Efficient and Scalable Multi-Geography Route Planning

Vidhya Balasubramanian   Dmitri V. Kalashnikov   Sharad Mehrotra   Nalini Venkatasubramanian

Department of Computer Science
University of California, Irvine
Irvine, CA 92697, USA*

ABSTRACT

This paper considers the problem of Multi-Geography Route Planning (MGRP) where the geographical information may be spread over multiple heterogeneous interconnected maps. We first design a flexible and scalable representation to model individual geographies and their interconnections. Given such a representation, we develop an algorithm that exploits precomputation and caching of geographical data for path planning. A utility-based approach is adopted to decide which paths to precompute and store. To validate the proposed approach we test the algorithm over the workload of a campus level evacuation simulation that plans evacuation routes over multiple geographies: indoor CAD maps, outdoor maps, pedestrian and transportation networks, etc. The empirical results indicate that the MGRP algorithm with the proposed utility based caching strategy significantly outperforms the state of the art solutions when applied to a large university campus data under varying conditions.

1. INTRODUCTION

Many emerging applications such as integrated simulations, gaming, navigation, and intelligent transportation systems require path planning over multiple interconnected geographies. We refer to the problem of path planning over such geographies as multi-geography route planning (MGRP). The goal is to determine the least cost weighted paths from sources to destinations where sources and destinations may reside in different geographies (described in multiple representation paradigms). These geographies may be heterogeneous, may represent space using different models (raster versus vector representations), different coordinate representations, and so on.

Our primary motivation to study the MGRP problem comes from our research in the emergency response domain via the RESCUE[1] and SAFIRE[2] projects. During emergencies first responders have to quickly and safely navigate through unfamiliar spaces to conduct search and rescue operations. Today, agencies are typically hired to conduct offline site surveys of public and critical infrastructure to collect GIS information information such as location of hazardous materials, ventilation structures, entry/exits and to create detailed site maps for planning; this process is expensive, time-consuming and often incomplete. In contrast, a real-time route planning system (enabled by MGRP) will help responders navigate through spaces/structures, to victims and stay in touch with each other.

Consider another example of a meta-simulation platform that models a campus level evacuation triggered by an extreme event and conducts detailed what-if analyses to understands the efficacy of campus response processes. Individuals in campus buildings will exit their respective buildings via stairwells and proceed to preplanned evacuation zones or other destinations through the pedestrian networks. They may proceed to parking lots or collect at different "transit points" to be transported to safe regions using public transport. The building data needed to model this evacuation may be in the form of floor plans (raster or vector data), the outdoor networks may be modeled in a transportation simulator using a graph representation. The building information, in turn may be stored in CAD database which contains information about the floor plans of say 500 buildings. To enable rapid evacuation, we need to identify appropriate paths/exits within buildings and routes on campus - actual shortest paths may require navigation through buildings and across areas on campus that are not actually part of a pedestrian network (e.g. across a field). Likewise, specialized simulators and geography representations may need to be incorporated to model other constraints - e.g. chemical release that occurs as a secondary effect of the primary disaster.

Building the capability of the meta-simulator to run diverse component simulators in consonance in the context of a task raises many challenges. One such challenge is the ability to do path planning over diverse geographies, i.e., the ability to find the best path from say inside a large building to some other location on campus. Such a least cost path may require an agent to exit the building via a specific exit, go through the pedestrian network, and pass through other regions and buildings. MGRP can be incorporated into such a simulation integration platform to model activities in multiple geographies, e.g. evacuation paths from building through

*This research was supported by NSF Awards 0331707 and 0331690, and DHS Award EMW-2007-FP-02535.

[1]`http://www.itr-rescue.org`
[2]`http://www.ics.uci.edu/~cert/safire`

the campus to outdoor transportation corridors and support multiple concurrent processes through geographies, e.g. occupant evacuation and first responder activities.

A straightforward approach is to integrate the multiple geographies into a single homogeneous map and then use traditional path planning solutions, such as Dijkstra's and Bellman Ford algorithms [10, 16, 23] or A* [25]. Depending on the number and size of the geographies, planning across a single homogeneous representation can be computationally expensive and inefficient. In fact, such integration, when feasible, requires significant manual effort (e.g., map conflation) – this is a significant drawback in emergency response context where rapid route planning may be needed over multiple independent maps. To overcome some of the problems of large homogeneous graphs, hierarchical techniques like HEPV [14, 15], HWA [7] or HiTi [17] can be applied. Such hierarchical techniques consider graph-subgraph hierarchies by dividing a large graph into fragments and pushing common nodes between fragments to the higher level [14, 15], while a few others use hierarchical techniques to provide faster planning in game grids [5, 6]. We discuss some of these techniques in more details in Section 2.

The second strategy (one adopted in this paper) is to develop a federated approach that does not convert the multiple heterogeneous geographies into a single map. In particular, we adapt the existing connectivity relationships between different geographies to create a flexible multi-geography overlay through the notion of "anchor points". A least-cost path is constructed by a combination of least-cost path across geographies. There are several advantages of such an approach. First, it allows individual geographies to be treated as "black-boxes" - these geographies could may been created for different purposes by different experts. E.g., network representation for traffic planning and congestion control, raster/grid cell representation for building evacuation etc. Second, it allows each representation and map to evolve independently without requiring translation to a common grid or graph representation. For instance, office spaces within a building can be reconfigured in a raster grid, outdoor paths/obstacles can be added or removed in a vector graph. Third, it promotes better reuse of already developed map data and applications executing on it and encourages separation of concerns. Applications such as route planning can be executed without completely rewriting the domain specific code (that use individual representations optimized for those applications).

Specifically, the **main contributions** if this paper are:

- Design of a multi-geography overlay data structure that logically connects pre-existing multi-geography representations (Section 3).
- Design of a MGRP algorithm using the proposed multi-geography data structure to support weighted least cost path queries with sources and destination in different geographies. The algorithm is designed to be able to prune search space by using cached path segments (Section 4).
- Formalization of the utility-based static precomputing problem for MGRP, studying its complexity and developing a range of semi-greedy solutions for the problem (Section 5).
- Empirical evaluation of our approaches in the context of a large campus with multiple geographies at the indoor and outdoor scale and comparing the proposed solutions with existing caching techniques (Section 6).

We next cover related work in Section 2 and then formalize the MGRP problem and the multi-geography model in Section 3.

2. RELATED WORK

Traditional techniques for path planning include the Dijkstra and Bellman Ford algorithms [10, 16, 23]; optimizations have been proposed for these basic shortest path algorithms e.g. [12, 24]. Integration of different geographies for path planning has also been studied in the context of real-time robotic localization and navigation in indoor and outdoor geographies. Hybrid and hierarchical representations [22] of indoor/outdoor geographies have been explored [13, 20, 22] and used for real-time simultaneous localization and mapping of robots, typically for smaller, well-understood spaces. Grid based planning techniques, e.g. A*, popular in games, simulations and robotic path planning etc. can be expensive at high grid resolutions; optimization techniques such as Fringe A* [4] and hierarchical approaches [5, 6] have been proposed. Other approaches utilize multi-resolution planning [3] and creation of topological maps on grids [2].

Related work in the data management community has focused on aspects of scalability [9, 18, 19, 30], query optimization, precomputation and caching. For instance, shortest (least cost) paths have been used to support nearest neighbor queries [21, 26] in database applications. In [26] all pair shortest paths are precomputed and stored using shortest path quad-trees to aid processing k-NN queries. Early techniques for hierarchical path planning, e.g., HEPV (Hierarchical Encoded Path Views) [14, 15] incurred high planning costs (proportional to the total number of source and destination border nodes). While precomputation and caching can help with this, it is impractical in the multi-geography scenario where there can be large number of geographies and each geography can be large. To reduce precomputation costs Shekhar et al. [11, 28] studied partial memorization strategies including storing the costs of paths to higher level nodes, or costs of all source shortest paths in lower level subgraphs etc to study computation gain with impact on storage. Similar materialization based techniques for hierarchical representations have been explored by [8, 17]. Caching common data across all geographies or caching all paths within a geography is not sufficient in itself as the number of geographies increases.

On the commercial side, shortest paths have also been widely studied and used in intelligent transportation systems and web based map applications such as yahoo maps and games [29]. Web-based map services typically implement approximate shortest paths; much effort is placed on being able to render maps at multiple scales to answer user queries. Typically, shortest paths are determined on either on single large homogeneous maps, or on multiple resolutions of the same underlying representation (e.g., graphs or grids). Unlike existing web based route support systems, and intelligent transportation systems that primarily focus on outdoor maps, multigeography path planning in our case must integrate multiple indoor and outdoor maps that are heterogeneous and possibly overlapping. We believe our work has the potential to enable a new level of navigation and integrated travel systems that for example, combine road networks with pedestrian networks and indoor spaces.

3. MULTI-GEOGRAPHY MODELING

In this section we describe a multi-geography model that encapsulates different geographies connecting them topologically to provide a global view of the space. We start by covering issues related to individual geographies in Section 3.1. We then explain possible hierarchical organizations of multi-geographies in Section 3.2. Next in Section 3.3 we define the concept of an overlay network and formalize the MGRP problem. Finally, we cover the self-containment requirement imposed by the algorithm on each geography, which enables more structured and efficient path planning.

3.1 Individual Geographies

The Multi-geography $\mathcal{G} = \{G_1, G_2, \ldots, G_{|\mathcal{G}|}\}$ is a set of $|\mathcal{G}|$ geographies. Geographies in $\mathcal{G}$ are heterogeneous and can be of varying formats and resolutions. They can have overlapping regions representing the same regions in different formats. For instance, there could be a pedestrian walking network map and a transportation network map, which together cover different aspects of the same given region.

Each geography G_i has a type $T[G_i]$ associated with it, which can be a topological network, a raster image, or a vector map. These different types of geographies represent space differently. For instance, in the case of networks the geography is represented through a set of nodes/vertices and edges. Nodes represent geographical regions whereas edges represent paths from one geographical region to another. Associated with edges are weights that represent the cost of traversal from one node to another. Networks are commonly used for representing transportation/pedestrian networks, roads, and so on.

In case of raster representation, a geography is represented through a grid along a coordinate system. Each grid cell has a resistance/cost that represents the cost of traversal of the grid cell. Note that one could translate a grid representation into a network representation by creating a node for each grid cell and an edge between two neighboring grid cells. The weights of the edges would be the resistance of moving from one grid cell to another. Another representation is vector maps in which geographical entities are represented using polygons, lines, and points. Each map has a coordinate framework. Examples of these are CAD and GIS maps.

Each geography $G_i \in \mathcal{G}$ has an associated concept of points which are within the geography G_i. The exact representation of point $P \in G_i$ differs from geography to geography depending upon the type of the specific geography. In a raster geography it is a grid cell, and in the case of a network it is a node. In case of maps it is a point in the coordinate system of the map. In addition a point can be a named entity such as a building name or a room name within a building. Similarly, each geography $G_i \in \mathcal{G}$ has a concept of (direct) paths, or links, that exist within G between some pairs of points $P_i, P_j \in G$. Each link e_k has associated with it the cost of its traversal w_k. The links are directional, that is, the cost of traversing a link in the direction from P_i to P_j does not have to be equal to the cost of traversing the same link from P_j to P_i.

Given the above observations, for any source and destination points $P_{src}, P_{dst} \in G$ we use the standard graph theoretic definition to define the least cost path $LCP(P_{src}, P_{dst})$ between the two points for that geography. It must be noted the goal of MGRP is to find the least cost path, which can be the fastest path, shortest path, least resistance path, and so on. The criterion is reflected in the link weights. For instance, for the shortest path the weight can be the actual distance. For the fastest path it can be the time needed to traverse the link.

3.2 Hierarchy

Geographies in $\mathcal{G}$ are hierarchically interconnected and organized into multiple layers $L_1, L_2, \ldots, L_M$. Each geography $G \in \mathcal{G}$ belongs to a single layer/level in the hierarchy, denoted $L[G]$. The topmost layer L_1 consists of several different geographical maps of different regions from $\mathcal{G}$. Geographies in lower layers are sub-regions of top level geographies. For any geography $G_i \in \mathcal{G}$ the function $P[G_i]$ returns the parent geography of G_i. For each geography $G_i \in L_1$, function $P[G_i]$ return the logical root G_0.

Lower level geographies are either of the same representations as the top-level geographies, or part of a structural hierarchy. For instance, a raster grid of a room is a sub-geography of the larger raster grid of a floor. An example of structural hierarchy is an indoor grid map of a floor when it is a sub-geography of an outdoor map that contains the building footprint this floor belongs to.

While hierarchical layering can help in a more structured and efficient path planning by providing guidelines as to which geographies are next to be searched, hierarchies are *not* a requirement for the algorithm proposed in this paper. The proposed solutions will work irrespective of how we arrange the geographies in a hierarchy, e.g., it can work for a single-level flat organization, as should become clear from the subsequent sections. Of course, the efficiency of the algorithm will depend on the choice of hierarchical organization.

Figure 1 illustrates a sample 4-level multi-geography. Here the top level geographies are $L_1 = \{G_1, G_2\}$. They represent outdoor networks of two different regions. The second level geographies in this case are buildings. Figure 1 shows only two buildings G_3 and G_4, which are 3- and 2-story buildings from G_1 and G_2 respectively. Nodes e and f represent the exits to the stairwells on the first floor of G_3 which are also the exits to the outside of this building. Nodes c and d are exits to the stairwells on the second floor, and a and b – on the third floor. Each floor in this example is represented as a network where nodes are room exits and exits to the stairwells. E.g., G_5 corresponds to the third floor of building G_3. A room is represented as an obstacle grid. E.g., G_{10} is a room on the third floor of building G_3.

3.3 Overlay Network

Adjacent neighboring geographies are naturally interconnected with each other. Typically, each geography has a set of entrance and exit points, such that a path can exit a geography only at the exit point and enter the geography only at an entrance point. For instance, the set of doors in a building can serve as a set of entrance and exit points of the building, assuming the only way to get inside a building is through a door.

A point P_i in a geography G_i that has at least one direct link to another point P_j in another geography G_j is called an *anchor point* for that geography. Each geography $G_i \in \mathcal{G}$ has a set of anchor points $\mathcal{A}_i = \{A_{i1}, A_{i2}, \ldots, A_{i|\mathcal{A}_i|}\}$. Each anchor point $A_{im} \in G_i$ has at least one direct link to another anchor point $A_{jn} \in G_j$ in another geography $G_j \neq G_i$.

A directional link between two anchor points is called a *wormhole*. Each pair of anchors A_{im} and A_{in} of the same

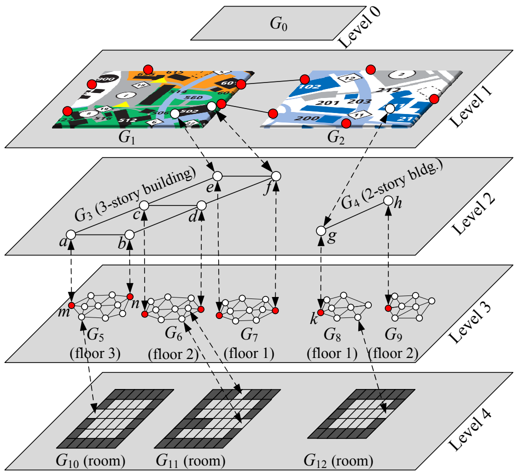

Figure 1: Multi-Geography Model.

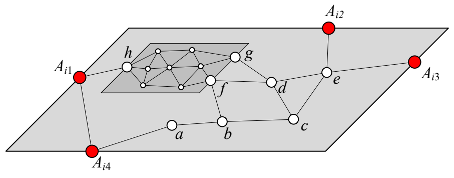

Figure 2: Sample Graph G_i.

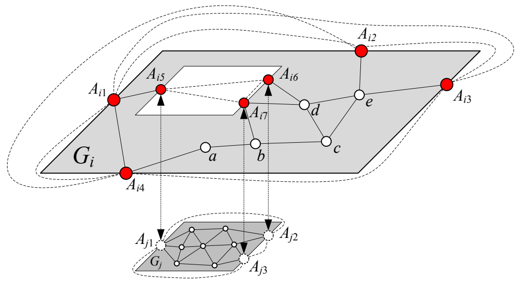

Figure 3: Hierarchy and Overlay Network. Wormhole connections from $\{A_{i1}, A_{i2}, A_{i3}, A_{i4}\}$ to $\{A_{i5}, A_{i6}, A_{i7}\}$ are not shown for clarity.

geography G_i are connected by the algorithm via an *internal* wormhole e_k. It corresponds to the least cost path $LCP(A_{im}, A_{in})$ between A_{im} and A_{in}. It should be noted that this $LCP(A_{im}, A_{in})$ is the absolute least cost path, and not the least cost path limited to only point from G_i. The cost w_k of link e_k is the cost of traversing this least cost path.

A directional link e_k between two anchors $A_{im} \in G_i$ and $A_{jn} \in G_j$ from two different geographies G_i and G_j is called an *external* wormhole. Wormhole e_k has associated with it the cost of its traversal w_k. This cost, for instance, can represent the delay of taking stairs between two adjacent floors in a building. While there can be multiple wormholes between geographies, we consider only the natural wormholes as candidates. That is, wormholes are only considered between geographies that overlap or are adjacent in spaces, e.g., stairs between adjacent floors. Specifically, a wormhole can only exist between geographies G_i and G_j, if one is the parent of the other, or if they are siblings and have a common parent, that is, if either $G_i = P[G_j]$, or $G_j = P[G_i]$, or $P[G_i] = P[G_j]$. A wormhole can therefore be classified as *horizontal* if it connects two siblings or *vertical* if it connects a child and its parent.

A vertical wormhole most often connects two anchors $A_{im} \in G_i$ and $A_{jn} \in G_j$ that correspond to the same point P in space via a link of cost zero. For instance, a building G_i and outdoor map G_j can be connected to each other at a doorway P of the building. For efficiency this case is represented as a single anchor that has presence in both the child and parent geographies.

The directional weighted graph formed by the set of all anchors for all the geographies and all wormholes is called the *overlay network*, or *overlay*, $\mathcal{O}$ for multi-geography $\mathcal{G}$. Overlay $\mathcal{O}$ will be employed to facilitate convenient path planning between geographies. Observe that any least cost path $LCP(P_{src}, P_{dst})$ from point $P_{src} \in G_i$ to $P_{dst} \in G_j$, where $G_i \neq G_j$, can be represented as: $LCP(P_{src}, P_{dst}) = LCP(P_{src}, A_{im}) \cdot \Big(LCP(A_{im}, A_{jn})\Big) \cdot LCP(A_{jn}, P_{dst})$. Here, A_{im} is an anchor point from geography G_i, and A_{jn} is an anchor point from G_j. The least cost path $LCP(A_{im}, A_{jn})$ can be computed completely inside the overlay network $\mathcal{O}$, abstracting out the details of intermediate geographies and drastically improving the efficiency.

Figure 2 illustrates the concepts defined in this section. It shows a flat geography G_i that consists of an outdoor road network (lighter shaded) and a room network of a 1-story building with exits f, g, and h (darker shaded). Figure 3 demonstrates a possible overlay network, wherein also the building is separated from G_i into a child subgeography G_j. All anchors of G_i are interconnected via wormhole links representing the corresponding (absolute) least cost paths. Pairs of anchors A_{i5}, A_{j1}, and A_{i6}, A_{j2} and also A_{i7}, A_{j3} represent the same physical point in space. Even though logically they are separated, in the actual implementation they are represented as a single node each for efficiency. Wormhole links among them are also not replicated.

3.4 Enforcing Self-Containment Property

The algorithm constructs overlays and, if needed, reorganizes geographies in $\mathcal{G}$ such that the *self containment property* for each geography $G_i \in \mathcal{G}$ holds.

DEFINITION 1. *Let $\mathcal{I}(G_i)$ be the geographic and overlay information, including nodes/points and links/wormholes, associated with geography G_i. A geography $G_i \in \mathcal{G}$ is* **self-contained** *if for any two points P_A and P_B from G_i the information stored in $\mathcal{I}(G_i)$ is sufficient to compute the least cost path $LCP(P_A, P_B)$, without using $\mathcal{I}(G_j)$ for any other geography G_j.*

Note specifically that $LCP(P_A, P_B)$ might not be fully inside G_i, but the information in G_i itself should still allow discovery of such a path. Figure 4(a) demonstrates such an example for two geographies G_1 and G_2 and points $P_A, P_B \in G_1$. The absolute least cost path $LCP(P_A, P_B) = P_A \rightarrow A_1 \rightarrow A_3 \rightarrow A_4 \rightarrow A_2 \rightarrow P_B$ is of length 6 and goes through G_1 and G_2. But if we limit the least cost path to be only inside G_1, then $LCP(P_A, P_B|G_1) = P_A \rightarrow P_B$ is of length 8. The algorithm always *enforces* the self-containment property and, as has been explained in Section 3.3, it adds a wormhole link between anchors A_1 and A_2 of G_1, as illustrated in Figure 4(b). This wormhole link is of length 4 and

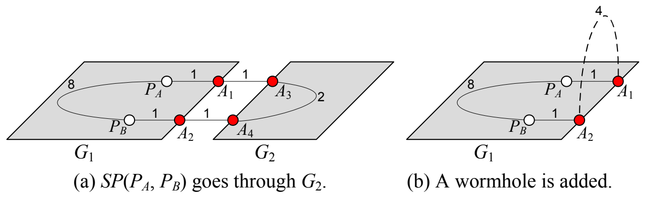

Figure 4: Example of Self-Containment.

corresponds to $LCP(A_1, A_2) = A_1 \rightarrow A_3 \rightarrow A_4 \rightarrow A_2$. Now, to compute $LCP(P_A, P_B)$ it is sufficient to use information $\mathcal{I}(G_1)$ only, since the wormhole link is a part of it.

3.5 Multi Geography Route Planning Problem

Given a hierarchical, layered multigeography $\mathcal{G} = \{G_1, G_2, \dots, G_{|\mathcal{G}|}\}$, where $G_i \in \mathcal{G}$ is **self-contained** and points, $P_{src} \in G_i$, $P_{dst} \in G_j$, G_i, $G_j \in \mathcal{G}$, find the least cost path, $LCP(P_{src}, P_{dst})$.

Our approach to solving MGRP builds upon A*, a goal-based path planning algorithm typically employed for grids [25]. We chose to base our solution on the A* technique as compared to traditional approaches such as Dijkstra due to its greater efficiency in terms of the search space explored. We develop extensions to A* to accommodate the hierarchical multi-geography model and implement multiple optimizations to improve performance and scalability without sacrificing on correctness of the least cost path. Key elements of our approach to solve the multi-geography route planning problem include:

1. Abstracting out details of individual geographies by designing and utilizing overlay network.
2. Optimizing representation of the overlay network by identifying and removing unnecessary nodes and links.
3. Using a hierarchical adaptation of A* algorithm to prune the search space (Section 4).
4. Exploiting path caching strategies to help to further improve the A* algorithm (Section 5).

We next describe the techniques that leverage the hierarchy to reduce the search space for more efficient path planning.

4. EXPLOITING HIERARCHIES

In this section we first present a brief overview of the original A* path finding algorithm in Section 4.1. We then describe our hierarchical A* approach in Section 4.2.

4.1 Original A* Path Finding Algorithm

In order to introduce the new A*-based approach let us briefly revisit the original A* algorithm [25]. Its pseudo code is illustrated in Figure 5. The task of A* is to find the least cost path $LCP(v_{src}, v_{dst})$ from point v_{src} to v_{dst}. The original A* algorithm maintains the set of already processed nodes S_{done}, which is initially empty, and the priority queue Q of the nodes to examine next, which initially contains just the source node v_{src}. The key of Q is the value of $f[v]$, explained next. For each node v the algorithm defines three values $d[v]$, $h[v]$, and $f[v]$. The value of $d[v]$ is the cost of the least-cost $v_{src} \rightsquigarrow v$ path observed thus far by the algorithm. The value of $h[v]$ is a lower bound on the least-cost

```
FIND-PATH-A-STAR($v_{src}, v_{dst}$)
1  $S_{done} \leftarrow \emptyset$    // Set of processed nodes
2  $Q \leftarrow \{v_{src}\}$    // Priority queue with $f[v]$ as key
3  $d[v_{src}] \leftarrow 0$    // Least cost distance from $v_{src}$
4  while $NotEmpty(Q)$ do
5      $x \leftarrow$ GET($Q$)
6      if $x = v_{dst}$ then
7          return $ReconstructPath(v_{src}, v_{dst})$
8      $S_{done} \leftarrow S_{done} \cup \{x\}$
9      for each $y \in$ GET-NEIGHBORS($x$) do
10         if $y \in S_{done}$ then
11             continue
12         $d \leftarrow d[x] +$ LINKCOST($x, y$)
13         if $y \notin Q$ then
14             $Put(Q, y)$
15             $h[y] \leftarrow$ HEURISTIC-DIST($y, v_{dst}$)
16         else if $d \geq d[y]$ then
17             continue
18         $came\_from[y] \leftarrow x$
19         $d[y] \leftarrow d$
20         $f[y] \leftarrow d[y] + h[y]$ // Est. dist. from $v_{src}$ to $v_{dst}$ via $y$
21 return failure

$ReconstructPath(v_{src}, v_{dst})$
1  $v \leftarrow v_{dst}$, $Path \leftarrow v_{dst}$
2  while $v \neq v_{src}$ do
3      $v \leftarrow came\_from[v]$
4      $Path \leftarrow v \cdot Path$
5  return $Path$
```

Figure 5: The A* Least Cost Path Algorithm.

$v \rightsquigarrow v_{dst}$ path, which is often computed as the straight line distance between v and v_{dst}, or by using heuristics. Value of $f[v]$ is an estimated length of the least cost $v_{src} \rightsquigarrow v \rightsquigarrow v_{dst}$ path which is computed as $f[v] = v[u] + h[v]$. The algorithm retrieves from the priority queue Q node x, with the lowest $f[x]$. The algorithm is constructed such that when x is extracted from Q, its $d[x]$ is guaranteed to be the cost of the least cost path $LCP(v_{src}, x)$ in the graph and the path itself can be reconstructed by invoking $ReconstructPath(v_{src}, x)$ procedure. If $x = v_{dst}$ then the algorithm terminates by returning the corresponding least cost path. Otherwise, it examines each neighbor y of x inserting them in Q when necessary and updating $d[y]$, $h[y]$, and $f[y]$ correspondingly.

The original A* algorithm can be applied to the MGRP problem. It will be able to successfully find the least cost path $LCP(P_{src}, P_{dst})$ for points P_{src} and P_{dst}, provided that it also takes into consideration the anchor nodes and wormhole links. However, the efficiency of the algorithm can be significantly improved by taking into account the hierarchies and by employing caching strategies, as will be discussed in the subsequent sections.

4.2 Hierarchical Adaptation of A*

In this section we develop a hierarchical adaptation of the A* path finding algorithm. The new solution employs the hierarchy to prune the search space for achieving better efficiency. Specifically, we will explore three techniques to limit path search. The first one allows to skip certain subgeographies from consideration, the second exploits the least common ancestor of the source and destination geographies, and the third one limits the search space when passing through a geography. All three techniques are implemented as part of GET-NEIGHBORS(x, v_{dst}, G_{LCA}) procedure used by the A* algorithm which now takes in two additional parameters v_{dst} and G_{LCA}. Parameter G_{LCA} will be explained

```
Get-Neighbors $(x, v_{dst}, G_{LCA})$
// $G_{LCA} = LCA(G[v_{src}], G[v_{dst}])$
1   $R \leftarrow \emptyset$ // Result set
2   if $IsExterriorAnchor(x)$ and $v_{dst} \notin Tree(G[x])$ then
3       for each $x \rightarrow y \in ExteriorLinks(x)$ do
4           if $y \notin P[G_{LCA}]$ then
5               $R \leftarrow R \cup \{y\}$
6   else
7       for each $x \rightarrow y \in AllLinks(x)$ do
8           if $y \in P[G_{LCA}]$ then
9               continue
10          if $v_{dst} \notin Tree(G[y])$ then
11              continue
12          $R \leftarrow R \cup \{y\}$
13  return $R$
```

Figure 6: Hierarchical Pruning.

later on in this section. The pseudo code of the procedure is presented in Figure 6. We will use the example in Figure 1 to better illustrate the concepts described in this section.

Avoiding Certain Subgeographies. Assume that A* is invoked to find the least cost path $LCP(v_{src}, v_{dst})$ from v_{src} to v_{dst}. For instance, v_{src} and v_{dst} could be two cells inside the rooms G_{10} and G_{11} respectively in Figure 1. Assume that at the current step the algorithm observes path $v_{src} \rightsquigarrow x$ and analyzes each of its neighbors $y \in$ Get-Neighbors(x) and the corresponding paths $v_{src} \rightsquigarrow x \rightarrow y$. For instance, x could be node e on Level 2 in Figure 1.

Suppose that x belongs to geography G_i. Let G_j be any child subgeography of G_i. Let $Tree[G_j]$ be the subtree of the hierarchy rooted at G_j. Subtree $Tree[G_j]$ contains G_j, its children, children of its children, and so on. In Figure 1, G_i is G_3 and G_j is G_7.

Observe that if $v_{dst} \notin Tree(G_j)$ then there is no need to go inside geography G_j. That is, paths $v_{src} \rightsquigarrow x \rightarrow y$ where $y \in Tree(G_j)$ need not be considered and can be pruned away. This is because since $v_{dst} \notin Tree(G_j)$ such a path would first leave G_i from some anchor point $A_m \in G_i$ and then return back to G_i via another anchor point $A_n \in G_i$. But all of the geographies are self-contained and thus since $A_m, A_n \in G_i$ it follows that $LCP(A_m, A_n)$ can be computed from $\mathcal{I}(G_i)$ alone, without considering $Tree(G_j)$. Observe that this is the case even if portions of path $LCP(v_{src}, v_{dst})$ actually go via geography G_j, as they will be captured by the wormhole links in G_i. Figure 1 illustrates these observations, e.g. we can see that there is no need to go inside floor G_7 since v_{dst} does not belong to it.

Avoiding considering such $v_{src} \rightsquigarrow x \rightarrow y$ paths greatly reduces the search space of the A* algorithm, making it significantly more efficient.

Least Common Ancestor. Let $v_{src} \in G_i$, $v_{dst} \in G_j$, and G_k be the least common ancestor (LCA) of G_i and G_j in the hierarchy. Observe that least cost path $LCP(v_{src}, v_{dst})$ is contained entirely in the set of geographies from $Tree(G_k)$. Consequently, when exploring neighbors y of x in the context of $v_{src} \rightsquigarrow x \rightarrow y$ paths the neighbors that do not belong to geographies from $Tree(G_k)$ can be pruned away. The only case where path $v_{src} \rightsquigarrow x$ is contained in $Tree(G_k)$ whereas $v_{src} \rightsquigarrow x \rightarrow y$ is not is when x is in G_k and y is in its parent geography $P[G_k]$. Thus, for pruning in the context of $v_{src} \rightsquigarrow x \rightarrow y$ paths it is sufficient to check whether y is in $P[G_k]$. If G_k is the root geography G_0 then this pruning strategy does not apply.

Passing Through a Geography. To explain another pruning strategy we will need to make several definitions. An anchor A_k is called an *exterior* anchor of geography G_i if it is connected via a wormhole to another anchor in geography $G_j \neq G_i$ such that G_j is not a child of G_i. An anchor A_k is in *interior* anchor if it is not an exterior anchor. A wormhole link between two exterior anchors is an *exterior* wormhole link. In Figure 3 anchors $\{A_{i1}, A_{i2}, A_{i3}, A_{i4}\}$ are exterior anchors and $\{A_{i5}, A_{i6}, A_{i7}\}$ are interior anchors of geography G_i, and wormhole links among $\{A_{i1}, A_{i2}, A_{i3}, A_{i4}\}$ are exterior links.

Consider again path $v_{src} \rightsquigarrow x \rightarrow y$. When x is an exterior anchor of geography G_i and v_{dst} does not belong to $Tree(G_i)$ this means the algorithm is simply passing through G_i without going into any of its children, since the children do not contain v_{dst}. Thus in such cases there is no need to consider edges incident to node x except for the wormhole links that lead to other exterior anchors. Often an exterior anchor A of geography G would have a significant fraction of its connections to be wormhole links to the interior nodes of G and this pruning strategy helps to avoid considering such connections effectively.

4.2.1 Exploiting Intrageography Hierarchies - Regionalization

In Section 3.3 we have discussed that any least cost path $LCP(P_{src}, P_{dst})$ can be represented as a sequence of least cost paths $LCP(P_{src}, A_{im}) \cdot LCP(A_{im}, A_{jn}) \cdot LCP(A_{jn}, P_{dst})$. Thus far we have focused on optimizing the $LCP(A_{im}, A_{jn})$ path that goes entirely inside the overlay network. In this section we discuss a hierarchical technique that optimizes the *local* least cost planning part that corresponds to paths $LCP(P_{src}, A_{im})$ and $LCP(A_{jn}, P_{dst})$.

It is possible that the internals details of a local geography G_i are hidden from the overall system. That is, G_i may be available only as a black box with the interface for computing the least path inside G_i for any two points in G_i. In that case the technique described in this section does not apply. However, often geographies are not provided as black boxes and amenable to hierarchical optimization techniques. Such techniques have already been explored in the past especially in the context of grids. In our work we use the regionalization technique from [5] with minor modifications.

Given a grid map, the idea is to use a region decomposition algorithm to identify smaller regions which might, for instance, correspond to rooms on a floor. Then the exit grid cells are found between regions. The overlay network is created between neighboring exits where the nodes correspond to the exit grid cells and edges to the least cost paths between them. This overlay network is then employed for faster path finding.

The region decomposition algorithm [5] starts at the top leftmost free cell that is not assigned to a region and then proceeds right until it hits an obstacle, then continues downward filling the region. The method detects if the region has shrunk left or right. and if the region re-grows after shrinking, then it stops, removing extra filled cells if needed.

We have discovered that, as is, the decomposition technique [5] does not work effectively on indoor maps, especially when rooms are differently shaped and/or irregular. Specifically, it generates many small regions with long common borders resulting in an unnecessarily large number of exists. To address this problem we have implemented several modifications that (a) bound the growth of a region

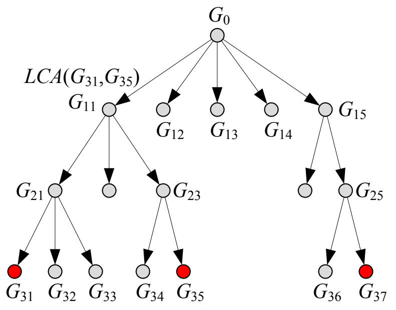

Figure 7: Geography Hierarchy Graph.

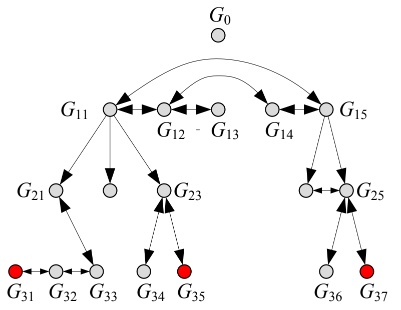

Figure 8: Geography Connectivity Graph.

to prevent the creation of long borders; (b) merge certain regions to form more natural subregions with smaller borders; and (c) eliminate redundant exits. This has resulted in a drastic reduction in the number of exits, leading to a better overall performance. The details of these techniques are covered in [1]. The algorithm guarantees that the regionalization maintains the optimality of the MGRP by the way the anchors are created and exits are placed. The effectiveness of the modified algorithm has been validated on different floor maps and complex building plans. The impact of regionalization on MGRP will be studied in Section 6.

5. CACHING STRATEGIES

In this section we discuss caching strategies for MGRP. First in Section 5.1 we present key observations about the geographies that must be traversed by a given path. These observations will lead to a design of two types of caches described in Section 5.2. The physical organization of these caches will be discussed in Section 5.3. Finally, Section 5.4 will cover the utility-based semi greedy strategy for deciding the best content of the cache.

5.1 Observations that Motivate Caching

To illustrate how caching can be employed consider Figures 7 and 8. Figure 7 shows a sample geography hierarchy graph, where each node corresponds to a geography and a directed edge representing a parent-child relationship. Figure 8 demonstrates a possible connectivity graph for this scenario. There, nodes correspond to geographies and a directed edge is created between any two geographies G_i and G_j if there is an anchor in G_i that is connected to an anchor in G_j via a wormhole link. The links in Figure 8 are bidirectional implying there are connections in both directions. Figures 7 shows for instance that geography G_{21} is the parent of G_{31}. At the same time Figure 8 shows that there is no direct connection between G_{21} and G_{31} and that G_{31} is connected to G_{21} only indirectly via siblings G_{32} and G_{33}.

Assume that the goal is to find the least cost path between points $P_{src} \in G_i$ and $P_{dst} \in G_j$. Let G_{LCA} be the least common ancestor of G_i and G_j in the geography hierarchy graph. For instance, in Figure 7, we might have $G_i = G_{31}$, $G_j = G_{35}$, and $G_{LCA} = G_{11}$. Let us define *source geography chain* $\mathcal{G}_{ij}^{src}$ for G_i and G_j as the sequence of geographies in the $G_i \rightsquigarrow G_{LCA}$ path in the hierarchy graph, except for G_{LCA}. Similarly, we can define the *destination geography chain* $\mathcal{G}_{ij}^{dst}$ for G_i and G_j as the sequence of geographies in the $G_j \rightsquigarrow G_{LCA}$ path except for G_{LCA}. Continuing with our example in Figure 7, we have $\mathcal{G}_{ij}^{src} = \{G_{31}, G_{21}\}$ and $\mathcal{G}_{ij}^{dst} = \{G_{23}, G_{35}\}$.

We can observe that if $LCP(P_{src}, P_{dst})$ exists then for any geography connectivity graph this path must pass through each of the geographies in $\mathcal{G}_{ij}^{src}$ and $\mathcal{G}_{ij}^{dst}$. This statement is trivial for geographies G_i and G_j as they contain the source and destination points. Let us prove it for the rest of the geographies in $\mathcal{G}_{ij}^{src}$ and $\mathcal{G}_{ij}^{dst}$. The proof is based on the observation that, by construction, the connectivity in the overall graph is such that for a geography G_k its $Tree(G_k)$ is directly connected to the rest of the graph only via G_k. Recall that by construction a geography can only be connected to its parent, its children, or its siblings. Thus for a path the only way in or out of $Tree(G_k)$ is through G_k.

For $\mathcal{G}_{ij}^{src}$ we can see that if parent $P[G_i]$ of G_i is in $\mathcal{G}_{ij}^{src}$ then the path must pass through it. This is because otherwise, the path will never be able to leave $Tree(P[G_i])$ subtree (to be more precise, $Tree(P[G_i]) \setminus P[G_i]$) of the hierarchy and thus will never be able to reach the destination. Similar logic applies to the parent of $P[G_i]$ and so on until G_{LCA} is reached. If the children geographies of G_{LCA} are not interconnected then the path must reach G_{LCA}, if they are interconnected however then the path might not reach G_{LCA} and go directly via its children instead.

The same logic applies to $\mathcal{G}_{ij}^{dst}$. A path that is not inside $Tree(G_k)$ could enter it only via G_k. Thus the geographies in $\mathcal{G}_{ij}^{dst}$ must be visited since P_{dst} belongs to the corresponding subtrees. Similarly, G_{LCA} will also be visited if its children are not interconnected, and it might not be visited if they are interlinked.

For instance, for Figures 7 and 8, when $G_i = G_{31}$ and $G_j = G_{35}$ path $LCP(P_{src}, P_{dst})$ will include geographies $G_{31} \to G_{32} \to G_{33} \to G_{21} \to G_{11} \to G_{23} \to G_{35}$. Thus it will pass through $\mathcal{G}_{ij}^{src} = \{G_{31}, G_{21}\}$ and $\mathcal{G}_{ij}^{dst} = \{G_{23}, G_{35}\}$. Since the children of $G_{LCA} = G_{11}$ are not interconnected it will also pass trough G_{11}. An example where $LCP(P_{src}, P_{dst})$ will not pass through G_{LCA} is when $G_i = G_{31}$, $G_j = G_{37}$, and $G_{LCA} = G_0$.

5.2 Two Types of Caches

With the help of the observations from the Section 5.1 we can define two types of caches to speed up the MGRP algorithm.

5.2.1 Node to Geography Cache

The first type of cache is the node to geography (NG) cache. Assume that the algorithm looks for $LCP(v_{src}, v_{dst})$ path and currently explores $v_{src} \rightsquigarrow x$ intermediate path. Let $G_i = G[x]$ be the geography of x and $G_j = G[v_{dst}]$ be the geography of v_{dst}. Let $\mathcal{G}_{ij}$ be the sequence that includes (1) the geographies in $\mathcal{G}_{ij}^{src}$, (2) geography $G_{LCA} = LCA(G_i, G_j)$, which is included if children of G_{LCA} are not interlinked, and (3) geographies in $\mathcal{G}_{ij}^{dst}$. Then we know that $LCP(v_{src}, v_{dst})$ must pass through all the geographies in $\mathcal{G}_{ij}$.

Suppose that for a geography $G_m \in \mathcal{G}_{ij}$ we have cached the least cost paths from x to all of the anchors of G_m and their costs. Then instead of exploring direct links/edges of x we can jump directly to geography G_m by treating the cached least cost paths as indirect links to G_m. This is because the path must pass through G_m and the only way inside G_m is via its anchors. Intuitively, the closer G_m is to the destination geography G_i in $\mathcal{G}_{ij}$, the more exploration steps of the algorithms will be skipped and hence the more efficient this optimization will be.

Notice that for this optimization to work, path from x to *all* of the anchors of G_m should be cached. Assume that this is not the case and one of anchors $A_k \in G_m$ is omitted. Since $LCP(v_{src}, v_{dst})$ might go through A_k, for correctness, the A* algorithm now will need to explore not only the indirect neighbors of x, but also all of the direct neighbors, defeating the purpose of this optimization.

Let us use Figure 1 to illustrate this idea of caching. There, P_{src} can be a point inside room G_{12}, P_{dst} a point inside G_{10}, and x can be an anchor k of G_8. Assume that the least cost paths from x to all anchors of G_5 are cached. Then instead of exploring direct neighbors of x in the context of $v_{src} \rightsquigarrow x$ paths, the algorithm can jump directly to the anchor points of G_5, avoiding many of explorations and thus reducing the search space.

To implement this NG caching policy the beginning of GET-NEIGHBORS(x, v_{dst}, G_{LCA}) procedure will need to be modified as illustrated in Figure 9. The idea is for path $v_{src} \rightsquigarrow x$ to keep track of its geography chain $\mathcal{G}_{ij}$. Then if paths from x to some of the geographies in $\mathcal{G}_{ij}$ are cached then simply jump to the geography that is closest in the hierarchy to the destination geography G_j. The LINKCOST(x, y) procedure in Figure 5 will also need to be modified for the indirect links to get their cost from the NG cache. Similarly, *ReconstructPath*(v_{src}, v_{dst}) procedure for indirect links will need to get the cached portion of the path from the NG cache.

5.2.2 Geography to Geography Cache

The second type of cache is the geography-to-geography (GG) cache. The GG cached can be viewed as a two dimensional $|\mathcal{G}| \times |\mathcal{G}|$ array GG. This array can be disk-based but in practice it is small and can easily fit in memory. Each its element GG_{ij} caches the set of geographies that can be traversed next on a path originated from a geography $G_i \in \mathcal{G}$ and with a destination in the geography $G_j \in \mathcal{G}$. Now, when the algorithm analyzes $v_{src} \rightsquigarrow x \to y$ intermediate path, if geographies $G[x]$ of x and $G[y]$ of y are different, and if y is not in any of the geographies in $GG[G[x], G[y]]$ then $v_{src} \rightsquigarrow x \to y$ path can be pruned away. This pruning strategy is reflected in Lines 12, 13 and 18, 19 in Figure 9.

For the case in Figure 8, if $G_i = G_{33}$ and $G_j = G_{35}$ then $GG_{33,35} = \{G_{21}\}$. From this example we can see that when looking for the least cost path $LCP(P_{src}, P_{dst})$, where $P_{src} \in G_{33}$ and $P_{dst} \in G_{35}$, if GG cache is not used then the algorithm might proceed exploring nodes in G_{32} and G_{31}. Using the GG cache, however, we can determine that for path $LCP(P_{src}, P_{dst})$ the only feasible geography after G_{33} is G_{21} and the geographies G_{32} and G_{31} need not be explored.

```
GET-NEIGHBORS($x, v_{dst}, G_{LCA}$)
1   $R \leftarrow \emptyset$// Result set

2   $\mathcal{G}_{ij} \leftarrow ComputeSrcToDstChain(x, v_{dst})$
3   for $k \leftarrow |\mathcal{G}_{ij}|$ to 1 do
4       $G \leftarrow \mathcal{G}_{ij}[k]$
5       if $NotInNGCache(x, G)$ then
6           continue
7       for each anchor $A \in G$ do
8           $R \leftarrow R \cup \{A\}$
9       return $R$

10  if $IsExteriorAnchor(x)$ and $v_{dst} \notin Tree(G[x])$ then
11      for each $x \to y \in ExteriorLinks(x)$ do
12          if $G[x] \neq G[y]$ and $y \notin GG[G[x], G[v_{dst}]]$ then
13              continue
14          if $y \notin P[G_{LCA}]$ then
15              $R \leftarrow R \cup \{y\}$
16  else
17      for each $x \to y \in AllLinks(x)$ do
18          if $G[x] \neq G[y]$ and $y \notin GG[G[x], G[v_{dst}]]$ then
19              continue
20          if $y \in P[G_{LCA}]$ then
21              continue
22          if $v_{dst} \notin Tree(G[y])$ then
23              continue
24          $R \leftarrow R \cup \{y\}$
25  return $R$
```

Figure 9: GET-NEIGHBORS() for NG and GG Caching.

Each element GG_{ij} of the GG cache are computed by analyzing all of the least cost paths from each anchor of geography G_i to geography G_j using one of the known all pair least cost paths algorithms [27]. From these paths the set of the next geographies that follow G_i can be trivially deduced.

5.3 Physical Cache Organization

We use physical cache organization that is similar to that of HEPV [15]. We cache only anchor nodes though the same ideas apply to any nodes in general. Assume that there are n anchors in total. Then the *complete* NG cache can be viewed as an $n \times n$ matrix NG. This matrix stores compactly the least cost paths between all pairs of anchors, where each element N_{ij} of NG stores information about the least cost path $LCP(A_i, A_j)$. Specifically, for path $A_i \to A_k \to A_\ell \rightsquigarrow A_j$, entry N_{ij} stores the cost of the path and the next hop anchor to be traversed from A_i, which is A_k. Consequently, the N_{kj} entry will in turn contain A_ℓ, and so on, allowing to reconstruct the sequence of anchors for the least cost path $LCP(A_i, A_j)$. The actual physical path is constructed from this sequence of anchors with the help of the overlay network, as it stores on disk the actual paths that correspond to the wormhole links between anchors.

For the incomplete NG cache some of its entries can be empty. To avoid pointing to the next hop entry that is empty, the N_{ij} entry now contains a sequence of anchors in the $LCP(A_i, A_j)$ path that ends with the first cached entry or with the destination anchor A_j. For instance, for LCP $A_i \to A_k \to A_\ell \rightsquigarrow A_j$ if N_{kj} is empty but $N_{\ell j}$ is not, the N_{ij} will contain the sequence A_k, A_ℓ instead of simply the next hop A_k. The NG cache is implemented as a disk-resident hash table with the source and the destination anchor pair as the key.

As explained in Section 5.2.2, in practice the GG cache can be represented as a small memory resident array. However, if necessary, it can also be represented as a disk-resident hash table similar to the NG cache.

5.4 Caching Strategies

The complete NG cache can be large for large geographies ($O(N^2)$ where N is the total number of anchors) and might not fit into the available storage space. Thus a solution might be preferred where only some of the elements of the complete of NG cache are present in the cache. This would create the storage size versus efficiency tradeoff, as a large cache size would lead to a more efficient processing. A strategy would also need to be developed to decide which elements to cache and which not to cache. Before we discuss the caching strategy employed by the proposed MGRP solution, let us formalize the problem of selecting the content of the cache.

5.4.1 *Formalizing Cache Content Selection Problem*

Assume that the size of the NG cache is restricted to be no greater than S. Let NG_{ij} be each cache entry storing the cost and path information for path $LCP(A_i, A_j)$. Each entry occupies some disk space s_{ij}. In terms of speeding up the computations, each entry has a benefit μ_{ij}^{cached} if cached, and a benefit $\mu_{ij}^{notcached}$ if not cached. The befit reflects the number of explorations needed by A* algorithm to discover $LCP(A_i, A_j)$. These explorations will be avoided if the path is cached. While benefit $\mu_{ij}^{notcached}$ is 0, the benefit μ_{ij}^{cached} is much more complex to compute. For instance, caching path $LCP(A_i, A_j)$ impacts the cost of any least cost path $A_k \rightsquigarrow A_i \rightsquigarrow A_j$.

Suppose that there are K anchors in total in $\mathcal{G}$. For each pairs of anchors A_i and A_j let n_{ij} be the number of times $LCP(A_i, A_j)$ will be invoked. Let the decision variable d_{ij} take the value of 1 if path $LCP(A_i, A_j)$ is cached and 0 if it is not cached. Then the goal is to maximize the benefit of the cache given the storage limitations:

$$\begin{cases} \text{Maximize} \displaystyle\sum_{i=1}^{K}\sum_{j=1}^{K} n_{ij}\left(d_{ij}\mu_{ij}^{cached} + (1-d_{ij})\mu_{ij}^{notcached}\right) \\ \text{subject to:} \\ \quad \displaystyle\sum_{i=1}^{K}\sum_{j=1}^{K} s_{ij}d_{ij} \le S \end{cases} \qquad (1)$$

Since $\mu_{ij}^{notcached}$ is zero, the part $(1-d_{ij})\mu_{ij}^{notcached}$ evaluates to zero as well. If we assume that μ_{ij}^{cached} and s_{ij} can be any constant independent values, then we can see that this problem is a traditional combinatorial optimization problem and can be reduced from a 0-1 knapsack problem directly and hence is NP-hard. However, in our case μ_{ij}^{cached} variables have dependencies that are hard to model accurately. The actual benefit of any cached entry depends on the number of steps skipped in the path planning as a result of caching this segment of data. It is impacted by such factors as which other entries are cached, the length of the path, the topology of the graph, and the heuristic employed during A* process.

5.4.2 *Semi-Greedy Utility Based Caching*

Characterizing the utility of the cached data is difficult due to the different variables and factors affecting it. One solution is to estimate the utility μ_{ij}^{cached} of NG_{ij} using sample A* runs between anchors A_i and A_i to evaluate the impact of NG_{ij} on different paths in terms of number of node visits saved. We will describe a solution that employs this method to estimate utilities to compute the cache using a semi-greedy strategy. The proposed solution for determining the content of the NG cache consists of the following two steps:

1. Estimating the cost C_{ij} of running A* between anchors A_i and A_j. The cost C_{ij} is indicative of how many steps the algorithm can skip if the path is cached.

2. Estimating the number of the least cost paths paths $A_k \rightsquigarrow A_i \rightsquigarrow A_j$ which have the same destination A_j as the least cost path $LCP(A_i, A_j)$ and hence can use the cached path $LCP(A_i, A_j)$ for faster MGRP.

The brute force solution for accomplishing the first task mentioned above is to run A* algorithm for each pairs of nodes A_i and A_i to determining the cost C_{ij}. The cost C_{ij} represents the number of nodes visited when computing A* between A_i and A_j. While the above strategy provides a reasonable estimate of benefit of caching, the drawback is that it requires running A* algorithm $O(K^2)$ times for K anchors. When K is large this solution is undesirable. We employ sampling to overcome this problem. For each pair of geographies G_m and G_n we choose some sample anchor points $\{A_{m1}, A_{m2}, \ldots, A_{mk}\} \in G_m$ and $\{A_{n1}, A_{n2}, \ldots, A_{n\ell}\} \in G_n$ and compute the cost for each A_{mi} and A_{nj} pair. Then, for the sampled anchors the cost is set to the actual computed costs. For the rest of the anchors for these two geographies the cost is set to the average sampled cost.

The second challenged is to determine which anchor pairs will potentially use the cached entry NG_{ij} for path $LCP(A_i, A_j)$. The naive solution is to first compute all least cost paths between all pairs of anchors. Then, to determine for each $LCP(A_i, A_j)$ every other least cost path $A_k \rightsquigarrow A_i \rightsquigarrow A_j \rightsquigarrow A_\ell$ it is a subpath of. This is expensive both computation and storage wise. To reduce this cost, we will make a simplifying assumption and consider only least cost paths of the form $A_k \rightsquigarrow A_i \rightsquigarrow A_j$ that have the same destination A_j as $A_i \rightsquigarrow A_j$. We then compute the least cost path tree $SPTree(A_j)$ for each anchor A_j. Naturally, any least cost path $A_k \rightsquigarrow A_j$ is affected by the least cost path $A_i \rightsquigarrow A_j$ if A_k belongs to the subtree of $SPTree(A_j)$ rooted at A_i. This is since such a $A_k \rightsquigarrow A_j$ will have to pass through A_i. Thus, by traversing the least cost path tree we can determine the set $\mathcal{P}_{ij}^{imp}$ of all the least cost paths impacted by NG_{ij}, including $LCP(A_i, A_j)$ itself.

The benefit μ_{ij}^{cached} of caching $LCP(A_i, A_j)$ is computed as the expected saved computations from caching this path. When the path is cached, instead of performing C_{ij} explorations by $A*$, the algorithm will now need to perform one traversal of the indirect link for the cached path. Similarly, for the rest of the paths in $\mathcal{P}_{ij}^{imp}$ the benefit will be proportional to C_{ij}. Thus the benefit is computed as γC_{ij} per each path in $\mathcal{P}_{ij}^{imp}$, where $\gamma \in (0, 1]$ is a coefficient of proportionality. But since maximizing $\sum\sum \gamma n_{ij} d_{ij} \mu_{ij}^{cached}$, see System (1), is the same as maximizing $\sum\sum n_{ij} d_{ij} \mu_{ij}^{cached}$, the γ factors out leading to the overall benefit function $\mu_{ij}^{cached} = |\mathcal{P}_{ij}^{imp}| C_{ij}$.

To select the best anchor-geography pair to cache in the NG cache, for each anchor A_i the algorithm keeps track of overall benefit of caching paths from A_i to all anchors of each geography G_m, which is computed as $\sum_{A_j \in G_m} \mu_{ij}^{cached}$. The anchor-geography pairs to put into the NG cache are then chosen using either static or incremental strategies.

The static greedy strategy puts in the NG cache the top k anchor-geography pairs with the maximum estimated benefit, such that they all fit into the allowed space S. In the incremental greedy strategy, the highest-benefit pairs A_i and G_m are added to the NG cache iteratively one by one. After a pair is added on one iteration, some of the affected benefits μ_{ij}^{cached} will be computed differently compared to the previous iterations. Specifically, if for $LCP\ A_i \rightsquigarrow A_k \rightsquigarrow A_j$ its subpath $A_k \rightsquigarrow A_j$ is already cached, then A* algorithm will need perform proportional to $C_{ij} - C_{kj}$ explorations to discover this path. This formula reflects the original cost, with the cost of already discovered subpath subtracted. After factoring out the γ proportionality coefficient, the benefit is now computed as $\mu_{ij}^{cached} = |\mathcal{P}_{ij}^{imp}| S_{ij}$. Here, $S_{ij} = C_{ij}$ if no subpath of $LCP(A_i, A_j)$ is cached and $S_{ij} = C_{ij} - C_{kj}$ for the longest cached $A_k \rightsquigarrow A_j$ subpath. The iterations are repeated until the space limit S is exceeded.

The static greedy approach has the advantage of being a faster algorithm to create the cache. However, the full impact of the relationships between path segments is not taken into account when caching.

5.4.3 *Factoring in Access History*

The above solution assumes that every path has an equal probability of being accessed. However, in practice this might not be the case and the likelihood of certain paths being accessed are higher than others. This will impact the caching strategy. For instance clearly there should be little benefit of caching a path that is unlikely to be accessed. To account for the actual access history, in addition to the method described above, we explore a second utility based approach. To estimate the access patters we run some sample test runs on a smaller sized NG cache. We determine the number of requests β_{ij} sent for the NG_{ij} entry of the NG cache by the algorithm. The benefit is then computed as $\mu_{ij}^{cached} = \sum_{k:A_k \rightsquigarrow A_j \in \mathcal{P}_{ij}^{imp}} (\alpha + \beta_{kj})(S_{ij} - 1)$. This formula assigns to each path the importance of $(\alpha + \beta_{kj})$. Here α is the base level importance of a path which is set to 1. By considering both the utility in terms of search area saved and the actual access patterns, the algorithm computes a better utility value that results in more efficient path computations during the run time.

6. EXPERIMENTAL RESULTS

This section presents the experimental setup and the results of our strategies. First we describe the data preparation process for the campus related geographical data.

6.1 Geography Data Creation

Testing has been done on real geographic GIS and CAD data for a section of the UC Irvine campus. From the GIS perspective, both an aerial view of the campus and layers modeling buildings, dorms, walking paths and main roads have been stored within the database. The CAD maps representing the campus buildings at the floor level have been rasterized manually and loaded within the database. The outdoor GIS map has been converted to an outdoor resistance grid: every cell of the grid has a different resistance value according to the nature of the cell (free, obstacle/building, surface type, etc). A pedestrian network (consisting of walkways) and transportation network (consisting of roadways) of the outdoor area have also been created. Wormholes between indoor and outdoor maps (typically doors, stairs, etc) have been identified and connected to meaningful waypoints on the map (e.g., intersections between different walking paths). Our preliminary analysis revealed that a 2-level geography (3-level with regionalization) was the most natural and meaningful representation for UCI campus dataset we had. The test data consists of 123 buildings with each floor in the building considered a single geography and in total there are about 383 indoor grids. Since creation of these raster grids requires considerable manual effort, we have cloned existing raster maps to stress test the algorithms. At the top level there are a total of 1971 anchors. With regionalization, we have a 3-level graph with approximately 60,000 anchors. The anchor overlay network has also been precomputed.

6.2 Experimental Setup

Input to the experiments comes from a query generator which generates sets of 5000 random queries based on a uniform distribution. Both the geographies and the points within the geographies are selected randomly based on the uniform distribution. The random queries select any source destination in different geographies and hence the queries can be between two floors in a building, between indoor and outdoor geographies, or between two outdoor geographies.

Data representation in the cache. The NG matrix is represented in the disk in a row major fashion. The rows are indexed by the source anchor id, and represent all the paths from a source A_i to all other anchors. Each column in the row is indexed by the destination anchor id. The columns are clustered based on the geographies the destination anchors belong to, and further ordered by their anchor ids. A memory index for each row contains the start id of each block in disk, and this id is a hash of the anchor id and the corresponding geography id. This allows the data manager to determine which block to retrieve based on either destination anchor id or geography id. The right block(s) can be retrieved for a single path query (single source and destination), or for a query which requests cost from an anchor to all anchors in a given geography.

Metrics. The main performance metrics are actual running time in milliseconds, and the number of number of nodes visited. The number of nodes visited indicates the search space of the algorithm and hence the complexity in terms of updating the costs and finding the path. This gives an indication of the improvement irrespective of the implementation details and data structures used which can impact the running time. For caching we also study the number of cache accesses performed, cache hit rate and I/O performance for the different strategies.

In this paper we cover only the main set of experiments that deal primarily with caching issues. A much more extensive set of experiments that cover various aspects of our approach can be found in [1].

6.3 Experimental Results

To understand the value of the basic MGRP algorithm (with no caching) we compare the MGRP path-planning mechanism with other existing planning techniques. We use the basic A* as our starting point; it has been shown to have

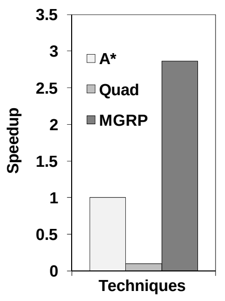

Figure 10: Cross Comparison.

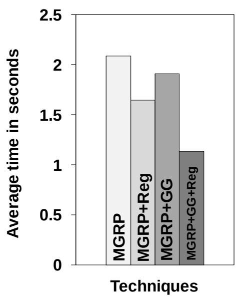

Figure 11: Impact of Optimizations.

better search directionality than Dijkstra resulting in lesser number of searched nodes. In addition to A*, we also implement a hierarchical algorithm from [15] adapted to the multi-geography model that we call `Quad` given the quadratic nature of the algorithm. The `Quad` technique will first find paths from the source to all anchors in the source geography, all paths between source anchors and destination anchors in the anchor interconnection graph, and find the paths between destination anchors and the destination. The path is the best path combining the source, source anchor, and destination anchor and destination path segments. We implement this algorithm and apply it on our data set, and whenever needed we run A* to determine the path segments. Since basic A* works on a single level geography representation, we manually integrated several representative indoor and outdoor geographies over which A*, `Quad` and `MGRP` were executed. Note that in cases when the source and destination points are in different buildings we also integrate the outdoor network into a single geography for A*.

Figure 10 plots the speedup for the three techniques averaged across the different geographies. The speedup is computed as the running time of the techniques divided by the running time of A*, and thus the speedup of A* is 1 in this figure. Even for the limited number of geographies in this experiment, `MGRP` executes faster (speedup of 3-4) as compared to A*. This is because `MGRP` does not perform local search except in the source and destination geographies, while A* performs local search in all connecting geographies. `MGRP` also performs significantly better than `Quad` in our test case. `Quad` based techniques have been shown to work well with complete caching using materialization approaches [28]. This includes caching paths and path costs from every point in a geography to all anchors within geographies (PA Cache). In our problem setting, generating and storing such a fine-grained PA cache for all geographies is prohibitively expensive (due to a very large number of points in each geography) even for a moderately low number of geographies; hence, we do not consider the case of complete PA caching as a scalable option.

The efficiency of `MGRP` in the multi-geography scenario is due to the fact that local level planning is done only once (a single run of A*) for each source and destination side. An additional byproduct of this is that the performance of `MGRP` is less impacted by the number of anchors in the source and destination geographies; this is experimentally validated in [1] under different source and destination geographies. However, note that multiple A* calls cannot be avoided if geographies are complete black boxes and running `MGRP` at the local level is not possible.

Impact of Geography Pruning and Regionization. The next set of experiments evaluates the impact of two types of optimizations on `MGRP`: (i) across geographies through geography pruning using the GG-Cache, and (ii) within a geography by adding sub-regions using the regionization technique discussed earlier. The results of `MGRP` with these respective optimizations on a set of 5000 queries generated uniformly are demonstrated in Figure 11. We can see that GG-cache based pruning improves the performance of `MGRP` by eliminating unwanted explorations when exploring the anchor interconnection graph. While the improvement is limited for our current data set, we believe it would be more significant in other multi-geography topologies. We find that regionization improves the speed of `MGRP` significantly, by about 20% overall. While the extent of the benefit obtained by regionization can vary based on the geography set, and the structure of the geographies; our experiments indicate that this technique is useful across different grids in our data set. The combination of regionization and GG pruning reduces the running time even more - the rest of the experiments presented in this section include both of these optimization techniques in the `MGRP` implementation.

NG Cache Performance. These experiments address the role of the NG cache in path planning performance. We evaluate the two utility based strategies proposed in the previous section under varying cache sizes for the campus-wide multigeography network (with about 400 sub-geographies). For comparison, we implement two other simple caching strategies - a `Random` caching strategy and a most-frequently used (`MFU`) technique. The `Random` caching strategy selects anchor-geography pair for caching based on a uniform distribution. The most frequently used strategy estimates the number of times each anchor pair is requested, sorts the pairs in order of frequency of use, and caches the top k entries.

The first of our methods (`Util`) applies the utility-based technique under the assumption that every potential cached segment has an equal probability of being accessed. The second utility-based technique (`UtilMFU`) factors in access histories (via `MFU`) to estimate the frequency of requests for cached segments. In all of the following experiments the algorithm queries the cache by requesting cost from an anchor to all anchors in a given geography, hence reducing the number of disk block reads. All solutions cache anchor-geography pairs (i.e, an anchor to all anchors in a geography). We vary the cache size from 0 Mb to size of the full cache of 50 Mb.

We first study the overall performance of the algorithm by measuring the time taken and search area in terms of number of nodes visited for all four approaches. The graphs in Figure 12 and 13 demonstrate the performance of our strategies in comparison to the other solutions. Our utility based strategies exhibit superior performance both in terms of path planning time and search area. By storing path segments with both higher benefit in terms of cost saved, and number of other paths impacted, `Util` and `UtilMFU` skip more searches, while also avoiding extra cache accesses by increasing the probability of finding the destination anchors earlier. `UtilMFU` performs best both in terms of time and search area, while `Util` is very good for smaller cache sizes.

This is reinforced in Figures 14 and 15 which demonstrate how the different strategies perform in terms of cache accesses and cache hit rate. The first graph shows how many times the cache is accessed - we count the accesses for every anchor pair queried. With very small cache sizes, the

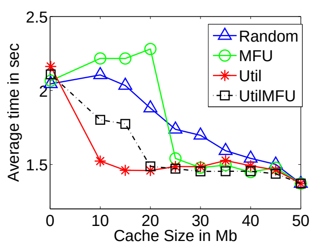

Figure 12: Average Time per Query.

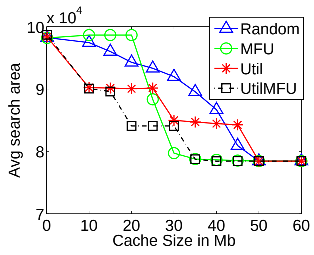

Figure 13: Average Search Area.

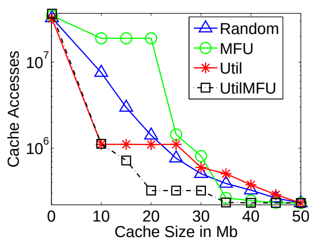

Figure 14: Cache Accesses.

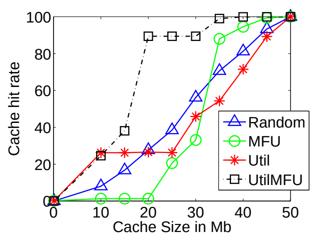

Figure 15: Cache Hit Rate.

number of accesses is high for all approaches. The number of accesses drop sharply for the Utility based approaches as cache sizes increase since useful data is available in the cache during the earlier stages of the `MGRP` algorithm and further accesses are avoided. This implies that utility based strategies provide benefit to the algorithm earlier. `UtilMFU`, which incorporates frequency of use information to `Util` shows improved cache access performance much faster hence performing well for all cache ranges. As is obvious, when the cache sizes are large, there is no significant difference in overall performance in the strategies. We expect to see greater improvement for the Util based approaches as the size of the outdoor network increases, since it permits farther "jumps" in exploration due to caching.

The IO performance (covered in detail in [1]) is similar. Our approaches again demonstrate better performance than the random and `MFU` approach, while `UtilMFU` has higher IO costs than basic utility approach. The lower number of cache accesses in general and the possibility of caching paths from anchors to smaller geographies results in smaller IO costs, specially for the first utility based approach.

7. CONCLUSION

In this paper we studied the problem of multi-geography route planning. We have proposed a multi-geography overlay structure that allows connecting heterogeneous geographies. We have presented a multi-geography planning algorithm that effectively uses cached data that utilizes two utility based caching strategies. We evaluated our solution on a real-world dataset that corresponds to a large university campus. Our experiments demonstrate a significant advantage of the proposed MGRP approach compared to the existing techniques.

8. REFERENCES

[1] V. Balasubramanian. Supporting scalable activity modeling in simulators. PhD Thesis.

[2] A. Bandera, C. Urdiales, and F. Sandoval. A hierarchical approach to grid-based and topological maps integration for autonomous indoor navigation. In *IEEE/RSJ IROS*, 2001.

[3] S. Behnke. Local multiresolution path planning. In *In Proc. of 7th RoboCup Int'l Symposium*, 2004.

[4] Y. Björnsson, M. Enzenberger, R. Holte, and J. Schaeffer. Fringe search: Beating a* at pathfinding on computer game maps. In *Proc. of the IEEE CIG*, 2005.

[5] Y. Björnsson and K. Halldorson. Improved heuristics for optimal path-finding on game maps. In *AIIDE*, 2006.

[6] A. Botea, M. Muller, and J. Schaeffer. Near optimal hierarchical path-finding. In *J. Game Development*, 2004.

[7] A. Car, H. Mehner, and G. Taylor. Experimenting with hierarchical wayfinding. Technical Report 011999, 1999.

[8] E. P. F. Chan and H. Lim. Optimization and evaluation of shortest path queries. 16(3):343–369, 2007.

[9] S. Chen, D. V. Kalashnikov, and S. Mehrotra. Adaptive graphical approach to entity resolution. In *Proc. of ACM IEEE Joint Conference on Digital Libraries (JCDL)*, 2007.

[10] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73, 1996.

[11] A. Fetterer and S. Shekhar. A performance analysis of hierarchical shortest path algorithms. In *Ninth IEEE Int'l Conf. on Tools with Artificial Intelligence*, 1997.

[12] A. V. Goldberg. Shortest path algorithms: Engineering aspects. In *ISAAC*, 2001.

[13] J. E. Guivant, E. M. Nebot, J. Nieto, and F. R. Masson. Navigation and mapping in large unstructured environments. *I. J. Robotic Res.*, 23(4-5), 2004.

[14] Y. Huang, N. Jing, and E. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *Proc. of CIKM*, 1996.

[15] N. Jing, Y. W. Huang, and E. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. In *TKDE*, 1998.

[16] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. In *J. of the ACM*, volume 24, 1977.

[17] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowl. Data Eng.*, 14(5), 2002.

[18] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (ACM TODS)*, 31(2):716–767, June 2006.

[19] D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM Data Mining (SDM)*, 2005.

[20] B.-Y. Ko, J.-B. Song, and S. Lee. Real-time building of a thinning-based topological map with metric features. In *IEEE/RSJ Conf. on Intel. Robots and Systs.*, 2004.

[21] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. VLDB, 2004.

[22] B. Lorenz, H. Ohlback, and E. Stoffel. A hybrid spatial model for representing indoor environments. In *W2GIS'06*.

[23] S. Pallottino and M. G. Scutella. Shortest path algorithms in transportation models: classical and innovative aspects. Technical Report TR-97-06, 1997.

[24] J.-S. Park, M. Penner, and V. K. Prasanna. Optimizing graph algorithms for improved cache performance. *IEEE Trans. Parallel Distrib. Syst.*, 15(9), 2004.

[25] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* 2003.

[26] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *ACM SIGMOD*, 2008.

[27] Seidel. On the all-pairs-shortest-path problem. In *STOC'92*.

[28] S. Shekhar, A. Fetterer, and Goyal. Materialization trade-offs in hierarchical shortest path algorithms. In *SSD'97*.

[29] S. Shekhar and H. Xiong. *Encyclopedia of GIS.* 2008.

[30] W.White, A.Demers, C.Koch, J.Gehrke, and Rajagopalan. Scaling games to epic proportions. In *ACM SIGMOD*, 2007.