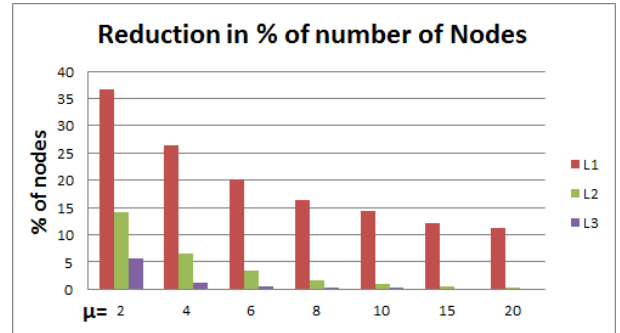


Figure 7: The table shows the percentage of nodes of the original NavMesh for different values of μ and different levels of the hierarchy.

431 To calculate the overall computational time of *HNA** and
 432 compare results, we have computed the average cost of calcu-
 433 lating a large number of paths as shown in figures 8, 9 and 10
 434 with an intel core i7-4770 CPU@3.5Gz, 16GB RAM.

435 For the City island scenario consisting of a NavMesh with
 436 5,515 polygons, we have tested up to 3 levels and increasing
 437 values of $\mu = \{2, 4, 6, 8, 10, 15, 20\}$. As we can see in figure 8a,
 438 the average cost of performing A^* in this scenario is 2.02ms.
 439 Using *HNA** we can improve performance with L1 and all the
 440 values of μ tested ($\mu \in \{2, 20\}$) with the fastest search being
 441 0.51ms for $\mu = 15$. A hierarchy of two levels also improves
 442 the computational times for $\mu \in \{2, 20\}$. However for the case
 443 of having a hierarchy consisting of 3 levels, we only obtain
 444 speedups for $\mu < 7$, since once we collapse 8 or more nodes
 445 between levels the total cost is actually worse than simply com-
 446 puting A^* at $L0$. To better understand why the computational
 447 cost can increase for certain values of μ and levels in the hierar-
 448 chy, we have displayed the cost of *HNA** at L1 and L2 in figure
 449 8 (b) and (c) using different colors for each of the significant
 450 steps of the algorithm.

451 The significant steps of the algorithm are: (1) calculating
 452 A^* at the higher level, (2) extracting intra-edges and (3) con-
 453 necting S and G within the higher node (Note that the other
 454 steps of the algorithm have an insignificant cost below 0.007ms).
 455 As we can see in this figure, the cost of computing A^* at the
 456 highest level decreases since the number of nodes becomes smaller
 457 by increasing levels and μ . However the cost of connecting S
 458 and G can escalate as the higher level nodes increase in size.
 459 This is mainly because as their size gets bigger, the number of
 460 inter-edges also becomes bigger, and thus it requires a higher

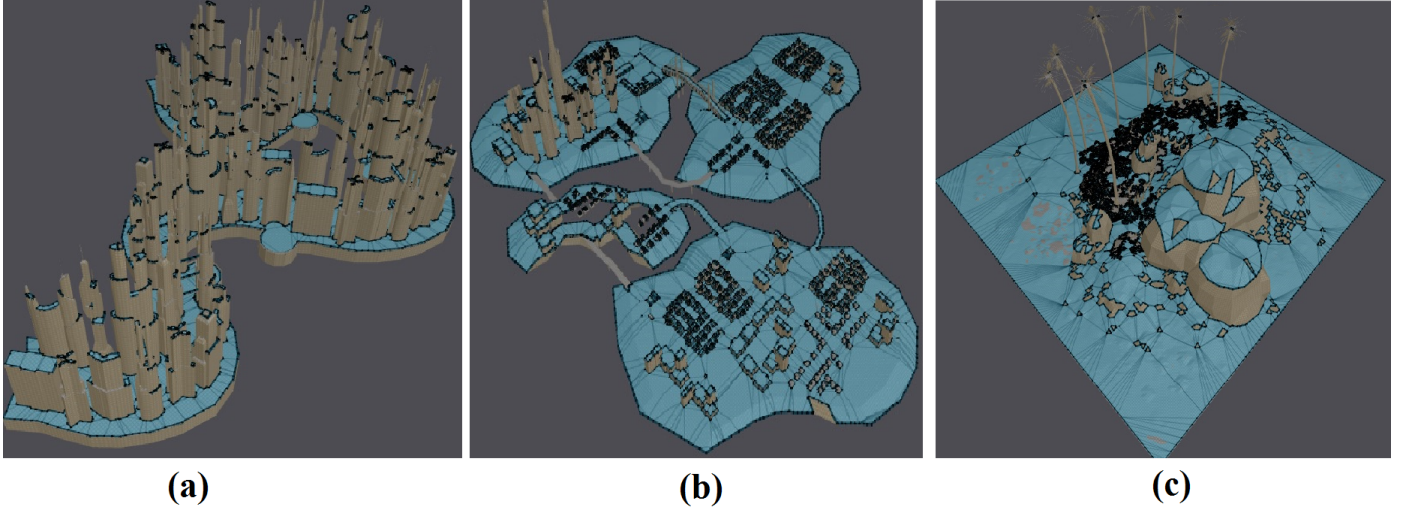


Figure 6: Scenarios used for evaluation (obtained from <http://tf3dm.com/>). (a) Serpentine city, (b) City island, (c) Tropical islands

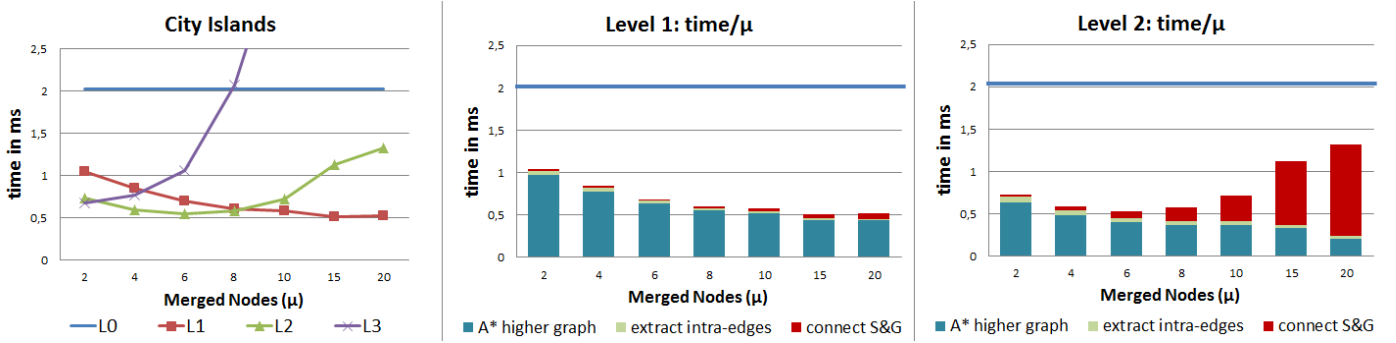


Figure 8: Performance results for the city island scenario (3 levels and $\mu = 2, 4, 6, 8, 10, 15, 20$). (a) show the cost of A^* at $L0$, and HNA^* at $L1, L2$ and $L3$ as μ increases. (b) and (c) show the cost of the different steps of HNA^* for $L1$ and $L2$ respectively.

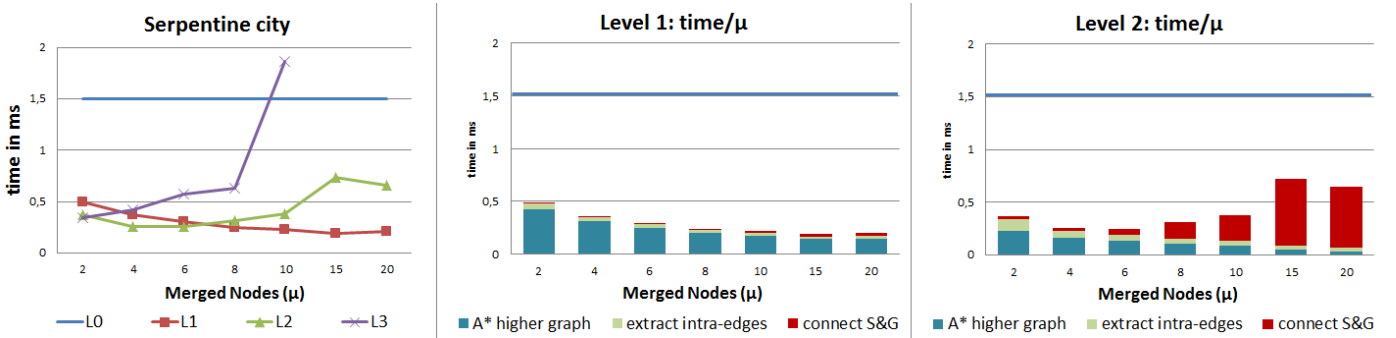


Figure 9: Performance results for the serpentine island scenario (3 levels and $\mu = 2, 4, 6, 8, 10, 15, 20$). (a) show the cost of A^* at $L0$, and HNA^* at $L1, L2$ and $L3$ as μ increases. (b) and (c) show the cost of the different steps of HNA^* for $L1$ and $L2$ respectively.

461 number of A^* searches to compute temporal intra-edges to connect S and G with the HNG .

462
463 From our experimental results, we observed that replacing
464 A^* by Dijkstra to perform the connection step can improve
465 performance. However the difference is only significant for
466 very large nodes with many inter-edges, while it is almost the
467 same for the configurations where HNA^* outperforms A^* at $L0$.
468 Therefore there is still room for improvement in this connection
469 step.

470 In the serpentine city scenario consisting of a NavMesh
471 with 3,908 polygons, we have tested up to 3 levels and $\mu =$
472 $\{2, 4, 6, 8, 10, 15, 20\}$. As we can see in figure 9a, the average
473 cost of performing A^* in this scenario is 1.5ms. By using HNA^*
474 we can improve performance in $L1$ and $L2$ for all the μ values
475 tested, with the fastest search observed for $\mu \in [15, 20]$ and $L1$
476 when it takes 0.19ms on average to compute a path. This repre-
477 sents a 7.7x speedup over basic A^* . As in the previous scenario,
478 for $L3$ we only observe faster searches for small values of μ . In

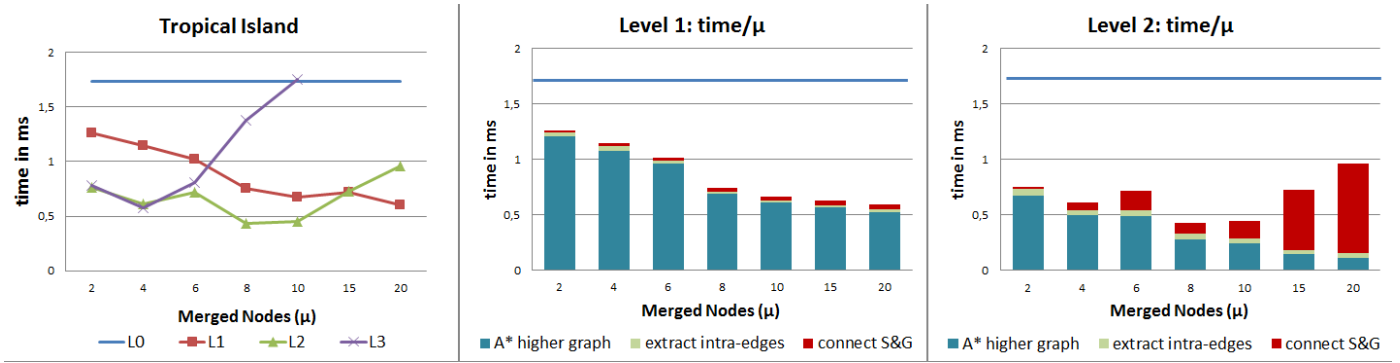


Figure 10: Performance results for the tropical island scenario (3 levels and $\mu = 2, 4, 6, 8, 10, 15, 20$). (a) show the cost of A^* at L_0 , and HNA^* at L_1, L_2 and L_3 as μ increases. (b) and (c) show the cost of the different steps of HNA^* for L_1 and L_2 respectively.

479 figure 9 (b) and (c) we can differentiate the cost for each of the
480 significant steps of the HNA^* algorithm.

481 In the tropical island scenario with an initial NavMesh of
482 12,666 polygons, we have also tested 3 levels of the hierarchy
483 and $\mu = \{2, 4, 6, 8, 10, 15, 20\}$. The time taken by each configura-
484 tion is shown in figure 10. For the combination of levels and
485 values of μ tested in this scenario, the best speedup obtained is
486 4.0x for L_2 and $\mu = 6$.

487 Therefore, the best speedups achieved by HNA^* have been
488 7.7x for the serpentine city, 3.9x for the city island, and 4.0x for
489 the tropical city. At L_1 the cost of the step connecting S and G
490 is almost insignificant compared to the total cost of HNA^* , how-
491 ever from L_2 onwards this step can become an important bottle-
492 neck for larger values of μ . This bottleneck depends largely on
493 the differences in shape and connectivity of the original graph.
494 For example the long structure of the serpentine city makes the
495 edge-cut smaller on average, as merging a larger number of
496 nodes does not increase the number of inter-edges as much as in
497 the city or the tropical island scenarios. Therefore the speedup
498 that can be achieved depends strongly on the configuration of
499 the space and connectivity of the graph G_0 .

500 Figure 11 shows the average number of inter-edges per multin-
501 ode at levels L_2 and L_3 in the hierarchy as the value of μ
502 increases. In general the number of inter-edges (i.e. the edge-cut)
503 increases with the value of μ . However we can observe how for
504 the serpentine scenario the number of inter-edges can actually
505 drop significantly above a certain value of μ , as opposed to the
506 other tested scenarios where it increases with μ .

507 The multilevel k-way partitioning algorithm used to create
508 the HNG attempts to reduce the edge-cut while balancing the
509 number of nodes per partition. Reducing the edge-cut will re-
510 duce the cost of connecting S and G , but in order to improve the
511 results achieved by our algorithm, it would be necessary to find
512 an alternative method for the step connecting S and G . As we
513 can clearly see in the different results (figures 8-10), increasing
514 both μ and levels always reduces the A^* search at the higher
515 level as the search is performed over smaller graphs.

516 Figure 12 illustrates an example of a worst case scenario for
517 HNA^* where the highest level contains excessively large nodes
518 with many inter-edges. This drastically increases the compu-
519 tational time of inserting and connecting S and G . In this ex-

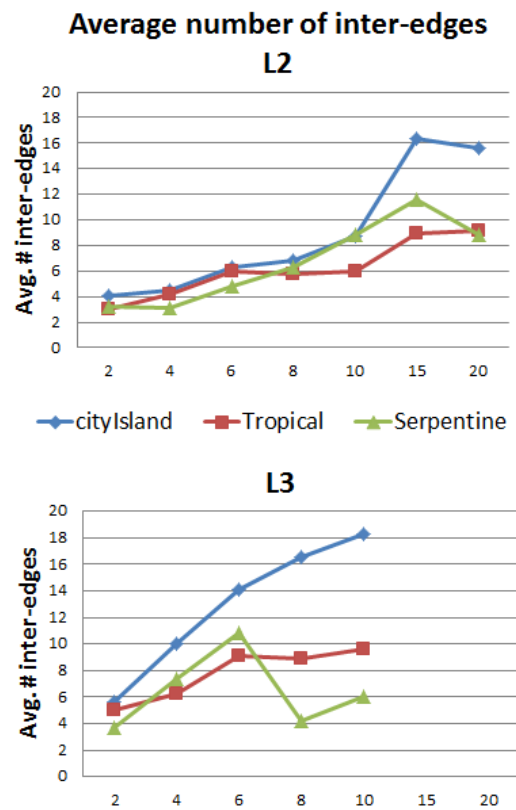


Figure 11: Average number of inter-edges per multinode for levels L_2 and L_3 of the hierarchy as the value of μ increases.

520 ample, the cost of HNA^* would be much higher than simply
521 performing A^* in the original NavMesh, since we are now compu-
522 ting 18 paths to connect S , and 10 paths to connect G . One
523 advantage of having a multilevel hierarchy could be to perform
524 the search dynamically at different levels when S and G belong
525 to neighboring nodes of the highest level.

526 In terms of path quality, there are some differences between
527 the paths found with A^* over the NavMesh, and the ones ob-
528 tained when applying HNA^* . These small deviations are due
529 to the fact that intra-edges compute distances between the cen-
530 ter points of edges, as opposed to A^* that takes into account

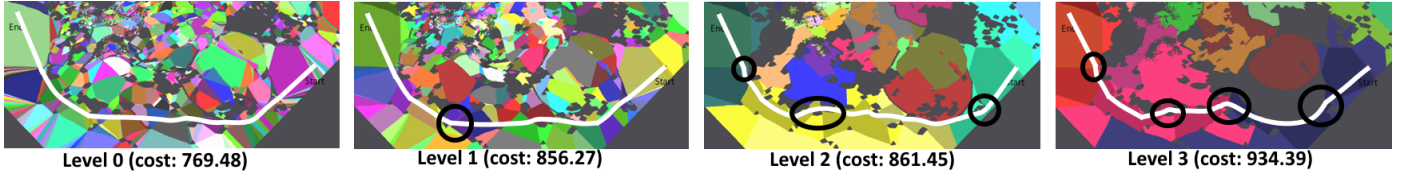


Figure 13: Example of paths calculated at different levels of the hierarchy for the Tropical Island scenario.

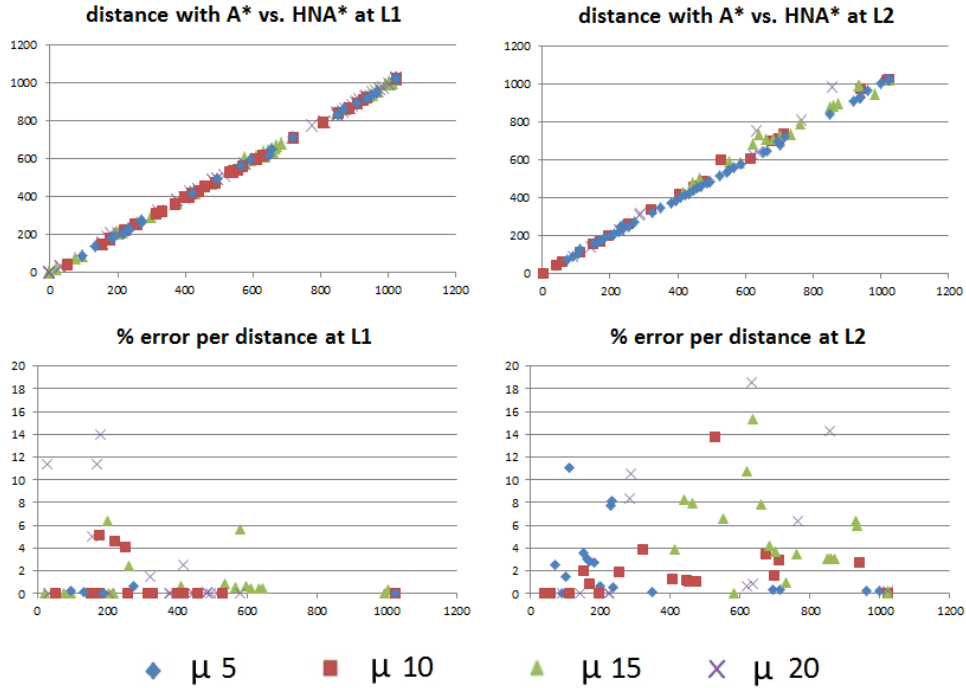


Figure 14: On the left, we show a comparison of path lengths obtained with A^* against HNA^* for different values of μ and level of search in the hierarchy. On the bottom row we show the percentage of error introduced as the length of the path increases.

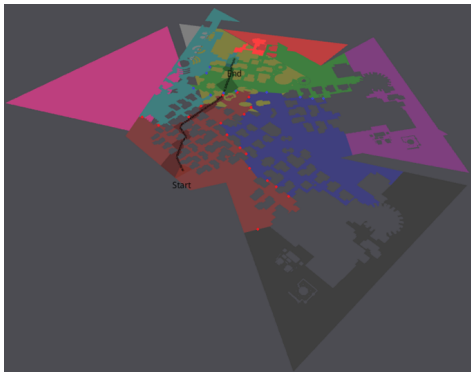


Figure 12: Example of a worst case scenario for connecting S and G . Edges shown as red dots for the cell containing S and blue dots for the cell containing G .

531 crossing portals through the closest point. In any case, since
 532 the paths for intra-edges are always computed using A^* over
 533 the NavMesh, the impact does not propagate up the hierarchy
 534 (i.e. the cost of an intra-edge at level i calculated off-line is not
 535 the sum of the costs of the intra-edges at level $i-1$ but it is com-

536 puted from scratch and stored). Figure 13 shows an example of
 537 path quality and cost in meters of the computed path for differ-
 538 ent levels of the hierarchy. We have chosen an example with
 539 a high error to show how as we increase the number of layers
 540 we can observe more deviation from the optimal route. In this
 541 particular example we observe that for levels 1 and 2 we get a
 542 path with an extra cost of around 10% and for level 3 it can add
 543 an extra cost of 20%. Note that the path differences happens
 544 between nodes of the higher level, or because paths are forced
 545 through the selected higher level node, when the optimal may
 546 be between two high level nodes.

547 Figure 14 shows a quantitative evaluation of the path length
 548 and percentage of error as the length of the path increases. The
 549 four graphs have on the X axis the length of the path between
 550 start and goal as computed by A^* . The top row shows on the Y
 551 axis the length of the path given by the HNA^* for searches per-
 552 formed at level 1 (left) and level 2 (right), with $\mu = \{5, 10, 15, 20\}$.
 553 All points close to the line $x = y$ indicate that both paths have
 554 similar lengths. To highlight the error, we show on the bottom
 555 row the percentage of error (Y axis) for different path lengths.
 556 As we can observe, the results are on average very similar.
 557 The maximum error found for L2 was 18.6% ($\mu = 20$), 15.4%

558 ($\mu = 15$), 13.8% ($\mu = 10$) and 11.0% ($\mu = 5$), and average
 559 errors of 0.39%, 0.17%, 0.11% and 0.05% respectively. The
 560 maximum error for L1 was 14% for $\mu = 20$, and approximately
 561 6% for other values of μ .

562 Navigation meshes can represent a very large number of en-
 563 vironments each with its own unique features that will make
 564 one configuration better than others. Nevertheless, we wanted
 565 to evaluate whether our decision of having a balanced number
 566 of cells and minimum edge-cut was in fact the best option to-
 567 wards achieving better speedups. We ran a comparison study
 568 using *MLkP* but assigning different weights to either the initial
 569 edges or the nodes of G_0 . By doing so we obtained a graph
 570 partition where the number of nodes of G_{i-1} merged in a node
 571 of G_i would be different, and/or the number of inter-edges be-
 572 tween nodes of the partition could vary significantly. Note that
 573 *MLkP* balances the weight given to the nodes of the G_0 graph.
 574 Therefore by randomly assigning different weights, we achieve
 575 an unbalanced partition in terms of the number of nodes per
 576 cluster. Similarly *MLkP* minimizes the weight of the edge-cut,
 577 so if the weights are randomly assigned to the original edges,
 578 then the final number of edges contained in each inter-edge will
 579 not be minimized. Figure 15 shows the results of this evalua-
 580 tion. We noticed that for small scenarios the differences were
 581 not very significant, but as the environment got bigger we could
 582 observe that in fact the balanced partition would provide on av-
 583 erage slightly better results and also worst case scenarios closer
 584 to the average. This can easily be explained by the fact that an
 585 unbalanced number of nodes creates higher level graphs with
 586 more nodes, which increases the path finding at the higher level.
 587 When the smallest edge-cut is not guaranteed, the result may
 588 end up with some nodes having a large number of inter-edges
 589 which drastically increases the step connecting S or G to the
 590 graph. Since this step is the current bottleneck of the algorithm,
 591 we can observe in the figure how worst case scenarios can al-
 592 most triple the total cost of the search.

593 5. Conclusions and Future Work

594 In this paper we have presented a novel algorithm to per-
 595 form hierarchical path-finding over NavMeshes based on a bottom-
 596 up approach. Using a multilevel k-way partitioning algorithm,
 597 we can create a hierarchy of several levels of complexity with
 598 a decreasing number of nodes per level based on a user input
 599 variable μ that determines the approximate number of nodes to
 600 collapse between consecutive levels of the hierarchy. An advan-
 601 tage of our bottom-up approach as opposed to top-down
 602 approaches is that our technique provides a balanced number
 603 of both walkable cells and inter-edges between partitions. We
 604 have shown how our *HNA** algorithm can obtain paths in this
 605 representation faster than when applying the basic path-finding
 606 directly over the navigation mesh.

607 A quantitative comparison between *HNA** and *HPA** would
 608 be interesting. However the main difficulty for such compari-
 609 son is that *HPA** is highly sensitive to the granularity of the grid,
 610 whereas *HNA** does not suffer from this limitation. Therefore it
 611 would be hard to find the right parameters for a fair comparison.
 612 Nevertheless we expect the benefits of *HNA** to become more

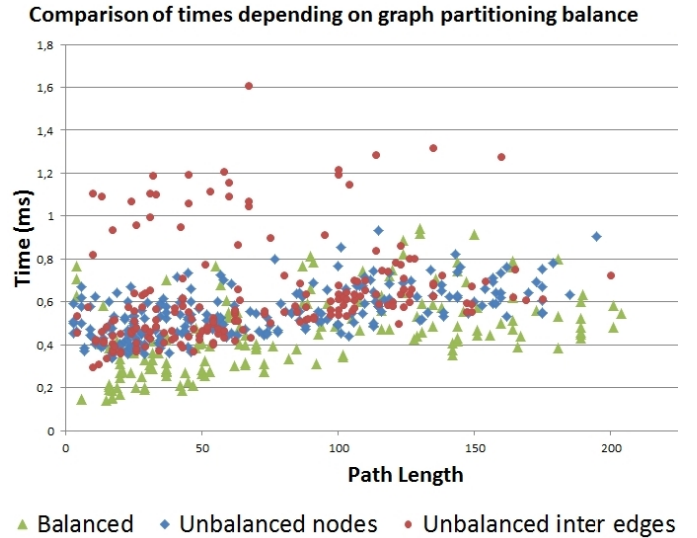


Figure 15: Time taken (in milliseconds) to compute paths with *HNA** as the length of the paths increases. Results from the city island scenario, with two levels of hierarchy and $\mu = 6$

613 noticeable as the environment complexity increases, because
 614 our bottom-up approach using *MLkP* partitioning provides a
 615 good balance of nodes and a minimal edge-cut, whereas this
 616 cannot be achieved with an axis aligned regular grid partition.
 617 Therefore as the environment increases in size and complexity,
 618 we expect *HPA** to start suffering from this lack of balance.

619 We have demonstrated results with the *A** algorithm, but
 620 the architecture presented in this paper could also be used with
 621 other variants of *A**.

622 We have shown improvements over a variety of scenarios
 623 to demonstrate the potential of the method, but have also eval-
 624 uated its limitations. Currently the main limitation of this tech-
 625 nique is the step that connects the starting and goal position
 626 into the hierarchical representation, since its performance drops
 627 as the number of level 0 nodes contained in the higher level
 628 node (multinode) increases. We have tested and compared two
 629 variants for this step, one consisting of calculating *A** from S
 630 and G to each inter-edge in their respective higher level node,
 631 and the second by performing one single Dijkstra search for the
 632 node containing S and the node containing G . Despite Dijkstra
 633 presenting improvements over *A**, it is not fast enough for the
 634 critical cases, therefore future work will focus on testing alter-
 635 natives for this step such as parallel searches, or pre-computing
 636 and storing additional data structures to further improve perfor-
 637 mance. Pre-computing information on a per-cell basis would
 638 be more challenging than when working with regular 2D grids
 639 since there can be a large variation in shape and size of the
 640 initial cells, thus making it difficult to estimate the possible po-
 641 sition for S and G .

642 We have observed that the best speedups can be achieved
 643 by having a one level hierarchy with G_1 containing around 85%
 644 less nodes than G_0 , or when having a two level hierarchy where
 645 G_1 has around 70% less nodes than G_0 , and G_2 has approxi-

mately 95% less nodes than $G0$. Even though it may seem that the fastest and simplest option would be to have a one level hierarchy, it is important to emphasize that the comparisons have been done with average costs for a variety of paths in the graph. Therefore, it would be possible to further extend HNA^* to improve performance based on the location of S and G . For instance, the current algorithm checks whether S and G are in the same multinode, and if so it simply performs A^* (meaning that this case does not benefit from having a hierarchical representation, but it is also not penalized). Moreover we have also shown that when S and G are in neighbouring nodes of the highest level, then the cost can be high since it is necessary to calculate multiple A^* searches to connect S and G , and a negligible cost in finding the high level path. We believe that these two scenarios could benefit from calculating HNA^* in the next level of the multilevel representation. As future work we would also like to consider dynamic updates of the NavMesh and how they could affect the hierarchical representation.

Acknowledgements

This work has been partially by the Spanish Ministry of Science and Innovation and FEDER under grant TIN2014-52211-C2-1-R.

Anonymous.

References

- [1] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems, Science, and Cybernetics* SSC-4 (2) (1968) 100–107.
- [2] M. Likhachev, G. J. Gordon, S. Thrun, Ara* : Anytime a^* with provable bounds on sub-optimality, *Advances in Neural Information Processing Systems* 16 (2004) 767–774.
- [3] S. Koenig, M. Likhachev, D*lite, Eighteenth National Conference on Artificial Intelligence (2002) 476–483.
- [4] M. Likhachev, D. Ferguson, G. Gordon, A. T. Stentz, S. Thrun, Anytime dynamic a^* : An anytime, replanning algorithm, *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (2005) 262–271.
- [5] W. Lee, R. Lawrence, Trading space for time in grid-based path finding., *AAAI*.
- [6] N. R. Sturtevant, Memory-efficient abstractions for pathfinding, *AIIDE* (2007) 31–36.
- [7] R. Lawrence, V. Bulitko, Database-driven real-time heuristic search in video-game pathfinding., *IEEE Trans. Comput. Intellig. and AI in Games* 5 (3) (2013) 227–241.
- [8] T. Huang, M. Kapadia, N. I. Badler, M. Kallmann, Path planning for coherent and persistent groups, *Proceedings of the IEEE International Conference on Robotics and Automation* (2014) 1652–1659.
- [9] J. Abello, Hierarchical graph maps, *Computers & Graphics* 28 (3) (2004) 345 – 359.
- [10] C. Tominski, J. Abello, H. Schumann, Cgv-an interactive graph visualization system, *Computers & Graphics* 33 (6) (2009) 660–678.
- [11] E. D. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artificial Intelligence* 5 (2) (1974) 115 – 135.
- [12] R. Holte, R. C. Holte, M. Perez, M. B. Perez, R. M. Zimmer, R. M. Zimmer, A. MacDonald, A. J. Macdonald, Hierarchical a^* : Searching abstraction hierarchies efficiently, in: *In Proceedings of the National Conference on Artificial Intelligence*, 1996, pp. 530–535.
- [13] R. C. Holte, C. Drummond, M. B. Perez, R. M. Zimmer, A. J. MacDonald, Searching with abstractions: A unifying framework and new high-performance algorithm, in: *Proceedings of the biennial conference-Canadian society for computational studies of intelligence*, 1994, pp. 263–270.
- [14] R. C. Holte, T. Mkdmi, R. M. Zimmer, A. J. MacDonald, Speeding up problem solving by abstraction: A graph oriented approach, *Artificial Intelligence* 85 (1) (1996) 321–361.
- [15] N. Sturtevant, R. Jansen, An analysis of map-based abstraction and refinement, in: I. Miguel, W. Ruml (Eds.), *Abstraction, Reformulation, and Approximation*, Vol. 4612 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2007, pp. 344–358.
- [16] V. Bulitko, Y. Björnsson, R. Lawrence, Case-Based Subgoaling in Real-Time Heuristic Search for Video Game Pathfinding, *Journal of Artificial Intelligence Research (JAIR)* 39 (2010) 269–300.
- [17] N. R. Sturtevant, R. Geisberger, A Comparison of High-Level Approaches for Speeding Up Pathfinding., in: G. M. Youngblood, V. Bulitko (Eds.), *AIIDE*, The AAAI Press, 2010.
- [18] A. Botea, M. Müller, J. Schaeffer, Near optimal hierarchical path-finding, *Journal of Game Development* 1 (2004) 7–28.
- [19] F. Garcia, M. Kapadia, N. I. Badler, Gpu-based dynamic search on adaptive resolution grids, *Proceedings of the IEEE International Conference on Robotics and Automation* (2014) 1631–1638.
- [20] D. Harabor, A. Botea, Hierarchical path planning for multi-size agents in heterogeneous environments., *CIG* (2008) 258–265.
- [21] Y. Li, L.-M. Su, W.-L. Li, Hierarchical path-finding based on decision tree., *RSKT* 7414 (2012) 248–256.
- [22] C.-J. Jorgensen, F. Lamarche, From geometry to spatial reasoning : automatic structuring of 3D virtual environments, *Proceedings of Motion In Games* (2011) 353–364.
- [23] S. Zlatanova, L. Liu, G. Sithole, A conceptual framework of space subdivision for indoor navigation, in: *Proceedings of the Fifth ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness, ISA '13*, ACM, New York, NY, USA, 2013, pp. 37–41. doi:10.1145/2533810.2533819.
- [24] K. Ninomiya, M. Kapadia, A. Shoulson, F. Garcia, N. Badler, Planning approaches to constraint-aware navigation in dynamic environments, *Computer Animation and Virtual Worlds* 26 (2) (2015) 119–139.
- [25] M. Kapadia, F. Garcia, C. D. Boatright, N. I. Badler, Dynamic search on the gpu, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013) 3332–3337.
- [26] W. van Toll, N. Jaklin, R. Geraerts, Towards believable crowds: A generic multi-level framework for agent navigation, in: *ASCI.OPEN*, 2015.
- [27] M. Kallmann, M. Kapadia, *Geometric and Discrete Path Planning for Interactive Virtual Worlds*, Vol. 8, Morgan & Claypool Publishers, 2016.
- [28] N. Pelechano, J. M. Allbeck, N. I. Badler, Controlling individual agents in high-density crowd simulation, in: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '07*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2007, pp. 99–108.
- [29] M. Kallmann, Dynamic and robust local clearance triangulations, *ACM Trans. Graph.* 33 (5) (2014) 161:1–161:17.
- [30] D. J. Demyen, M. Buro, Efficient triangulation-based pathfinding, in: *AAAI*, Vol. 6, 2006, pp. 942–947.
- [31] Z. Bhatena, A. Gheith, D. Fussell, Near optimal hierarchical pathfinding using triangulations (2014).
- [32] R. Oliva, N. Pelechano, Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments, *Computers & Graphics* 37 (5) (2013) 403–412.
- [33] M. Mononen, Navigation-mesh toolset for games, *GitHub Recast and Detour*.
URL <https://github.com/memononen/recastnavigation>
- [34] W. van Toll, A. Cook, R. Geraerts, Navigation meshes for realistic multi-layered environments, in: *Intelligent Robots and Systems (IROS)*, 2011 *IEEE/RSJ International Conference on*, 2011, pp. 3526–3532.
- [35] F. Gravot, T. Yokoyama, Y. Miyake, Precomputed pathfinding for large and detailed worlds on mmo servers. (2014) 269–286.
- [36] G. Karypis, V. Kumar, Multilevel k-way partitioning scheme for irregular graphs, *Journal of Parallel and Distributed Computing* 48 (1998) 96–129.
- [37] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, *Bell system technical journal* 49 (2) (1970) 291–307.
- [38] Karypis, Metis - serial graph partitioning and fill-reducing matrix orderings, *METIS Software Package*.
URL <http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>
- [39] E. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik* 1 (1959) 269–271.