Semantic Location Modeling for Location Navigation in Mobile Environment *

Haibo Hu Dik-Lun Lee
Department of Computer Science
The Hong Kong University of Science and Technology
{haibo, dlee}@cs.ust.hk

Abstract

Location-based applications require a well-formed representation of spatial knowledge. Current location models can be classified into symbolic or geometric models. The former attempts to represent logical entities and their semantics, but requires a large amount of manual effort for describing them. On the other hand, the latter represents the geometric coordinates but not the semantics.

In this paper, we present a semantic location model which preserves topology and distance semantics to support location navigation but at the same time facilitates programmatic model construction and maintenance. The model is based on a sound location theory. It is mainly composed of two hierarchies: a location hierarchy and an exit hierarchy, which can be derived from spatial maps, such as floor plans, without manual intervention. Through a series of model construction algorithms and a real example, we show that our model is simple but powerful enough to capture spatial connectivity and hierarchical relationship to support location-based applications. Furthermore, the location and exit hierarchies are easy to understand by human users.

1. Introduction

Mobile and ubiquitous computing deals with locations, sensors as well as objects of interest. Location-based services (LBSs), need a well-formed representation of location knowledge to cater for the need of location sensing, browsing, navigation and querying. Furthermore, it should be understandable and convenient to convey between location sensors, database servers (in which spatial queries are processed) and end users.

This paper presents a location model to meet these objectives. Moreover, it is the first semantic location model that supports programmatic model construction and maintenance. More specifically, all data in the model can be derived from a digital map without manual intervention. We believe this is the basic requirement for any location model to be applicable in a large scale. The model is built on top of the notion *location* and *exit*. The result of the modeling process is a *location hierarchy* and an *exit hierarchy*, which completely preserve the topology and distance semantics between locations.

In the paper, we build a sound theoretical foundation to define the model and propose a series of algorithms to construct the model from scratch. We illustrate the whole modeling process through a real example and show how cartographic information on locations and exits, typically found in floor plans and maps, can be modeled into two hierarchies, namely location and exit hierarchies, from which spatial semantics, such as the topology relationships and distance between locations, can be derived and stored. Furthermore, this model supports end-user queries and operations, such as shortest path queries and nearest neighbor search. Its sound theoretical background and fruitful semantics make itself a feasible and flexible model for location-based mobile and ubiquitous applications.

The rest of the paper is organized as follows: Section 2 reviews recent research progress on modeling locations. In Section 3, we formalize our model by a series of definitions, theorems and algorithms. A detailed modeling example is shown in Section 4. Finally in Section 5, we conclude this paper with the introduction of some promising applications and future work.

2. Related Work

Location modeling, as well as spatial knowledge representation, has attracted much attention in ubiquitous computing community and yielded much paper and project work in the literature.

Earlier location models attempt to differentiate the concept of "geometric" and "symbolic" models [8]. Symbolic locations and objects have their unique names (called "sym-

* Supported by Research Grants Council, Hong Kong SAR under grant HKUST6079/01E.

bols") to identify them among all entities. A partial ordering of these symbols forms a location hierarchy, such as a tree or a lattice. However, such models suffer from the huge cost of manual construction and maintenance. In geometric models, on the other hand, locations and objects are represented as points, areas or volumes within a reference coordinate system. Different sensors and applications can thus integrate their location information. In order to take the advantage of both models, a semi-symbolic model that decouples application representation (symbolic-based) from the sensor representation (geometric-based) of locations was developed in [8] by Leonhardt.

Narayanan formalized and extended Leonhardt's model in [9]. The geometric part, which is composed of "raw locations", resides at the lowest level. "Spatial realms" are the types of locations and "states" are instances of these "realms". Realms and states represent the symbolic view and are located on top of the "raw locations". The latest work in this category is conducted by Jiang and Steenkiste. In [7], they proposed a hybrid location model in which a location hierarchy represents the symbolic aspect of the space. In each location, a coordinate system is defined to capture the geometric attributes of locations that are contained in this location. In this way, a computable location identifier, called ALI, can be defined to designate an unambiguous location area. Binary operators, such as $distance(ali, ali), contains(ali, ali)$ are defined as primitives for upper level location services. However, in essence, this model has a particular emphasis on geometry: after ALI coordinate translation into a universal reference system, these operators are just common geometric operators; only the hierarchy is symbolic, however, it requires to be input from outside the model.

Attempts from industry include "SLoP" (Spatial Location Protocol) by the Internet Engineer Task Force (IETF) [1]. It's a purely descriptive model, which aims at unambiguously expressing location information in an interoperable form in the Internet. It restricts the representation format of the location information, such as the coordinates, identifier, timestamp and precision, in a well-formed XML document or plain text. Brumitt and Shafer from Microsoft adopted a symbolic model to represent a semantic space and developed the "EasyLiving" project using SQLServer 2000. They made use of 3 tables to represent "spaces" (i.e., locations), "containment" and "presence" (i.e., a location to geometric space mapping), respectively.

More recently, the research is diverted to specific modeling techniques for various augmented applications. Gessler et al. visioned other semantics, besides geographic ones, such as network, time, and user context, to be provided by the location model for sophisticated 3G applications [4]. In [2], Bauer et al. augmented the physical world to incorporate virtual objects and locations (e.g., a local-area network). [10, 12, 11] show another trend of location modeling. They modeled the movement of mobile objects (e.g., users), i.e., the change of object location.

3. The Semantic Location Model

3.1. Application Scope of The Model

Not a single model can cater for the needs of all location-aware ubiquitous applications, unless it's as generic as the ER-model. Our motivation is to develop a practical model for a mobile user to browse, navigate and search physical locations in indoor or other constrained environments. Such applications include: location-aware navigation guide in museums and large-scale office buildings; discovering the shortest path to the nearest facilities, etc. In this regard, we limit ourselves to represent the physical world in a visitor's context. More specifically, we investigate how physical locations are connected to each other through paths and what relationships they form.

3.2. Entities: Modeling the Physical Space

We adopt the ER-model as the underlying modeling infrastructure. In our model, **entities** are heterogeneously defined as they are in the real world, such as buildings, rooms and corridors. They can have arbitrary number and different sets of attributes, e.g., temperature for rooms, zipcode and population for street blocks. Nevertheless, among these attributes, two mandatory attributes are *id* and *location*. The id is required to uniquely identify the entity throughout the physical space. The "location" attribute, which designates the geographic information [1] of the entity, is defined as follows:

Definition 1 *The* **location** *of an entity is a bounded geographic area with one or more "exits", where an* **exit** *is a boundary point from and into which the entity can be departed and entered.*

Essentially, an exit denotes a critical place where the location state is changing from one entity to another. For example, a room exit is the place where the state is changing from "room" to "corridor".

In this first step of the modeling process, entities are identified from a digital map (a floor plan for an indoor or a district plot for an outdoor environment): their geographic and non-geographic attributes are stored through some graphical user interface (GUI). The identifying process is either manually or programmatically executed by some area-recognition algorithms in computer graphics.

1 As we concern the physical world, we require each entity to occupy a certain geographic area.

3.3. Semantics of Locations

In general, a location model should store not only the geometric information of the locations, but also, more importantly, the underlying semantics, i.e., meanings. These meanings are application-specific. In the context of location navigation, browse and search, we consider two fundamental semantics: the *topological relation* and *distance* between locations of entities.

Topological Semantics From a geometric point of view, the topological relations between two locations are: *disjoint*, *contains*, *within* and *overlap*, which are based on the 9-intersection model [13]. However, in the context of location navigation, the topological relations should reflect reachability semantics instead of intersection. Therefore, we base our new definitions of the topological relations on *path*:

Definition 2 *Exit x is* **directly reachable** *from exit y, denoted as $y \rightarrow x$, iff there exists a physical access (e.g., a passage for pedestrians or a road for automobiles) from y to x which involves no other exits* [2].

Definition 3 *A* **path** *from location a to b is a sequence of exits, $x_1, x_2, ..., x_s$ where $x_1 \in a$, $x_s \in b$ and $\forall i$, $x_i \rightarrow x_{i+1}$. As each exit belongs to a location, the path is also a sequence of locations* [3].

Definition 4 *A* **root exit** *is an exit of the entire modeling space, i.e., it locates at the boundary of the space. A* **root path** *of location p is a path from a root exit to an exit of p.*

Based on the definition of root path, we define the new topological relation *under* and *directly under* as follows:

Definition 5 *Location a is* **under** *location b, denoted as $a\backslash\backslash b$, iff any root path of a goes through b. a is* **directly under** *b, denoted as $a\backslash b$, iff $a\backslash\backslash b$ and no other location c satisfies $a\backslash\backslash c$ and $c\backslash\backslash b$.*

Intuitively, $a\backslash\backslash b$ means that if we need to arrive a, b **must** be first arrived at. Thus in some sense, b serves as a's ascendant.

Distance Semantics The semantics of distances differ in various applications: a location-based commercial advertising application may adopt the Euclidean distance; a finding nearest-restaurant LBS application may view distance as the road-network distance; an automatic robot controller, much differently, may see the distance as the energy consumption along the path. Therefore, we generalize the definition of distance semantics between locations as the accumulated distance along their path in terms of a valid metric, which satisfies:

bounded: $\forall$ location o_1, o_2, $distance(o_1, o_2) < \infty$;

zero-reflexibility: $\forall$ location o, $distance(o, o) = 0$;

transitive inequality: $\forall$ location a, b, c, $distance(a, b) + distance(b, c) \geq distance(a, c)$.

It's noteworthy that "symmetric" is not such a requirement, as in reality, $distance(a, b)$ may not be equal to $distance(b, a)$, e.g., two bus terminals connected through a one-way road.

3.4. Location Hierarchy: Modeling Topological Relations

The goal of semantic location modeling is to provide a meaningful representation of locations to the end-user and programs for browse, navigation and search purpose. As human can easily understand hierarchy in symbolic location models [8, 3], we model all the semantics (topological relations and distances) into hierarchy.

Any hierarchy requires a binary relation between elements, in the sequel, we show that the topological relation "directly under" is appropriate as being such a relation.

First, we have the following lemma on the properties of the "under" and "directly under" relation. Note that in order to form a hierarchy, a virtual root location designating the entire modeling space is always included.

Lemma 1 *$a\backslash\backslash b$, is a* **partial order** *relation, i.e., it has the following properties:* **reflexivity**, **antisymmetry** *and* **transitivity**.

Proof: 1) $\forall$ location a, the root path of a must pass through itself, i.e., $a\backslash\backslash a$;
2) if $a\backslash\backslash b$ and $b\backslash\backslash a$, any a's root path goes through b and any b's root path goes through a, the only way to avoid such a paradox is that $a = b$;
3) if $a\backslash\backslash b$ and $b\backslash\backslash c$, b is in a's root path, on the other hand, c is in b's root path, therefore, c is in a's root path, i.e., $a\backslash\backslash c$. □

Lemma 2 *The relation $a\backslash b$ has the properties of* **irreflexivity** *and* **uniqueness**, *i.e., there is one and only one b satisfies $a\backslash b$ for any a.*

Proof:
1) Irreflexivity: obviously location a is not directly under itself because location sequence of any path doesn't allow repetition.
2) Uniqueness: we first prove at most one b exists. By contradiction, if $a\backslash b_1$ and $a\backslash b_2$. There are at least two root paths, namely, $root \rightarrow\rightarrow b_1 \rightarrow\rightarrow b_2 \rightarrow\rightarrow a$ and $root \rightarrow\rightarrow$

2 If there exists a physical access from y to x, then x is *reachable* from y, denoted as $y \rightarrow\rightarrow x$. Since we assume that the physical space is a connected space, $y \rightarrow\rightarrow x$ is always true for any x, y.

3 In this paper, we only consider simple path, where no location appears more than once in the sequence. This requirement suits for most of the navigation or search applications where a path with a repetitive location in the sequence is considered as an inefficient path.

$b_2 \rightarrow\rightarrow b_1 \rightarrow\rightarrow a$, otherwise, either $a\backslash b_1$ or $a\backslash b_2$ doesn't hold. However, from the first path, b_2 can go directly to a without going through b_1. Thus in the second path, after reaching b_2, we can apply this knowledge to directly reach a and get round b_1. This shows that some a's root path doesn't go through b_1, which contradicts $a\backslash\backslash b_1$. Secondly, if a has no b satisfying $a\backslash b$, root is set as the unique b. □

Based on these properties, we finally show in the following proposition that the *directly under* relation forms a tree hierarchy for all the locations.

Proposition 1 *A graph of locations $G = (V, E)$, where V denotes all locations l_i and $< l_i, l_j > \in E$ iff $l_i\backslash l_j$, is a rooted tree.*

Proof:
1) We first prove that any other vertex v connects with the root. According to the uniqueness property in Lemma 2, we iteratively find p_1 that $v\backslash p_1$, p_2 that $p_1\backslash p_2$, etc. The p_i sequence finally reaches the root node, otherwise, it contradicts the uniqueness property because a certain location a doesn't have any b that satisfies $a\backslash b$.
2) Secondly, we prove there is no loop in G. If there were, the loop must be such sequence: $p_1\backslash p_2, p_2\backslash p_3, ..., p_{n-1}\backslash p_n$, $p_n\backslash p_1$. Otherwise, there is at least one location p_i in the loop which has 2 or more than 2 p satisfying $p_i\backslash p$, which contradicts the uniqueness property. However, even such sequence is impossible, because $p_1\backslash p_2$, ..., $p_{n-1}\backslash p_n$ implies $p_1\backslash\backslash p_n$ (by the transitivity property in Lemma 1), combining the last one $p_n\backslash p_1$, we derive $p_n = p_1$ by the antisymmetry property in Lemma 1. This contradicts the assumption that p_i are unique locations in the loop.
From the above two aspects, G is a free tree. Since root is the only location that doesn't have any p satisfying $root\backslash p$, it is the root node of G, i.e., G is a rooted tree. □

Proposition 1 defines the location hierarchy in terms of the "directly under" relation. By its definition, the hierarchy preserves topological semantics of reachability. From a root point of view, a location can be reached only if its ascendants are reached first. Nevertheless, in practise, it's not straightforward to implement the "under" and "directly under" relations by their definitions. In the next subsection, we introduce the **exit hierarchy**, which captures the distance semantics and also provides a practical and efficient way of computing the location hierarchy.

3.5. Exit Hierarchy: Modeling the Semantic Distances

Semantic distances are required for many location-based services, such as nearest service discovery and shortest path finding. The distance space is based on a predefined metric. As described in Section 3.3, any distance metric can be applied as long as it has the bounded, zero-reflexibility and transitive inequality properties.

The semantic distance is based on the path between two exits:

Definition 6 *The "shortest distance" between location a's exit x and b's exit y, denoted as $dist(x, y)$, is the shortest path distance from x to y in the certain distance metric.*

From the above definition, if x and y are directly reachable, their distance value is a "primitive", i.e., it cannot be derived by this definition. Such primitive distances are required to be specified from outside the model, according to the chosen metric. The model's responsibility is to store these primitives and derive other combinational distances according to Definition 6. Formally, we define *primitive distances* as follows:

Definition 7 *The distance between exit x and y is a* **primitive distance** *iff $y \rightarrow x$.* [4]

Intuitively, to preserve semantic distance we can model all the exits as vertices of a graph and primitive distances as the edge weights, however, it's not a good solution. First, the computation of shortest distance in a huge graph [5] is rather costly: existing single-pair-shortest-path algorithms are all memory based with time complexity $O(n^2)$. Secondly, a flat graph doesn't capture the essence of the exits. Imagine two exits: a building's exit and a room's exit. They have definitely different significance in the involvement of distance semantics: a building exit is probably a mandatory exit for any path from outside the building to its inside while a room exit only appears in a path to the room. A flat graph cannot convey such information. On the contrary, an "exit hierarchy" in which a building exit locates at a higher level of the hierarchy than a room exit, more clearly delivers such difference. Thirdly, as mentioned in Section 3.4, we need a computable definition of "under" relation which is equivalent to Definition 5 so that the location hierarchy can be programmatically constructed. As we will see later in this subsection, the location hierarchy can be derived from the exit hierarchy.

Informally, the exit hierarchy is defined based on the "directly reachable" relation: all root exits are on top of the hierarchy (level 0), any exit that is directly reachable from level 0 is in level 1, and so on. Figure 1(a) illustrates such an exit hierarchy. Level 0 vertices may denote building exits, while level 1 and 2 vertices may denote floor and room exits, respectively. It's formal definition is as follows:

4 To clarify subsequent definitions and theorems, in the sequel, we assume path is always bidirectional, i.e., "directly reachable" is a symmetric relation.

5 In reality, it's very common that a modeling space has tens of thousands of such exits, e.g., a university campus or a urban district.

Definition 8 *An* **exit hierarchy** $G = (V, E)$ *is a levelled undirected graph, where V denotes all exits, edge* $\langle v_i, v_j \rangle \in E$ *iff* $v_i \rightarrow v_j$. *The level of each vertex* v *is defined as:*
1) 0, if v *is a root exit; or*
2) $\min_u u.level + 1$, $u \in V$ *and* $< v, u > \in E$.

To build the exit hierarchy from scratch, first of all, we construct the plain graph G which contains all exits as vertices and their directly reachable relations as edges. Then breadth-first traverse G from each of the root exits. Each traversal will associate every vertex v with a level value in the layer of which it's traversed in BFS. $v.level$ in Definition 8 is just the minimum level value among all traversals for v. The detailed pseudo-code is omitted here due to space limitation.

The exit hierarchy has the nice property:

Lemma 3 *If edge* $< u, v > \in G$, *the exit hierarchy, then* $|u.level - v.level| \leq 1$.

Corollary 1 *An location a's all exits differ at most 1 in their "level" values.*

The exit hierarchy is a full representation of distance semantics, however almost as complicated as a graph to perform shortest path search. In the sequel, we propose the "compact exit hierarchy" (CEH) which is derived from an exit hierarchy but has a tree skeleton. The basic idea is to cluster vertices of the same level in the hierarchy. Before we define the CEH, it's necessary to define the criterion of clustering.

Definition 9 *Two exits* x *and* y *such that* $x.level = y.level$, *are* **downward reachable** *iff there is a path from* x *to* y *whose exits'* $level$ *values are all higher than* $x.level$.

Lemma 4 *The* **downward reachable** *relation is a equivalence relation, i.e., it satisfies reflexive, symmetric and transitive.*

Essentially, "downward reachable" indicates that the two exits can reach each other without passing through higher level exits. Lemma 4 shows that this binary relation forms a partition of all exits. Therefore it's set as the clustering criterion for CEH.

Definition 10 *A* **compact exit hierarchy** *(CEH) is a graph* G^c *which is derived from the exit hierarchy* G *s.t.:*
1) $v \in G^c.V$ *iff* v *is an equivalence class of* $G.V$ *in terms of "downward reachable";*
2) $\langle u, v \rangle \in G^c.E$ *iff* $\exists x \in u, y \in v$, *s.t.,* $\langle x, y \rangle \in G.E$.

Figure 1(b) illustrates the corresponding CEH of the exit hierarchy in Figure 1(a). A vertex in G^c is actually a "supernode" which contains several vertices of the same level in G. Such supernode still keeps the "level" attribute. Intuitively, a CEH seems to have a tree-like shape. The following proposition confirms this observation.

Proposition 2 *The CEH* G^c *is a forest.*

Proof: Equivalently, we prove that any vertex $v \in G^c$ (except level 0 vertices) with level lev has one and only one connected level $lev - 1$ vertex, which is denoted as v's parent. Firstly, v must have at least one such parent. Otherwise, no root path exists for any exits$\in v$, which contradicts that any exits can be reached from root exits. Secondly, if v has more than one such parent, which are denoted as $u_1, u_2, ...$ $\forall x \in u_1$ and $\forall y \in u_2$, by definition, $x.level = y.level$, and $\exists s, t \in v$ such that, $x \rightarrow s$ and $t \rightarrow y$. Since s and t are downward reachable, by definition, x and y should also be downward reachable, because there is a path $x \rightarrow s \rightarrow\rightarrow t \rightarrow y$ all of whose exit levels are greater than $x.level$. However, this contradicts the fact that $x \in u_1$ and $y \in u_2$, i.e., they are in two different partitions. From the above two factors, G^c is a forest. □

Corollary 2 *The CEH* G^c *is a tree iff all root exits are downward reachable.*

We propose Algorithm 1 to extract the CEH from an exit hierarchy in a bottom-up fashion. Basically, it tries to identify the equivalence partitions for vertices in the same level of G.

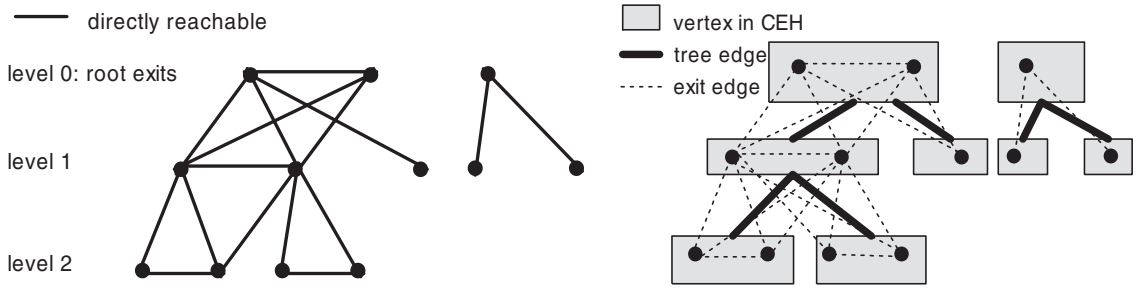
Algorithm 1 Extract CEH From Exit Hierarchy

Input: G: the exit hierarchy
$maxLevel$: $\max_{v \in G.V} v.level$
Output: G^c: CEH
Procedure:

1: create an empty graph M;
2: $M.V$ = vertices of $maxLevel$ in G;
3: $M.E$ = edges between vertices of $maxLevel$ in G;
4: find connected components $u_1, u_2, ..., u_n$ of M;
5: insert $u_1, u_2, ..., u_n$ as vertices into G^c;
6: **for** $level$ from $maxLevel - 1$ down to 0 **do**
7: create an empty graph M;
8: $M.V$ = vertices of $level$ in $G \cup$ vertices of $level + 1$ in G^c;
9: $M.E$ = edges between vertices of $level$ in $G \cup$ edges between u, a $level$ vertex in G and p, a $level + 1$ vertex in G^c if $\exists v \in p$, $\langle u, v \rangle \in G.E$;
10: find connected components $u_1, u_2, ...$ of M;
11: remove vertices of $level$ in G from $u_1, u_2, ...$;
12: insert $u_1, u_2, ...$ as vertices into G^c;

A CEH captures the same exit hierarchy in a simplified way, so it's suitable for visualization to end-users. Furthermore, as it's essentially a tree structure, shortest paths can be more efficiently found. To achieve this, the *exit edge* [6]

6 There are two types of edges in G^c, as illustrated in Figure 1(b),



(a) Exit Hierarchy (b) Compact Exit Hierarchy

Figure 1. Sketches of Exit Hierarchy

weights in the CEH G^c requires to be defined, based on their edge weights in G.

Definition 11 *$\forall x, y \in G.V$, , weight(x,y) and path(x,y) can be defined in G^c iff:*
1) $x, y \in v, v \in G^c.V$; or
2) $x \in u, y \in v$, and u is the parent of v in G^c or vice versa. where weight(x,y) = shortest distance from x to y in G, path(x,y) = shortest path from x to y in G.

In other words, exit edges are defined for those exits that are in the same equivalence partition or in two partitions that have parent-child relationship in G^c. The edge weights are derived from G by finding the shortest path distances between these exits. Such a process is not costly, because all these exits are located in the vicinity of each other.

Given a CEH G^c with all edge weights set as Definition 11, finding the shortest path from any exit x to exit y in G is much simplified, due to the following proposition.

Proposition 3 *For any two exits $x, y \in G$, assume $x \in u, y \in v$, where $u, v \in G^c.V$. $u, S_1, S_2, ..., S_n, v$ is the tree path from u to v in G^c. Then the shortest path from x to y in G can be represented by $path(x, s_1) \cup path(s_1, s_2) \cup ... \cup path(s_n, y)$, where $s_i \in S_i$, $\forall 1 \leq i \leq n$. And the shortest distance is $weight(x, s_1) + weight(s_n, y) + \sum_{i=1}^{n-1} weight(s_i, s_{i+1})$.*

Proof:

1) Firstly, the proof needs the following lemma:

Lemma 5 *For any S_i, there is at least one $s_i \in S_i$ for any path between x and y in G.*

Proof Sketch: Otherwise, there must be edges between vertices in G whose levels differ more than 1, which is prohibited by Lemma 3. □

This lemma says the tree path in G^c forms the skeleton of the actual path in G.

namely, tree edges (bold lines) which link two vertices in G^c and exit edges (dotted lines) which link two exits.

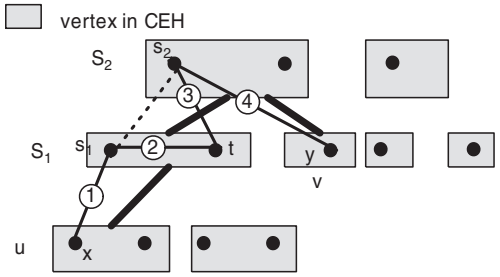


Figure 2. Find the shortest path in G in the help of G^c

2) Secondly, no vertex in G that doesn't belong to $u, S_1, S_2, ..., S_n$ or v should be excluded in the shortest path, otherwise removing it can decrease the path distance.

3) Thirdly, only one s_i is needed in G for S_i in G^c. Otherwise, suppose $s_i, s_i' \in S_i$, the path is $... \cup path(s_{i-1}, s_i) \cup path(s_i, s_i') \cup path(s_i', s_{i+1}) \cup ....$ However, since $weight(s_{i-1}, s_i) + weight(s_i, s_i') \leq weight(s_{i-1}, s_i')$ (triangular inequality), removing s_i in the path can decrease the path distance.

The proposition is proved by the above 3 arguments. □.

Proposition 3 essentially tells us how to find the shortest path between exit x and y efficiently, given the CEH G^c. Figure 2 illustrates such a process. First, the two vertices u, v in G^c that contain x, y respectively are identified. Secondly, the tree path $u, S_1, S_2, ..., S_n, v$ is located in G^c (the bold lines in Figure 2). In the third step, various combinations of $s_i \in S_i$ for all i are considered and the shortest path is retrieved. For this example, the shortest path is $path(x, s_1) \cup path(s_1, s_2) \cup path(s_2, y)$. Since $path(x, s_1) = x \rightarrow s_1$, $path(s_1, s_2) = s_1 \rightarrow t \rightarrow s_2$ and $path(s_2, y) = s_2 \rightarrow y$, the actual shortest path is $x \rightarrow s_1 \rightarrow t \rightarrow s_2 \rightarrow y$.

To further enhance the performance in the third step, a dynamic programming approach can be applied. Let $d(x, y)$

denote the shortest distance from x to y in G, $w(a, b)$ denote the exit edge weight from a to b in G^c, and $s_{i,j}$ denote the jth exit in S_i of G^c. We have the following recursive formula for the DP algorithm:

$$d(x, y) = \min_j d(x, s_{n,j}) + w(s_{n,j}, y) \quad (1)$$

$$d(x, s_{i,j}) = \min_k [d(x, s_{i-1,k}) + w(s_{i-1,k}, s_{i,j})] \quad (2)$$

$$d(x, s_{1,j}) = w(x, s_{1,j}) \quad (3)$$

The complete pseudo-code for the shortest path search is illustrated in Algorithm 2.

Algorithm 2 Find the shortest path in G

Input: G, G^c: the exit hierarchy and its CEH
x, y: the source and destination exit
Procedure:

1: identify u, v as the two vertices in G^c containing x, y;
2: locate the tree path $u, S_1, S_2, ..., S_n, v$ in G^c;
3: build dynamic programming algorithm according to Equation 1,2,3 to find the shortest path sequence $s_1, s_2, ..., s_n$,;
4: the exact shortest path from x to y is $path(x, s_1) \cup path(s_1, s_2) \cup ... \cup path(s_n, y)$

Implementing "Under" Relation Given the exit hierarchy G and CEH G^c, the "under" relation between location a and b can be efficiently implemented. Algorithm 3 shows the pseudo-code. Based on Lemma 5, the idea behind is to first locate the root tree path of a in G^c, and then check whether deleting b's exits and associated edges in G will disconnect this tree path in G^c. If so, a is under b; otherwise, it is not.

The algorithm requires to locate b's exits in a tree path in G^c. Corollary 1 guarantees that they appear in at most two consecutive vertices in the tree path, i.e., j is at most $i + 1$. Therefore, the intermediate graph M in this algorithm is fairly small and connectivity is efficient to check. In this way, we argue that the exit hierarchy and its CEH provides an efficient implementation of the "under" relation for the location hierarchy.

3.6. The Geometric-Symbolic Model

Similar to the one proposed in [8], our location model is a combination of geometric and symbolic one. The two hierarchies, location hierarchy and exit hierarchy are the symbolic views, while the geometric information is maintained as the attributes of locations and exits. Nevertheless, the major difference between ours and former models lies in that in our model, the symbolic hierarchies are constructed in terms of the geometric attributes. Location semantics is retained automatically without manual inference, through the well definition of the topological relations and semantic distance. Once the model is built, high-level location navigation, browsing and search are conducted without referring to the geometric attributes underneath.

Algorithm 3 Check the Under Relation between two locations

Input: G, G^c: the exit hierarchy and its CEH G^c
a, b: location a and b
Output: $under$: boolean value denoting if a is under b
Procedure:

1: $under$ = true;
2: **for** each exit x in a **do**
3: identify $u \in G^c.V$ s.t., $x \in u$;
4: locate the root tree path $root, s_1, s_2, ..., s_n, u$ in G^c;
5: locate b's exits in this tree path;
6: **if** they don't exist in this tree path **then**
7: $under$ = false;
8: **else if** they exist in $s_i, ..., s_j$ of the tree path **then**
9: build empty graph M;
10: $M.V$ are exits in $s_i, ..., s_j$ of G^c;
11: $M.E$ are edges between $M.V$ in G;
12: insert two special vertices "i-1","j+1" representing s_{i-1}, s_{j+1} respectively;
13: insert edges between "i-1" and those vertices in s_i;
14: insert edges between "j+1" and those vertices in s_j;
15: remove vertices that are b's exits and associated edges;
16: **if** "i-1" and "j+1" are still connected in M **then**
17: $under$ = false;
18: return $under$;

4. A Complete Example: Modeling an Indoor Office Area

In this section, we illustrate the whole modeling process for a specific physical area. It is a portion of the Computer Science laboratory area on 4th floor of the academic building at Hong Kong University of Science and Technology. Figure 3 depicts the floor plan of the modeling area.

4.1. Step 1: Identifying Locations and Exits

First of all, we need to identify all entities, i.e., locations, together with their associated exits, that are of interest to the application. This step can either be processed manually or programmatically. A program that applies computer graphics techniques can recognize rooms, exits and the corridors in Figure 3. Further criteria for specific applications, e.g., corridors should be at least 30 meters long, can be accompanied by customizing the program.

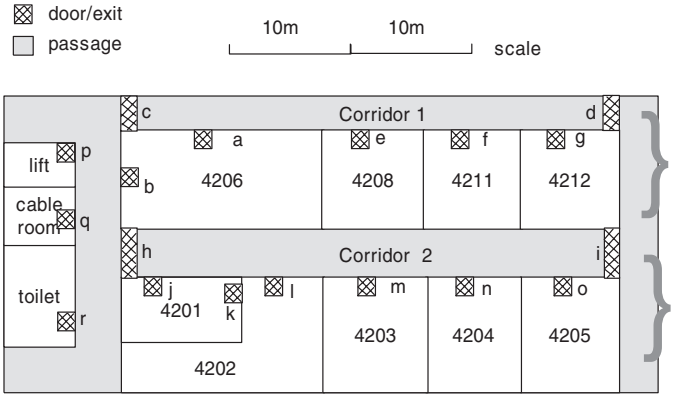


Figure 3. Floor Plan of CS Laboratory Area

In this example, the following geographic areas are identified as entities: Room 42xx (totally 9 rooms), lift, cable room, toilet and corridor 1 and 2. And all the doors in Figure 3 are identified as exits.

4.2. Step 2: Construct Exit Hierarchy

In the spirit of Definition 8 and Algorithm 1, we build the exit hierarchy as illustrated in Figure 4. Exit p is chosen as the root exit, because it represents a lift exit, which is the only way of leaving the modeling space. In Figure 4(a), to enhance legibility, edges between c, b, r, q, h, between f, e, a, g, d and between j, l, m, n, o, i are not depicted. For the same reason, exit edges in the same vertex in G^c are also omitted, i.e., only tree edges are depicted in Figure 4(b). Exit edges are only required to represent distance semantics, e.g., shortest paths, which is described in step 4.

4.3. Step 3: Construct Location Hierarchy

Given the exit hierarchy in step 2, we can programmatically build the location hierarchy by its definition in Proposition 1 and Algorithm 3.

The resultant location hierarchy is illustrated in Figure 5(a). As the lift contains the root exit, it's the $root$ of the entire modeling space. From the figure, all laboratory rooms except 4206 have been categorized into two groups: those under Corridor 1 and 2. Room 4206, on the other hand, although its geographic area is adjacent to the other laboratory rooms, it's a location directly under $root$, because one of its two exits b is a level 1 exit in Figure 4(a). In reality, this laboratory belongs to the department of Chemistry, which confirms that our model captures more accurate topological semantics than simple geometric models.

4.4. Step 4: Retrieving Primitive Distances

In this step, primitive semantic distances, i.e., the distance between two "directly reachable" exits need to be retrieved and stored in our model for further distance-related queries, e.g., shortest path or spatial navigation. Figure 5(b) depicts such primitive distances. In real implementations, these distances are the edge weights in the exit hierarchy G. The exit edge weights in G^c are directly derived from them by Definition 11.

4.5. Incorporate Virtual Locations

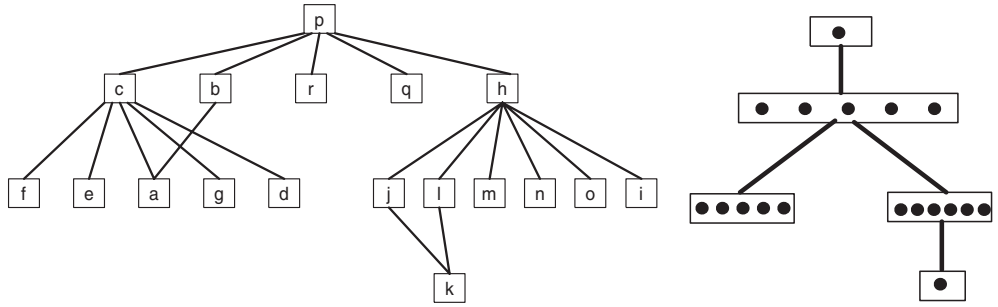
In many LBS applications, virtual entities, which are the abstraction of low-level physical entities, are of interest. Their locations are often the aggregation of low-level physical entities' geographic areas. For example, in Figure 3, Zone 1 and Zone 2 are such locations. Zone 1 contains Rm4206, 4208, 4211, 4212 and corridor 1, while Zone 2 contains Rm4201, 4202, 4203, 4204, 4205 and corridor 2. We see in below, how these two virtual locations can be incorporated into our existing location hierarchy.

By definition, Zone 1 have b, c, d as its exits. But since these exits belong to the virtual location, they are classified as virtual exits and denoted distinguishingly as b', c', d'. We imagine these exits locating almost at the same place as the corresponding physical exits b, c, d, but a bit more exterior. Therefore, any root path to b must go through b', so are c (with c') and d (with d'). In this sense, by Definition 5, Zone 1 becomes the parent of Rm4206 and corridor 1, which is further the parent of Rm4208, 4211 and 4212. This integration coincides with human's cognition: in the location hierarchy, a virtual location should be the ascendant of all physical locations it contains. In this way, we argue that our semantic location model can seamless support virtual locations, which cater for the needs of location abstraction and multi-resolution in many LBS applications.

5. Conclusion and Future Work

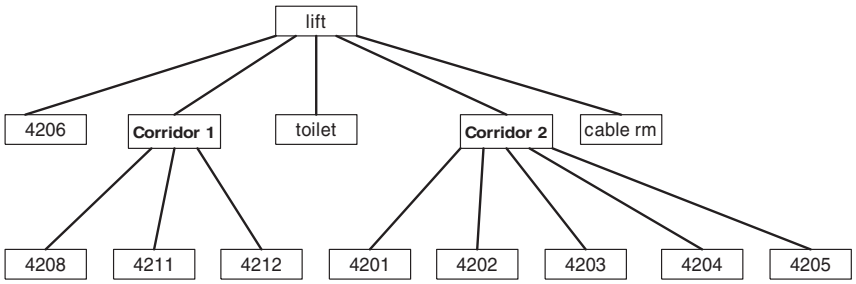
In this paper, we proposed a semantic location model for location-based applications, especially those in the realm of location navigation and browsing. Both topological and distance semantics are formally defined, derived and stored in this model. The modeling result is mainly composed of two hierarchies: the location hierarchy and exit hierarchy. We founded a sound theoretical background and developed a series of algorithms to generate these hierarchies from scratch, given a digital map representation of the modeling space.

We consider our model applicable to large scale location-based applications. To name a few of them, it's desirable to introduce *semantic nearest neighbor search and location-aware navigation*. The former is to find the nearest facil-



(a) The Exit Hierarchy G (b) CEH G^c

Figure 4. Exit Hierarchy of Figure 3



(a) The Location Hierarchy

From Exit	To	Distance(m)
... ...	... ...	... ...
p	c	6
p	h	10
p	b	5
... ...	... ...	...
c	h	12
b	h	5
d	i	12
... ...	... ...	...
a	e	11
... ...	... ...	...

(b) Primitive Distances (Edge Weights in G and G^c

Figure 5. Location Hierarchy and Primitive Distances

ity or service from a mobile user according to the actual physical path distance, instead of Euclidean distance. In our previous work [6], we proposed an index-based efficient searching algorithm to execute such semantic nearest neighbor queries, based on the exit graph G and its CEH G^c. The latter is to improve the experience of location visualization for hand-held device users. Due to their limited display screens, it's desirable to visualize only a necessary set of locations (and exits) in user's context. Our location and exit hierarchy are helpful to determine such a set, for example, a set which comprises the current location, its siblings, children and parent in the location hierarchy. Such a concise display of location information renders the user an unambiguous and clear understanding of his current location context.

To fully represent and reason spatial knowledge, a semantic location model is just the beginning. As our future work, we are to develop a complete set of querying, manipulating and reasoning operations and expressions which address both the geometric and semantic properties of a spatial entity. Hopefully, this will finally lead to a general language which defines, manipulates, queries and reasons locations.

6. Acknowledgment

This work was supported by Research Grants Council, Hong Kong SAR under grant HKUST6079/01E. And we also owe our thanks to the anonymous reviewers for their insightful comments.

References

[1] S. Activity. Spatial location bof (spatial) of ietf. http://www-nrc.nokia.com/ietf-spatial/, 2000.

[2] M. Bauer, C. Becker, and K. Rothermel. Location models from the perspective of context-aware applications and mobile ad hoc networks. In *Proceedings of Location Modeling Workshop at Ubicomp*, 2001.

[3] B. Brumitt and S. Shafer. Topological world modeling using semantic spaces. In *Proceedings of Location Modeling Workshop at Ubicomp*, 2001.

[4] S. Gessler and K. Jesse. Advanced location modeling to enable sophisticated lbs provisioning in 3g networks. In *Proceedings of Location Modeling Workshop at Ubicomp*, 2001.

[5] H. Hu and D. L. Lee. Semantic location modeling for location-aware applications in mobile and ubiquitous computing. Technical report, Hong Kong University of Science and Technology, June 2003.

[6] H. Hu, J. Xu, and D. L. Lee. Semantic nearest neighbor search. Technical report, Hong Kong University of Science and Technology, February 2003.

[7] C. Jiang and P. Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing. In *UniComp 2002*, 2002.

[8] U. Leonhardt. Supporting location-awareness in open distributed systems. *PhD thesis, Department of Computing, Imperial College London*, 1998.

[9] A. K. Narayanan. Realms and states: a framework for location aware mobile computing. In *Proceedings of the first international workshop on Mobile commerce*, 2001.

[10] T. O'Connell, P. Jensen, A. Dey, and G. Abowd. Location in the aware home. In *Proceedings of Location Modeling Workshop at Ubicomp*, 2001.

[11] T. Pederson. Object location modelling in office environments - first steps. In *Proceedings of Location Modeling Workshop at Ubicomp*, 2001.

[12] C. Schlieder, T. Vogele, and A. Werner. Location modeling for intentional behaviour in spatial partonomies. In *Proceedings of Location Modeling Workshop at Ubicomp*, 2001.

[13] S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, and L. Chang-tien. Spatial databases - accomplishments and research needs. *TKDE*, 1999.