Pervasive and Mobile Computing 15 (2014) 26–47

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

TIP-tree: A spatial index for traversing locations in context-aware mobile access to digital libraries

Wendy Osborn [a,*], Annika Hinze [b]

[a] Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, Canada
[b] Department of Computer Science, University of Waikato, Hamilton, New Zealand

ARTICLE INFO

Article history:
Available online 8 December 2013

Keywords:
Spatial indexing
Context-aware mobile systems
Location-based access
Digital libraries

ABSTRACT

We present a spatial index called the TIP-tree, for efficient, uniform, location-based access to digital library collections that are external sources to a context-aware mobile system. Using a tourist information system, we also utilize the TIP-tree to manage the context of location. We show how access to resources from within and outside of the tourist information system can be carried out seamlessly. We also propose a strategy to show how the TIP-tree can be navigated within the structure itself to continually provide information to the user. An empirical evaluation of the navigation strategy with both synthetic and real data shows that navigation is very efficient, with a constant number of node accesses regardless of data and trajectory size, and at least an 80% improvement in node accesses over searching with other spatial indexing approaches.

1. Introduction

A mobile tourist information system provides specific information to a traveler based on several factors, including their current location, interests and past viewing history. In addition to offering information that is managed within the system, information that is available from external repositories should also be offered to the user in a location-based manner. In particular, digital libraries contain a wealth of information for the user to access. The overall goal of our system is therefore the following: while traveling with a mobile device, the user is continually presented with information from not only the mobile information system, but also from external digital libraries, based on the above mentioned factors, or contexts.

To accomplish this, context information must be kept by the tourist information system for each user. A user profile typically keeps track of context information that is more static in nature and changes infrequently (e.g., topics of interest, such as architecture and parks, and past viewing history). A user's current location, however, is obtained continuously from their mobile device, and changes frequently as she/he moves around. The information delivered to a user is thus determined by both the user's profile and the user's location: the system selects only information that refers to items in close proximity to the user (location filter) and that matches the user's interests (profile filter).

Mobile location-based systems can be used in two scenarios: following pre-defined routes or in an open setting in which the user freely chooses their route. For systems with pre-defined routes, such as walking tracks and set tours, it would be possible to also store all location co-ordinates and their linked information within the user profile. As a user is navigating a track, the system would then simply traverse the list of locations according to the user's current location, and offer information filtered by user interest.

* Corresponding author. Tel.: +1 403 329 2294; fax: +1 403 317 2882.
E-mail addresses: wendy.osborn@uleth.ca (W. Osborn), hinze@cs.waikato.ac.nz (A. Hinze).

1574-1192/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.pmcj.2013.12.002

However, in an open system where a user is not restricted to pre-defined routes of linked locations, two issues arise around the location-based storage of information. Firstly, there exist many municipalities in the world, each of which has its own co-ordinates. For example, World Gazetteer[1] contains over 300,000 co-ordinates. Secondly, in addition to existing municipalities, a user may want information about a location that is not specific to a municipality (e.g., an iceberg that has migrated to a location off the coast of New Zealand).

Given both sources of co-ordinates, there is the potential of having to store close to a million co-ordinates in a user profile, which is completely undesirable. Therefore, information must be organized by its location in such a way that efficient information retrieval based on the user's current location is possible. Due to the mobility of our system's users, the retrieval strategies typically employed by large-scale Geo-information systems for stationary users are not suitable. Instead we require a navigation strategy that can deal with continuous queries by mobile users.

Paper objectives and contributions. This paper has two primary objectives. The first objective is to provide a framework for efficient and uniform access from a context-aware mobile information system to resources contained externally in digital libraries, as well as resources within the mobile information system itself. This is accomplished by utilizing a spatial index, called the TIP-tree. The second objective is to propose an algorithm for efficiently navigating the spatial index to continually provide the mobile user with information whenever it exists. An empirical evaluation of the navigation algorithm versus traditional spatial searching shows that the navigation algorithm is efficient and outperforms repeated spatial searching. The following list summarizes the key contributions of this paper[2]:

- Incorporating the TIP-tree into a tourist information system for managing access to both internal and external resources. We organize known sources of information in a spatial index by their GPS co-ordinates' location instead of their place names.
- Access to any digital library collection. Our system has been shown to provide access to digital library collections managed by the three most prominent Dl software systems: Greenstone [2], DSpace [3] and Fedora [4].
- One-pass access to relevant collections. Because GPS co-ordinates can be obtained directly from a mobile device, and the information sources are organized by their GPS co-ordinates, we can directly access both internal and external sources directly by using the spatial index.
- No requirement of a "geographically-aware digital library". Previously, a collection needed to be preprocessed to identify place names (i.e., locations) in its documents. This is no longer a strict requirement. Any resource (internal or external) can now be added by specifying its GPS location, which can be obtained by from a World Gazetteer. However, place name mark-up provides an additional advantage. It allows for place names to be selected (as a hyper-link) and further index accesses to take place, although an extra translation step to obtain GPS co-ordinates is required. But it will not be required in all situations when the index is accessed.
- Handling collections that reference multiple locations. If a digital library collection contains documents that mention multiple locations, the collection can be added to the index multiple times, once per every location that is mentioned.
- Navigating the index for continuous information dissemination to the user. We base the TIP-tree on an efficient spatial index, which could be searched from the beginning every time a user moves along with their mobile device. However, this is unnecessary. The spatial index can also be efficiently navigated to continually provide information to the user, thus avoiding extra searching.

Note that although we present some screen-shots to provide context for the paper, the visualization of any retrieved resources is outside the scope of the work described in this paper.

Road-map. The remainder of this paper is structured as follows. Section 2 provides a brief summary on TIP (Tourist Information Provider), digital libraries and the mqr-tree, all of which are required for our work. Section 3 summarizes related work and its limitations. In addition, this section will present our contributions. Section 4 presents our first user scenario that demonstrates uniform access to both internal and external resources. Section 5 presents the required modifications to the TIP architecture, while Section 6 presents the TIP-tree, its organization of internal and external resources, and the algorithm for continuous navigation of the index. Section 7 presents our second scenario that demonstrates how the TIP-tree is continuously navigated in order to provide updated information to the user. Section 8 presents our empirical evaluation of the TIP-tree. Finally, Section 9 concludes the paper and discusses outstanding issues.

2. Background

This section presents a brief introduction on digital libraries, the tourist information system TIP, and the mqr-tree, all of which are required for this work.

[1] http://www.world-gazetteer.com/.

[2] This paper extends our previous work [1], which was expanded to include the following material: (1) Increased detail about the mqr-tree implementation, including structure, insertion, and search. (2) A section on R-tree and its implementation to support our comparison with the TIP-tree. (3) A section on insertion and basic point search for the TIP-tree. (4) Addition of new TIP-tree evaluations using real data and a very large synthetic data set. (5) Addition of comparison of TIP-tree with R-tree (sorted and random data). (6) A new section on push and pull communication as used for the TIP-tree. (7) Increased detail about the TIP architecture. Finally, we have updated and expanded the literature review of spatial indexing methods.

Fig. 1. TIP Interface.

2.1. Digital libraries

A digital library [2] is a collection of documents that are digitized, individually cataloged, and organized for computerized dissemination and access. Digital library software systems provide the means for creating, searching and browsing collections of documents of various types, such as PDF, HTML and Microsoft Word. Some digital libraries can be searched and browsed using different meta-data items, such as title, source, etc. Also, these digital libraries can also have their "look and feel" customized. Many digital library software systems exist, the most prominent ones are Greenstone [2], DSpace [3] and Fedora [4]. While all three vary in their functionality, all three allow their collections to published and accessed on the World Wide Web. This feature allows collections from these systems to be incorporated into and accessed by location from our spatial index. We utilize Greenstone and DSpace in our running examples. However, any web-accessible digital library collection can be incorporated into this system for location-based access.

2.2. TIP — tourist information provider

TIP is a mobile tourist information system that provides the user with information based on their contexts such as their current location, interest in specific groups of sights and topics, and travel history [5–7]. Examples of groups of sights include beaches, green spaces and buildings, while examples of topics include history and architecture. A user's travel history can include any locations or sights that a user has visited. The information about a user's interests in sight groups and topics, as well as their history information is maintained in a user profile. Fig. 1 depicts the TIP interface in a mobile emulator. The current user's profile contains the following: *groups* = *buildings*, *parks* and *topics* = *architecture*, *history*. Given their current location at the University of Waikato, the university is presented as a building near the user's current location.

The TIP 3 architecture (see Fig. 2, for details see [7]) is organized by separating the core system logic (event-based communication, location engine, geo-coding and filter engine) and TIP data (user profile DB and sight data) from the various services that interact with it (e.g. internal services and external services). A change in the user's location triggers the core system to provide information about the user's vicinity (i.e., data within a given spotlight radius). Filter engine and location engine search and retrieve information based on the user's current location and other context information as given in their profile.

TIP uses a service-based approach with event-based communication between its components [8]. Internal services that can interact with the system core include recommendation services [9], while external ones include a link to Greenstone [10]. The TIP/Greenstone bridge had previously been used to provide a link to Greenstone. However, in that version of TIP, the Digital Library had to be location-encoded. That is, within each document, location markers were set (i.e., XML encoding of place-names) and the service had to scan the documents for these place names and geo-code them to obtain co-ordinates. There was no uniform access to internal and external location data and no spatial index. Each service was treated as a separate entity.

This service-oriented TIP architecture allowed us to modify how the location context is handled via the location engine, and extend it to provide access to external resources via the use of services. These features are the reasons why we chose TIP for our work.

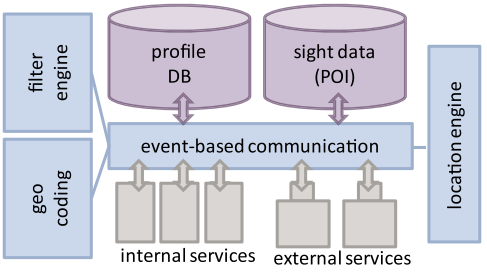

Fig. 2. Simplified TIP architecture.

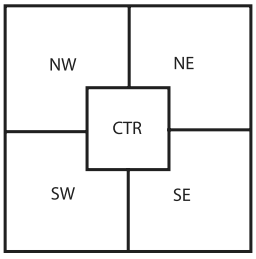

Fig. 3. mqr-tree node layout.

2.3. Spatial indexing and the mqr-tree

Spatial indexing [11] provides an efficient mechanism for accessing data using location information, such as geographic co-ordinates. One such spatial index is the mqr-tree [12], which organizes both points and objects uniformly, using the spatial relationships that exist between them (e.g., one object is northeast of another). This supports efficient navigation within the spatial index. A performance comparison versus other spatial indexes [12] shows that the mqr-tree supports a one-path search when indexing point data — a feature that most other spatial indices cannot achieve. For these reasons we chose to augment the mqr-tree for our work. We summarize the mqr-tree, including its structure, insertion algorithm and search algorithm here. More details, including a larger example of a tree constructed using multiple insertions, is available in [12].

The mqr-tree is an approximation spatial access method that organizes both objects and points uniformly for efficient search and retrieval. All objects and subregions of space that contain objects are represented using minimum bounding rectangles (MBRs). Because a point can be represented as an MBR of zero area, it is treated in the same manner as an object. Therefore, an *objectMBR* is a representation of either an object or a point in the description below.

mqr-tree structure and object placement. Associated with every node is a *node MBR*, which defines the subspace that is represented by a node. The node MBR encompasses all MBRs in both the node and any subtrees that are descendants of the node. All nodes in the mqr-tree use the same node structure, which is depicted in Fig. 3. A node has 5 locations (i.e., "quadrants") — northwest (NW), northeast (NE), southeast (SE), southwest (SW) and center (CTR). Each location in a node contains one of the following two entries:

- (*MBR, obj_ref*) where *MBR* is either a point or an approximation for an object in the database, and *obj_ref* is a pointer to it, or
- (*subtreeMBR*, *node_ptr*) where *subtreeMBR* is the *nodeMBR* that encompasses all MBRs in the subtree referenced by *node_ptr*.

Every object MBR and subtree MBR is placed in an appropriate location in a node by comparing the center of the object MBR (or subtree MBR) with the center of the node MBR. The relationship between the object (or subtree) MBR and the node MBR is defined by the following relationship rules:

- If the center of the object (or subtree) MBR is equal to the center of the node MBR, the object (or subtree) MBR is placed in the CTR location of the node.
- If the center of the object (or subtree) MBR is northwest of the center of the node MBR, the object (or subtree) MBR is placed is the NW location.
- If the center of the object (or subtree) MBR is northeast of the center of the node MBR, the object (or subtree) MBR is placed is the NE location.
- If the center of the object (or subtree) MBR is southeast of the center of the node MBR, the object (or subtree) MBR is placed is the SE location.

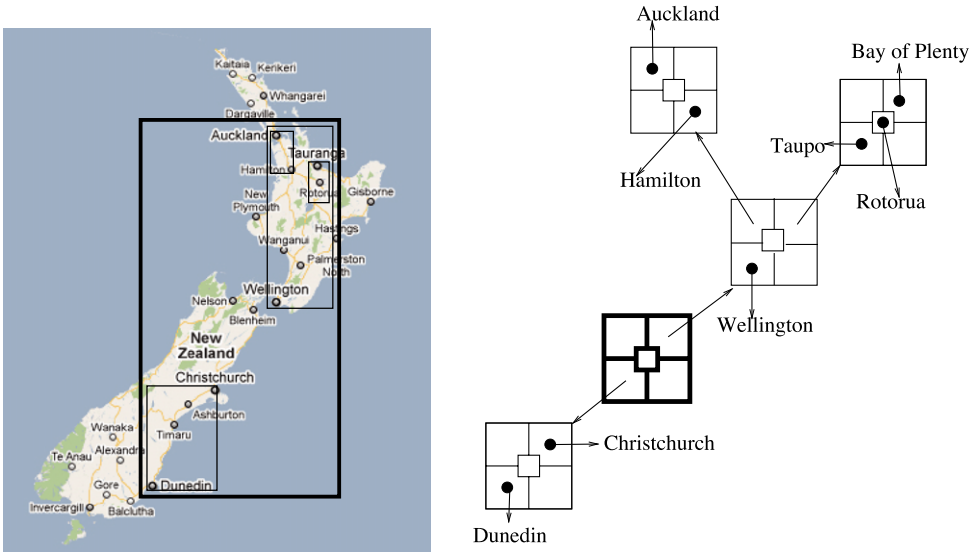

Fig. 4. mqr-tree example.

- If the center of the object (or subtree) MBR is southwest of the center of the node MBR, the object (or subtree) MBR is placed is the SW location.

Because all MBRs are handled in the same manner, a node can contain references to both objects and subtrees.

mqr-tree example. Fig. 4 depicts an mqr-tree that contains points representing a selection of sights in New Zealand. The root node (highlighted in bold lines on the index) represents the node MBR (highlighted in bold lines on the NZ map). It references two MBRs with corresponding subtrees, which represent the subtrees of communities on the North Island and South Island respectively. Note that the South Island subtree is southwest of the root node MBR, and therefore this MBR is placed in location SW. Similarly, the North Island subtree is northeast of the root node MBR, and therefore is placed in location NE. Looking at the node containing Dunedin and Christchurch, note that each city is located SW and NE (respectively) of the node MBR containing them, and are placed SW and NE (respectively) in the node. For the node containing Taupo, Rotorua and the Bay of Plenty, because Rotorua overlaps the center of the node MBR containing it, it is placed in location CTR. For one last example, the node containing Wellington also contains references to two other subtrees.

Although the resulting mqr-tree resembles a quad-tree [13], several differences exist, including lower coverage of whitespace, and flexible and dynamic coverage of space. More discussion on the differences can be found in [12].

Insertion algorithm. The mqr-tree insertion strategy has the following main tasks: (1) inserting objects and points so that the relationships between them and all other MBR in the tree (as stated above) are maintained in every node, and (2) handling the placement of other existing MBRs in the tree that no longer have the proper relationship between other MBRs, due to the insertion of a new object.

Beginning at the root node, the node MBR is adjusted (if required) to include the new object MBR. Then, the appropriate location — NE, SE, SW, NW, or CTR — is identified for inserting the object MBR, using the relationship rules given above. If the location is empty, the object MBR is inserted. If the location contains another object MBR, a new child node is created from the location, and both the new and existing object MBRs are inserted in the new node. Otherwise, the search for an insertion location continues in the same manner in the subtree.

At the same time, for each node along the insertion path, a change in the node MBR may result in other object or subtree MBRs no longer being in the proper location in the node with respect to the node MBR. Fig. 5 shows one example where the potential exists for displaced MBRs. Here, the node MBR (i.e., the outer rectangle) has been increased to accommodate an object. Parts of the former node MBR (shown in gray) have changed locations. For example, parts of the former NE location are now part of the NW, SW, SE and CTR (not shown) locations respectively. Any MBRs whose centers are in these locations must be relocated.

These MBRs must be removed and re-inserted in the same manner as the new object, although the initial node of insertion for these displaced objects is their current node, and not necessarily from the root node. The process of relocating displaced MBRs may require the relation of other MBRs in the process. Therefore, the relocation process will continue until either: (1) no more node MBR changes occur, or (2) changes in any node MBRs do not displace the other MBRs or subtrees.

It must be noted that not all insertions result in multiple displacements and re-insertions. As the mqr-tree increases in size, in many cases there exists a location where an object MBR can be inserted and no displacements of other MBRs occur in the tree. Therefore, most displacements will occur when the mqr-tree is smaller.

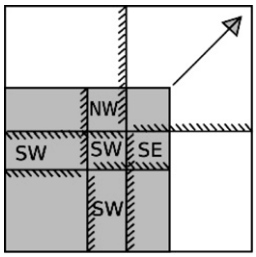

Fig. 5. Potential object displacement due to node MBR expansion.

Search algorithms. The mqr-tree supports both a point search and a region search in the same manner. First, beginning at the root node, the search point (or region) is tested for overlap with any existing MBR in each of the five locations location of the root. If a location contains no MBR, then the search does not proceed via the location any further. If the location contains an MBR and overlap exists, then the search proceeds in one of the following ways. If the location contains an object MBR, then the corresponding object is returned from the database as part of the result. If the location contains a subtree MBR, then the same search repeats in the corresponding subtree.

If the mqr-tree contains MBRs that represent objects (i.e., non-point) data, the potential for overlap between all MBRs exists, and therefore a search may proceed along several paths of the tree. In addition, if the search being performed is a region search, then a search may also proceed along multiple paths. However, if the mqr-tree is indexing point data only and the search is a point search, then the search will only proceed along one path of the tree due to the existence of no overlap [12].

3. Related work

We discuss here existing work related to tourist information systems in general, location-based indexing, and the benchmark for spatial indexing (R-tree).

3.1. Mobile tourist information systems

Many mobile tourist information systems have been proposed, including AccesSights [14], CATIS [15], CRUMPET [16], Cyber-Guide [17], Guide [18], Gulliver's Genie [19] and TIP [7]. Most of these systems focus on providing structured information and recommendations. Additionally, most of these systems only provide the user with information that is directly maintained by the system itself. They do not provide information from sources that are maintained in external sources, such as digital libraries.

The first system to be proposed that integrates a mobile tourist information system with an external digital library is TIP. The TIP/Greenstone Bridge [10] provides the functionality for TIP to connect to and retrieve information from a Greenstone digital library [2]. In addition to the information that is provided by TIP for a specific location, a user can obtain further information on items and features in a Greenstone digital library that are related to the same location. Some limitations of the TIP/Greenstone bridge include the following. First, it only retrieves external documents from Greenstone collections. Second, it only works with geographically aware digital libraries (i.e., place names must be identified and marked up). Third, TIP locations are defined by place names, while a mobile device defines location using GPS co-ordinates, extra steps are required to obtain a place name and have the user select an appropriate one before the user is able to access a Greenstone collection. Finally, information needed to access internal TIP information and external Greenstone information are kept separate, and cannot be presented to the user for consideration at the same time.

3.2. Location-based middleware

As mobile applications have become more widely used, architectures and communication paradigms have been developed that support the specific requirements of mobile environments. Event-based communication has been suggested for mobile applications, and is also used for TIP [8,20,21]. Similarly, a number of middleware solutions have been suggested that use a service-oriented approach for mobile environments [22,23,8].

Middleware for a location- or context-aware system provides a unified interface for context acquisition and access. However, neither communication paradigms nor composition/orchestration of software components inherently deal with the design of the context data repository. Aspects of data management and fast access structures are typically addressed at the back-end or server. In this case, the data repository is assumed to be relatively static while the queries use a moving spotlight radius. Another group of applications deals with situations in which both the query and the data are dynamically changing (e.g., to answer questions such as "where are my friends?"). In these cases, the repository has to deal with localized queries and high update rates [24]. However, this paper focuses on the co-operation between a number of repositories, each of which remains relatively static.

3.3. Location-based indexing

Many location-based indices have been proposed in the literature. Surveys on established location-based indexing are presented in [25,11,26]. These approaches are classified into three categories [25]: (1) main memory methods — data structures which primarily manage points and reside in memory, (2) point access methods — structures which manage point data and are organized for storage and retrieval from secondary storage, and (3) spatial access methods (i.e., spatial indexes) — structures that manage both object and point data and are organized for storage and retrieval from secondary storage. We focus on spatial access methods, because the mqr-tree — the index that we have chosen to base our work on — falls into this category.

Spatial access methods provide uniform access to both point and object data. Also, they remain height-balanced in the presence of a dynamic object set. Many spatial access methods are proposed in the literature [12,27–36]. They can be classified [25] into approximation, clipping, and mapping methods. We focus our summary on approximation methods, since the TIP-tree is part of this category.

Approximation methods store a hierarchy of approximations of both objects and the space occupied by subsets of objects. Since the space is not partitioned, approximations can overlap. Many approximation methods are proposed, including the R-tree [27], the R^*-tree [28], GiST [29], the X-tree [30], the R*Q-tree [31], the PR-tree [32], the 2DR-tree [33], the VoR-tree [34], the DR-tree [35], the MSI approach [36], and the mqr-tree [12].

A limitation of most existing spatial access methods is their inherent one-dimensional node structure, which forces data in multi-dimensional space to be mapped to one dimension [33]. However, there exists no n-dimensional to one-dimensional mapping that preserves all topological relationships between objects in space [25]. Furthermore, traversing a two-dimensional space, such as a map, will not be efficient if data is forced into a one-dimensional ordering. The 2DR-tree and mqr-tree are the only spatial access methods that utilize two-dimensional nodes in all levels of the index structure, and therefore can preserve topological relationships among objects. Because the mqr-tree has been shown [12] to be the more efficient with respect to search performance among the two approaches, we base the TIP-tree on the mqr-tree in our work.

In addition, some applications of location-based indices exist in mobile information systems.

Wang et al. [37] propose the use of a grid index which partitions the overall space that is being managed by the system. This grid index tracks the locations of all active region queries and the users who reside in them at any given time. On arrival in a new grid cell, the user requests the queries (and results) that overlap the cell, which are then stored on their mobile device. These requests must be made when moving to a new cell or when the queries change. Although the grid index can handle moving points, the user's mobile device must cash queries, results and a portion of the (possibly sparse) grid file itself, which is much additional (and possibly wasted) overhead on devices with limited memory.

Jung et al. [38] propose a continuous monitoring of nearest neighbor queries using a modified grid index. Attached to every cell is a minimum bounding rectangle (MBR) that defines the actual bounds occupied by points in the cell. If an nn query does not overlap the MBR, none of its points will be part of the current result. As with other grid-based approaches, one limitation is with many portions of the index being sparse.

Hu et al. [39] propose a proactive caching strategy for supporting multiple spatial query types (e.g., nn, region, range). Their strategy stores not only the results from previous queries, but also the R-tree nodes that lead to the existing results. For new queries, the cached partial R-tree is searched first, and any objects that are not accessible from the cached R-tree are fetched from the server. Although savings in transmission are possible, the caching overhead and local processing may prove to be prohibitive.

Tao et al. [40] propose an approach for continuous nearest neighbor queries that utilizes an R-tree to speed up searches. One limitation of this approach is the need to perform repeated "vertical" searching for new co-ordinates. Park et al. [41] improve on their strategy for processing path nearest neighbor queries. Their strategy performs a "horizontal" search for nearest neighbor objects along the trajectory. Although their strategy is shown to be efficient when an R-tree or grid index is used, it requires that the trajectory be known in advance.

Lee et al. [42] propose a strategy that augments results and validity regions with additional complementary objects, in order to reduce the number of queries to the server. An R-tree is used to locate both valid and complementary objects. One potential drawback is repeated searching in order to obtain a sufficient number of complementary objects.

3.3.1. Benchmark spatial indexing: R-tree

The TIP-tree is an approximation method. Therefore, we chose to evaluate the TIP-tree against both the mqr-tree, on which the TIP-tree is based, and the R-tree, which is considered one of the benchmark approaches to spatial indexing. A summary of the mqr-tree is presented in Section 2. We present a summary of the R-tree below.

The R-tree is a spatial access method based on the B-tree [43]. It is a balanced tree that is built using a bottom-up approach and obeys minimum space utilization rules for its nodes. The R-tree stores approximations (using minimum bounding rectangles, or MBRs) of spatial objects (and points), the subregions that contain objects, and the subregions that contain other subregions. This reduces the coverage of regions in the data space that contain no objects. In addition, the R-tree avoids the use of multiple entries for an object by allowing subregions to overlap. A leaf node stores MBRs for spatial objects. A non-leaf node stores MBRs for subregions that contain a set of spatial objects and/or other subregions. Fig. 6 gives an example R-tree that indexes the selection of locations in New Zealand. The root node contains three MBRs, one for each leaf node. Note that,

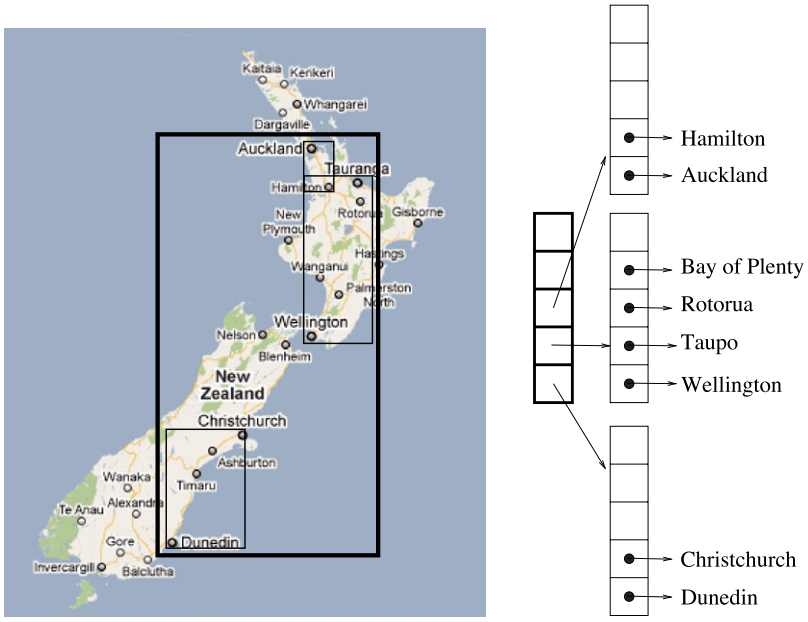

Fig. 6. R-tree example.

even through the R-tree is indexing points, overlap still exists between the leaf nodes containing (Auckland, Hamilton) and (Bay of Plenty, Rotorua, Taupo, Wellington).

The R-tree supports both point queries and region queries in the same manner. A search begins at the root by examining all entries to find MBRs that overlap the query region or point. For all MBRs that overlap with the query, the search continues in the corresponding subtrees until it reaches either one or more leaf nodes, or non-leaf nodes with no MBRs overlapping the query region or point. The search algorithm retrieves MBRs for all qualifying leaf node entries and tests them for overlap with the query region or point. For all points or minimum bounding rectangles that overlap, the corresponding objects are returned for the result. Note that because overlap exists, multiple paths may be searched.

If we refer again to Fig. 6, if a point query is specified on Hamilton, even though Hamilton is in one node, both of the nodes (Auckland, Hamilton) and (Bay of Plenty, Rotorua, Taupo and Wellington) would need to be searched, requiring an extra, wasted node access.

The insertion algorithm first identifies the appropriate leaf node for storing the new object, which is either a point or a minimum bounding rectangle. The insertion path contains minimum bounding rectangles that require the least enlargement to include the new object. Overflow is handled by splitting the affected node into two nodes so that the new nodes cover the smallest amount of area and both nodes are not checked for the same query. An exhaustive split, quadratic split, or linear split can be performed to produce the two new nodes. Details on the various splitting strategies can be found in [27].

After inserting the new object, the insertion algorithm must adjust the minimum bounding rectangles along the insertion path. If a split causes an overflow in the parent node, the algorithm propagates the split upwards as necessary. If a split propagates to the root, a new root node is created.

4. Usage scenario 1 — retrieval of resources

In this section, we demonstrate how TIP can be utilized to access both internal and external resources. The scenario has a user arriving in Hamilton, NZ, who wants to access information on the city using TIP. In addition, our user finds links to other locations and is interested in looking into these further. Fig. 7 shows the sequence of interactions that will take place (clockwise from top right).

First, TIP determines the user's current GPS co-ordinates from their mobile device. Using these co-ordinates, TIP searches its index to identify all resources — both internal and external to TIP — that are near the user's current location. TIP presents three links to the user — the first is information on Founders Theatre that is maintained by TIP itself, the second is a link to a Greenstone collection on plants and gardens, and the third is a DSpace collection on the history of the Waikato region. Our user is interested in gardens, and decides to follow the link to the Greenstone collection. TIP uses the Greenstone service (via the TIP/Greenstone Bridge described previously [10]) to obtain documents from the Plant & Garden collection.

Next, the user follows the link for the Hamilton Chinese Garden document. In addition to information on the garden itself, there is a link to the location of Wellington. Our user plans to visit Wellington in the near future, and decides to click on the place name for more information. TIP obtains the GPS co-ordinates that corresponds to Wellington and searches the index

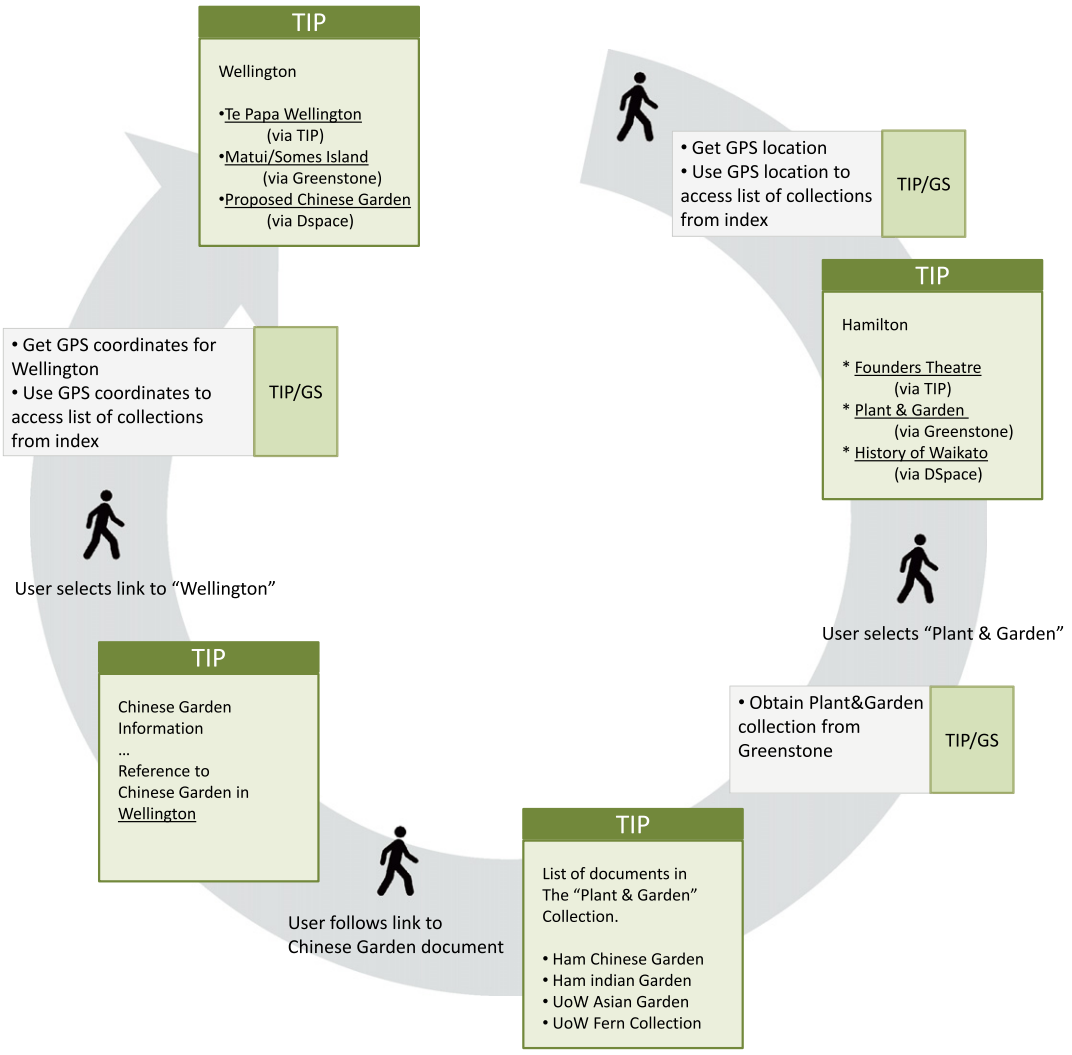

Fig. 7. TIP/Greenstone interaction sequence.

for all internal and external resources that it has on the location of Wellington and displays these to the user for further follow up. This is depicted at the end of Fig. 7.

5. Modified TIP system architecture

In this section we describe the interaction within the modified TIP architecture to facilitate location-based access to TIP and external resources.

Fig. 8 shows the conceptual design of TIP with our extensions. In particular, two components were added: The first is a spatial index that organizes TIP and external resources using their location co-ordinates. The second is services to handle the retrieval of collections and documents managed by different digital library software systems.

Given the user's current co-ordinates, which are provided by the mobile device (one of the internal services), the co-ordinates are first sent to the geo-spatial index module. On the index, either a point search or a navigation (see Section 6) is performed, which will return either:

1. a set of references to TIP sights and/or external digital library links, or
2. nothing, if no information is found near the co-ordinates provided to the system.

Once a set of references to sights and links is returned, the user's profile can then be applied via the filter engine to further reduce the number of links by eliminating those that would not be of interest to the user. The resulting set of references is sent to the user's mobile device in the form of links to be represented, e.g., as icons on a map or as list of points of interest. Note that end-user visualization elements are omitted in the architecture; user communication including push and pull mode of communication is discussed later in Section 8.7.

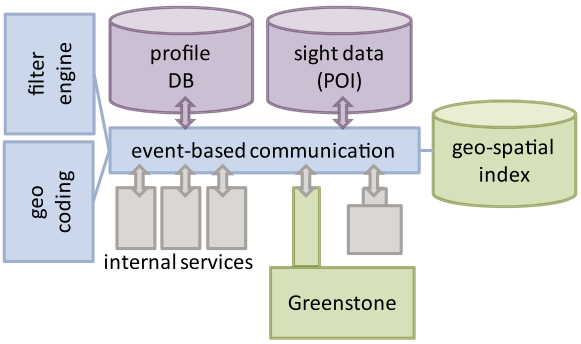

Fig. 8. Modified TIP architecture and link to Greenstone.

As the user selects one these links, a request is sent to the TIP server and the corresponding information is fetched in the following way. For a reference to TIP sight, the corresponding information is retrieved from the Sight Data repository (containing information about points of interest). For a digital library link, the link information is sent to the appropriate digital library service, which will fetch the page(s) that corresponds to the link. This information is then sent back to the user's mobile device.

If the user selects a link for a different location (in the form of a Place Name) in a page, then the place name is sent to the geo-coding module first to obtain co-ordinates before performing the search for TIP sights and digital library links as given above.

Our extended architectures show the service connection to Greenstone only. TIP already has a bridging service that we can use here [10]. In addition, this architecture can provide access to digital library collections managed by other software systems by adding the appropriate retrieval services and thus retrieval from other external DL sources, such as DSpace and Fedora digital libraries is handled similarly to the one shown here.

6. TIP tree: spatial index with navigation

In this section, we present the TIP-tree index and its associated algorithms. The goals of the TIP-tree are twofold: (1) to provide efficient *uniform location-based access* to information for both TIP information and external digital libraries, and (2) to provide efficient *in-place navigation* (instead of repeated searching from the root of the index) of the location context so that up-to-date information can be provided continuously to the user. The TIP-tree inherits the general 2-dimensional node structure from the mqr-tree [12]. It improves upon the mqr-tree by adding and adjusting four features: introduction of points of interest in list nodes (described in Section 6.1), enhanced point search algorithm (Sect. 6.3), enhanced insertion and update algorithm (Sect. 6.2), and introduction of in-place navigation (Sect. 6.4). These four features together ensure uniform location-based access and in-place navigation. Furthermore, the navigation algorithm as well as the enhanced point search and insertion algorithms take into consideration the mobile environment for which the original mqr-tree was not suitable.

6.1. Points of interest in list nodes

We added list nodes to capture information about points of interest (POIs) at a given location. Thus a list node is an array of items (e.g., sights, collections, etc.) that can be indexed and accessed by the same data point (i.e., co-ordinates) in TIP. We use a list node to avoid inserting multiple identical co-ordinates for different sights and collections in the same place. Therefore, a list node entry stores information on each item, including name, whether it is internal or external to TIP, and if it is external, the name of the digital library that manages it and instructions on how it is accessed (i.e., usually via a URL). Fig. 9 shows an example TIP-tree node, which extends the mqr-tree node (Fig. 3). It depicts two locations - NW and SE - each referencing a list node that contains collections.

Fig. 10 shows the example of a TIP-tree that is indexing resources across New Zealand. Note how the locations of the cities shown on the map (left-hand side of Fig. 10) correspond to their location in the index: Bay of Plenty, north-west of Taupo (North Island) and Christchurch, north-east of Dunedin (South Island). Note how the center point in the top-right node refers to Rotorua, which is situated in the middle between Taupo and the Bay of Plenty (see map).

As one can see, the TIP-tree structure utilizes the same two-dimensional node structure from the mqr-tree (see Fig. 4), which has been extended to include list nodes from every point in the index. For example, the point representing Auckland provides access to a list node that contains resources on All Blacks games (located in TIP), a Greenstone digital library on the Auckland Museum (external to TIP), and a DSpace collection containing documents from the University of Auckland (also external to TIP). In the other link from the same level, we can find the list node for the Hamilton points of interest that we used in our Usage Scenario 1 on retrieval of resources (see Section 4).

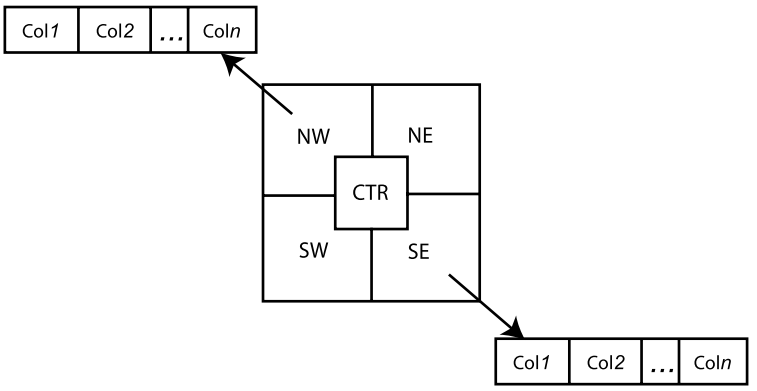

Fig. 9. tip-tree node layout.

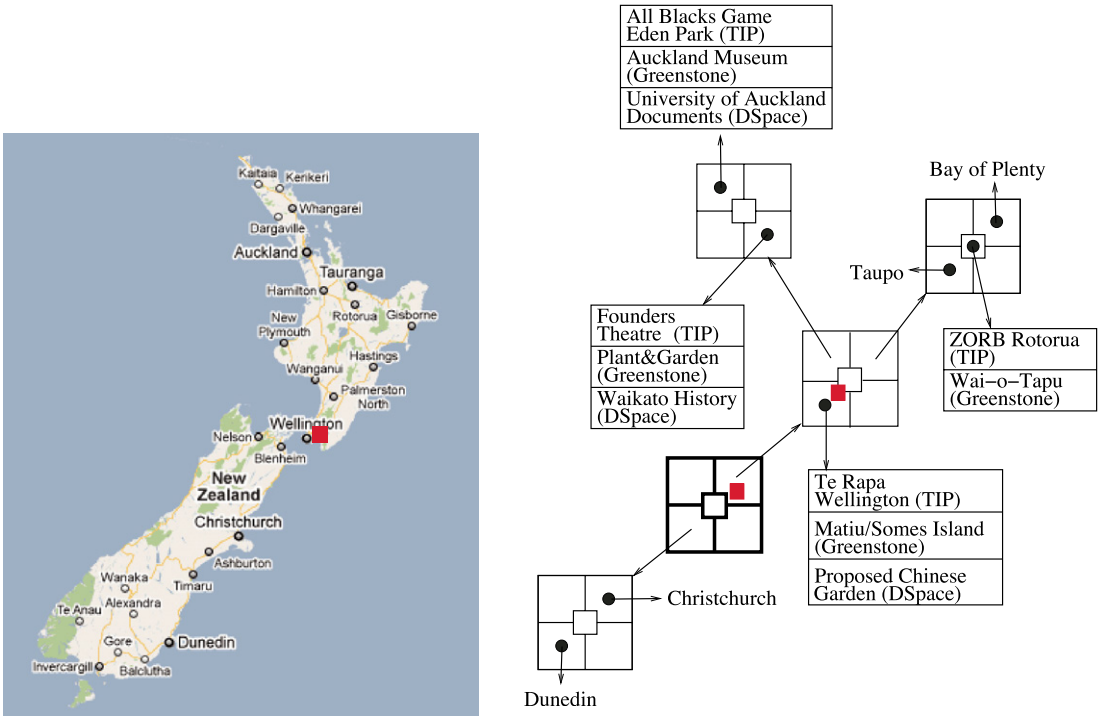

Fig. 10. TIP-tree with point search example.

Furthermore, the center nodes in the mqr-tree were used to handle multiple overlapping center points. By contrast, the center nodes in the TIP-tree are free to be used for additional POIs. This can be seen in the top-right node in Fig. 10: the list for Rotorua attractions is linked from the node center.

6.2. Insertion and updating

The TIP-tree insertion algorithm (which is also used for constructing the initial TIP-tree via repeated insertion) is adapted from the insertion algorithm for the mqr-tree, with the following additions and changes. First, for a co-ordinate that is to be added to the index, if a matching co-ordinate already exists, insertion is only a matter of adding a record for the new resource to the corresponding list node. Otherwise, insertion of a new co-ordinate requires the additional step of creating a new list node for the new co-ordinate and adding a record for the new resource to it.

The TIP-tree insertion algorithm works in the following way. For each resource to be added (i.e., either TIP information or from an external digital library) and corresponding co-ordinate that indicates its location, a location is found for inserting the new point by using the mqr-tree insertion algorithm (see Section 2.3 for a description of this algorithm). However, during the process of traversing a path of the tree in the search of an insertion location, a search for an existing co-ordinate that matches the one being inserted can also take place. Therefore, beginning at the root node, a test is made to see if the point

already exists in the node. If so, then the corresponding list node is fetched and a record for the new resource is added to it. Otherwise, the node MBR for the root node is expanded (if required) to accommodate the new point and the location that contains the new point is identified. If that location is empty, the point is inserted and a corresponding list node is created with a record for the new resource added to it. Otherwise, the search for either an existing point or a location for a new point continues in the subtree until either an existing point is found, or a location is found for a new point. As with the mqr-tree, any existing points that are no longer in their proper locations must be removed and re-inserted.

Although the example index (Fig. 10) organizes resources based on physical location, it is also possible to add a collection based on a location being mentioned in a digital library collection. For example, if a collection at the University of Auckland mentions Hamilton, this collection can then also be added to the corresponding list for Hamilton by using the co-ordinate that represents Hamilton.

Updates to an existing resource in the TIP-tree occur by deleting the existing resource record, and re-inserting it with the updated information and/or location co-ordinate, using the above insertion strategy. If the situation exists where the only resource record is being relocated, then the corresponding co-ordinate must be deleted as well.

6.3. Enhanced point searching

The TIP-tree point search algorithm (i.e., search for single co-ordinates) is an extension of the original mqr-tree point search algorithm. The main differences are: (1) because the data set that is being searched consists of points (i.e., single co-ordinates and not MBRs), the point that resides in the location (or "quadrant") that is found by the search point is counted as a "match", and (2) a list node must be fetched for a "matching" point.

Beginning in the root node of the TIP-tree, the search identifies the location in which the search point is located. If the location contains a point, the corresponding list node is retrieved. All items that are contained in the list node are sent to the user for display on their mobile device. If the location is empty (i.e., contains neither point nor link), then the search terminates. Otherwise, the search continues in the subtree in the same manner until either a point is found, or an empty location is found.

For example, if the user searches for information near Wellington, the system's GPS automatically submits search co-ordinates referring to the location marked with a large, solid square on the map in Fig. 10 (Latitude −41.294833, Longitude 174.795799). The index search starts with the tree root (shown in bold in the lower center of the tree in Fig. 10) and identifies the current location in the NE location (upper right, indicated by a large, solid square). As the location does not contain a POI but a link, the system retrieves the linked subtree node. Within the this node, the system again searches for the user location and identifies the SW location (lower left, indicated by large, solid square). This location contains a point, which links to a list of POIs (e.g., Te Rapa and other items). This list is retrieved and displayed to the user.

6.4. In-place navigation

The TIP-tree navigation algorithm is an "in-place" search algorithm that navigates the search through the index as updated user location co-ordinates are sent from the mobile device. This strategy differs significantly from point search algorithm in that the current search begins from the previous result location, and is not re-initiated from the root node. This results in fewer node accesses, which translates to significant time savings in fetching results for the user.

Fig. 11 shows the pseudocode for navigating the TIP-tree. Navigation begins by performing an initial TIP-tree point search (described above in Section 6.3) to find a starting node for navigating the index.

Next, within the bounded region (as defined by the node MBR) covered by the node, the navigation algorithm identifies which location the search point is currently located (NW, NE, SE SW, CTR). Following this, a check is performed to see for an existing co-ordinates in the current location. If a co-ordinate exists, the corresponding list node that contains digital library collections and TIP information will be retrieved for the user, along with the distance the user is from the co-ordinate. Otherwise, nothing is displayed back to the user, since no co-ordinate (and therefore, no related collection or TIP information) exists at this location.

The user will then move, which causes the co-ordinates of the search point to be updated. This will result in one of the following situations:

1. The search point remains in the current location. Nothing changes other than updating the user's distance from the existing co-ordinate in the location, if any co-ordinate was found earlier. If any digital library collections and TIP information were fetched for the user, these are still available to them.
2. The search point moves to a different location, and there is nothing (i.e., point or subtree) in the location. In this case, nothing occurs into the search point is updated again.
3. The search point moves to a different location, and a co-ordinate exists in this location. In this case, a new set collections (and distance from them) that correspond to the new co-ordinate are retrieved and displayed to the user.
4. The search point moves to a different location, and this location references a subtree and the search point is within the bounded region (node MBR) of the root node for the subtree. In this case, the search point is sent to the subtree and its location is re-evaluated in the root node of the subtree.
5. The search point moves to a different location, and this location references a subtree but the search point is not within the bounded region (node MBR) of the root node for the subtree. Here, nothing occurs until the search point is updated again.

```
traversing = true;
search point (x,y) = findStartingNode(x,y);
while(traversing) {

   (x,y) = update_co-ordinates(x,y);

    /*is search within the bounded region covered by current node?*/
    if( (x,y) within node_MBR(X) ) {
          Loc = determineLoc(X);

          if(contains_co-ordinate(Loc)) { /*does location contain co-ordinate?*/
              Sights = access_all_sites(Loc);
              display_all_sights(Sights);
          }

          else if(with_node_MBR_of_child(Loc)){ /*traverse subtree?*/
          X = child_node(X);
          }
          /*otherwise, there is no information to display, so nothing is done
             until co-ordinates are updated.*/
    }

    /*if search outside current region, move to parent region if exists.*/
    else if( has_parent_node(X) ) {
          X = parent_node(X);
    }
    /*otherwise, outside of space covered by index, so terminate search*/
    else {
          traversing = false;
    }
}
```

Fig. 11. TIP-tree navigation pseudocode.

6. The search point falls outside the current bounded region. It is sent to the parent node and its current location is re-evaluated using the above criteria.

This process repeats until either: (1) the search point falls outside of the space covered by the index, or (2) the user switches to pull mode. In either case, the navigation search is terminated, and must be re-started with an initial search to a node where navigation will start again.

To demonstrate the navigation algorithm, an extended example is presented the next section, in our second usage scenario.

7. Usage scenario 2 — navigation

We present here a usage scenario that demonstrates the TIP-tree navigation. We consider a user who arrives at Auckland Airport, and then makes her/his way to Wellington. Along the way, TIP will notify our user of any nearby points of interest. Fig. 12 shows the locations (location indicated by the points that are, referenced by circled numbers) where TIP will display information to the user, and the corresponding locations (using the same numbered points) in the TIP-tree.

The user will begin receiving information from the TIP-tree on arrival at Auckland Airport (Latitude −37.00806, Longitude 174.79167, ①). An initial search of the index will set the search point in the bounded region (and NW location) containing the co-ordinate for Auckland (following the algorithm explained in Section 6.3). Corresponding to the co-ordinates for Auckland are the digital library collection for the Auckland Museum, a digital library collection from the University of Auckland, and information on the upcoming All-Blacks rugby game at Eden Park. The latter two collections are managed by Greenstone and DSpace respectively, and both reside on servers that are external to TIP. These collections are fetched from their respective servers using the corresponding retrieval service (see Fig. 8) and made available to the user. The third item is fetched from within TIP and also presented to the user.

As the user proceeds to Hamilton (②), the location of the search point will be updated (SE) but the search remains in the same bounded region. The information available for Hamilton will now be fetched and displayed to the user. As the user continues along the west side of the North Island (③), the search point will be transferred to the parent node and its NW location. Since the TIP-tree contains no information on this area, nothing is retrieved for the user. Finally the user makes their way to Wellington (④, SW location, same bounded region) and information on Wellington is now displayed to them.

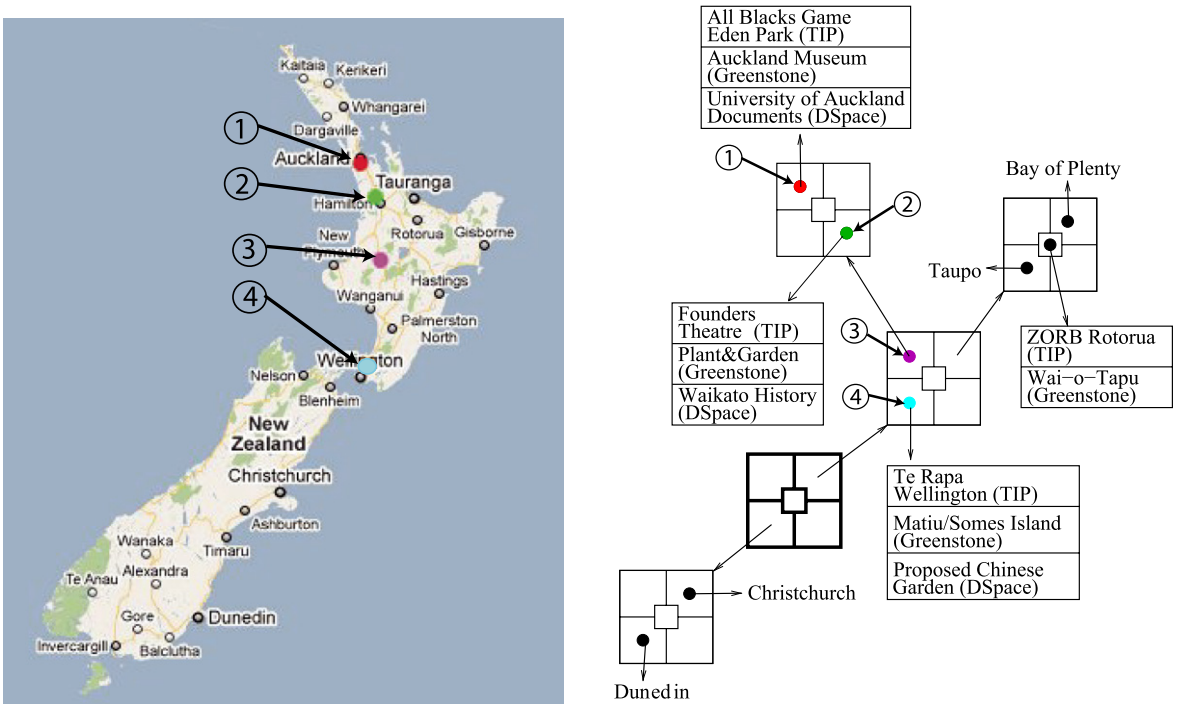

Fig. 12. TIP-tree with navigation example.

8. Evaluation

In this section we present our empirical evaluation of the TIP-tree. We compared the performance of the TIP-tree against both the original mqr-tree (described in Section 2.3) and the R-tree (see Section 3.3.1), which is considered one of the benchmark strategies in spatial access methods. The TIP-tree uses the navigation strategy presented here to process the user's trajectory, while both the mqr-tree and the R-tree process a user's trajectory by performing a point search on every co-ordinate in the trajectory.

We first present an overview of our evaluation methodology. Then, we present and discuss the results of our tests, and finish with further discussion of issues that arose from our evaluation.

8.1. Overview of methodology

Here, we present the data sets, evaluation criteria and a brief overview of the test setup that was used.

Data sets. The evaluation used five sets of co-ordinates — four point sets (PS1, PS2, PS3 and PS4) and two trajectories (T1 and T2). PS1 and PS2 each contain 10,000 co-ordinates and consist of a mix of both real and synthetic points that represent various locations. The real co-ordinates represent locations of existing communities in New Zealand and were obtained from the World Gazetteer. The synthetic co-ordinates represent other locations in New Zealand and are randomly generated. PS1 contains locations on the North Island of New Zealand, PS2 contains locations in the Waikato, a region in the North Island of New Zealand. Thus even though PS1 and PS2 contain the same number of points, PS2 describes a much smaller region than PS1 and thus the point density is higher. The rationale for using additional, randomly created, locations is that a collection of documents may refer to a region that does not exist in the World Gazetteer (e.g., iceberg photos from off the coast of New Zealand) or may be located in a remote area that is not maintained in the World Gazetteer.

The point set PS3 consists exclusively of real data. Specifically, the data set contains 1016 locations of interesting architecture in the city of Hamilton, which were pulled from a local trade site.[3] The points in this set are real-world locations; however, the point set is, naturally, not a complete set of all POIs possible in Hamilton. As a consequence, some suburbs of Hamilton are represented with fewer houses in the data set than other suburbs.

The remaining point set PS4 consists of one million synthetic points that represent various locations on the North Island of New Zealand. This data set allows us to evaluate the scalability of our algorithm for a very large number of points.

The first trajectory T1 set, T1, contains 712 co-ordinates, which represent a user's 16 km route from Hamilton to Cambridge, New Zealand. The second trajectory set, T2, contains 400 co-ordinates, which represent a user's 3.5 km route

[3] We used anonymized data about local properties for sale on.

test	variable	data set	trajectory
1	Index size	PS1 (real+synthetic data, larger region)	-
2	Density of points	PS2 (real+synthetic data, smaller region)	-
3	Size of path (length)	PS1 (real+synthetic data, larger region)	T1 (longer)
4	Size of path (density)	PS2 (real+synthetic data, smaller region)	T1 (longer)
5	Index size	PS3 (real data)	T2 (shorter)
6	Size of Path (length)	PS3 (real data)	T2 (shorter)
7	Index size - large data set	PS4 (synthetic data)	T1 (longer)

Fig. 13. Overview of evaluation: 7 tests.

from the University of Waikato through a nearby neighborhood. The trajectories were recorded using a mobile device and trail capturing software, while traveling by vehicle.[4]

Test criteria. The primary performance criteria in all of our tests is the number of disk accesses required to locate whether data of interest exists at a specific location or not. We evaluate the worst case scenario — no caching is used. Therefore, every time a node is required, it must be fetched from secondary storage, which requires one disk access. We discuss the influence of our findings on the overall access time in Section 8.7.

Test setup. We performed seven tests in our evaluations (see table in Fig. 13). The first test studies the effect of the number of points in the index on the number of disk accesses required to locate information. The second test is similar to the first, except the points are drawn over a smaller geographical region. This leads to an increased geographical density in points, while the same number of points is represented in the index. The third test studies the effect of the number of co-ordinates along the trajectory on the number of disk accesses required to locate information. The fourth test is similar to the third, but uses the overall set of points drawn over a smaller geographical region. The fifth and sixth tests evaluate the effect of the number of points in the index and the number of co-ordinates along a trajectory respectively, when using the real-world data set. Finally, the seventh test evaluates the effect of the number of points in the index as the number of points reaches one million points. Each test and its results are presented below in detail.

8.2. Test 1: #overall indexed points

For the first test, we used point set PS1, the set of 10,000 points referring to locations around the North Island of New Zealand. We performed ten test runs. Each run constructed (a) a TIP-tree, (b) an mqr-tree, (c) an R-tree using a subset of the points sorted by increasing longitude, and (d) 10 R-trees using a subset of the points that were randomly sorted.

Index settings (a) and (b) use our new TIP-tree and the original mqr-tree, respectively. Setting (c) with an R-tree over sorted points is included here because it was the situation that was found to produce the best R-tree results. For setting (d), we created multiple R-trees with random order of points, because its structure is depending on the order in which the points are inserted. This is not the situation with either the TIP-tree or the mqr-tree because the final structures are not insertion-dependent. Then, all 10 indexes processed the trajectory of 712 co-ordinates.

For each index setting, the test runs used 1000, 2000, 3000, up to 10,000 points to create the trees (see *x*-axis in Fig. 14). The average number of disk accesses per co-ordinate was calculated for each data structure. In addition, one average is calculated over all 10 R-tree runs.

Fig. 14 depicts the results of the test runs. From the graph, we observe that on average, the TIP-tree only needs to execute one disk access per co-ordinate in the user's trajectory. For the mqr-tree, the number of disk accesses is higher and increases as the number of points in the mqr-tree increases. For 1000 points, the average number of disk accesses is approximately 4.8 and for 10,000 points it is approximately 6.2. For the R-tree built from a sorted data file, the number of disk accesses also increases, with the lowest at 14 for 1000 points to the highest at 38 for 10,000 points. The same trend occurs with the R-tree built from random data ordering, but is more significant, with an average number of disk accesses of approximately 30 for 1000 points, up to an average of 350 when indexing 10,000 points.

The reason for the improved performance of the TIP-tree is because each "search" begins at the last node visited by the previous co-ordinates in the trajectory. If several subsequent co-ordinates are close to each other, then it is very likely that the same node will be visited repeatedly, and no further search is needed to find the correct node. Because the original mqr-tree performs all searches from the root, this will result in a higher number of nodes being accessed.

8.3. Test 2: density of points in a region

For the second test, we evaluated the effect of density on the average number of disk accesses required to locate information. Here, we used point set PS2 with 10,000 points, as in the first test. However, all points are locations from in and close to the Waikato Region, which is a subsection of the North Island of New Zealand. Therefore, we are looking at the effect of increasing density on the number of disk accesses required to locate information.

[4] Recorded using the My Tracks App on Andoid phone, available on Google Play with id.

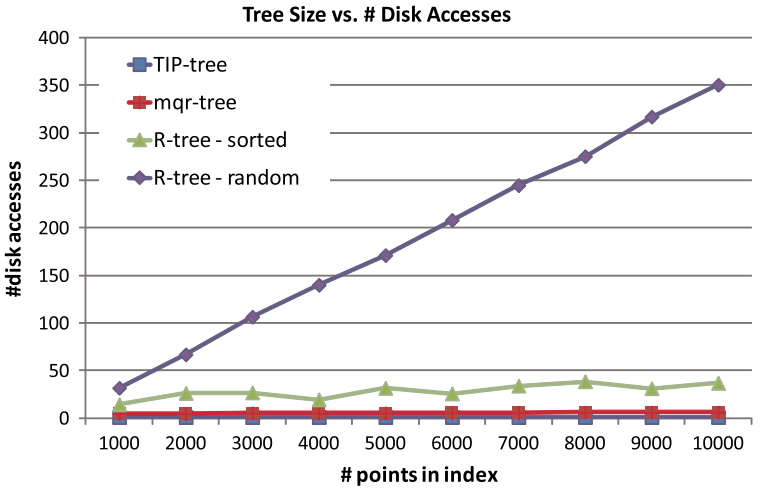

Fig. 14. Overall index size (test 1).

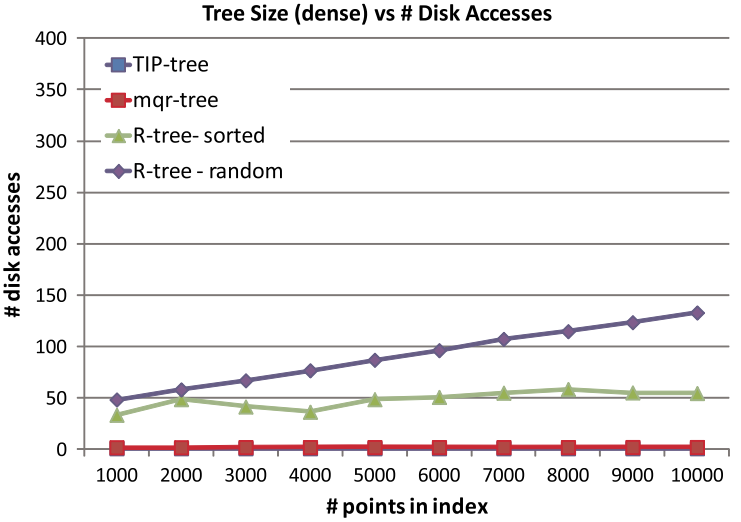

Fig. 15. Density of points in a region (test 2).

We perform the same test runs for the second test as performed for the first test. Fig. 15 depicts the outcome of the second test. Again, we find that on average the TIP-tree requires only one disk access per co-ordinate on the trajectory. However, we also find that, although the TIP-tree has a lower average disk access over the mqr-tree, the improvement is not as significant as that found in the first test since the average number of disk accesses required by the mqr-tree is between 2 and 2.25. However, there is still significant improvement over using the benchmark R-tree strategy, which ranges between 50 and 130 disk accesses in the random ordered case, and between 34 and 59 disk access in the ordered data case.

The slight improvement of the TIP-tree over the mqr-tree is at first sight a surprising result, since we had assumed that the higher the density of points, the more disk accesses that would be required. What is happening is the following. The mqr-tree and the TIP-tree achieve a significantly lower amount of coverage of space when a dense set of points are being indexed. This means that more points are being placed higher in the index and fewer rectangles that represent the coverage of space are being stored (i.e., reduced tree height). This results in points (and therefore, information) being located much faster by the mqr-tree than in an mqr-tree that is indexing points that are more spread out geographically.

8.4. Test 3 and 4: #co-ordinates on traversal path

For the next two tests, we evaluated how the number of co-ordinates on a user's trajectory affects the average number of disk accesses. Will too many or too few co-ordinates cause more disk accesses than necessary?

For test 3, we used PS1 — the set of 10,000 points that represents the North Island — to construct the TIP-tree, mqr-tree, and 10 R-trees (which we use to calculate one average for the R-tree). For test 4, we used the denser set PS2 of 10,000 points

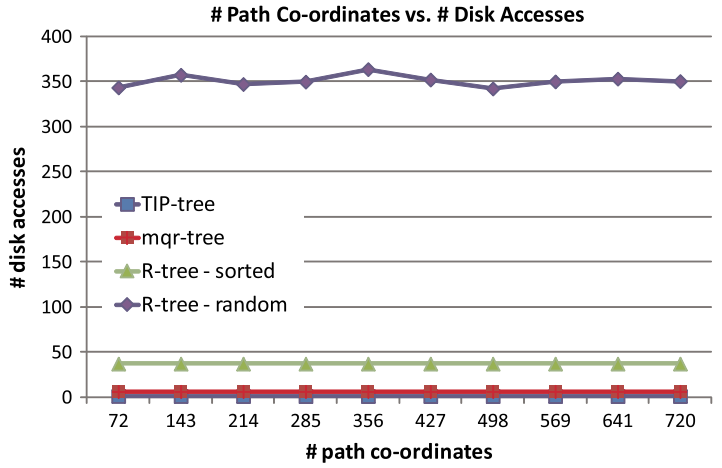

Fig. 16. Size of traversal path (test 3).

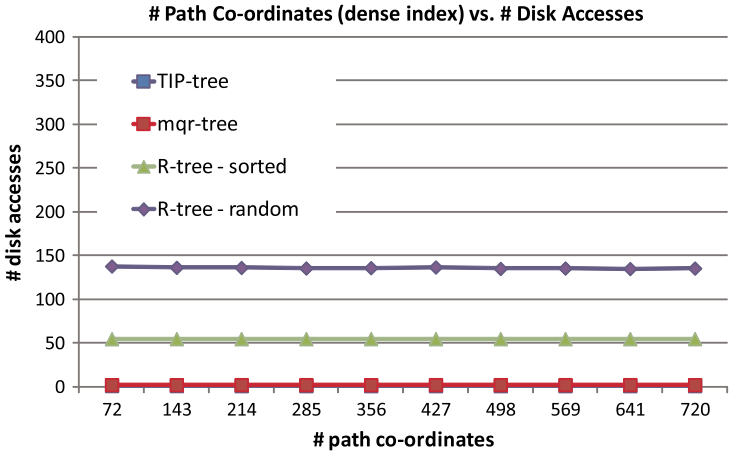

Fig. 17. Size of path — density (test 4).

that represents the Waikato Region. For both tests, we performed ten test runs, using approximately 10%, 20%, 30% … up to 100% of the co-ordinates in the Hamilton to Cambridge trajectory T1.

Fig. 16 presents the results of test 3, while Fig. 17 presents the results of test 4. We observe that the TIP-tree achieves a lower average number of disk accesses for each co-ordinate over the mqr-tree, and significantly so over the R-tree. However, in all cases, we also find that the average number of disk accesses is constant, regardless of the number of points on the trajectory that are used for locating information. This is a very significant result, especially with respect to efficiency of the TIP-tree — fewer points does not result in an increase in the number of disk accesses, and therefore traversal remains efficient.

8.5. Test 5 and 6: evaluation using real-world data

For the next two tests, we evaluated on the real-world data set PS3 on how the number of co-ordinates in the index and the number of co-ordinates on a user's trajectory affect the average number of disk accesses. For the overall index size, we ran 10 tests, where each used 10%, 20%,…,up to 100% of the 1016 points in creating the indexes, and were evaluated with the 400-point trajectory T2. For the trajectory test, we also ran 10 tests, where each used 10%, 20%,…,up to 100% of the 400 points in the second trajectory point set, on an index that contained all 1016 points.

Fig. 18 presents the results of test 5, while Fig. 19 presents the results of test 6. We find in the results of these tests with real-world data a very similar trend to the one seen in the results of tests 1 to 4, which used a mix of real and synthetic data. In test 5, the number of overall points in the index results in a constant number of disk accesses in both the TIP-tree and the mqr-tree. For the R-tree on random data, we also see a similar pattern of significant increase in the number of disk accesses as the number of points in the index increases, while on sorted data we see the similar pattern of mild increase in

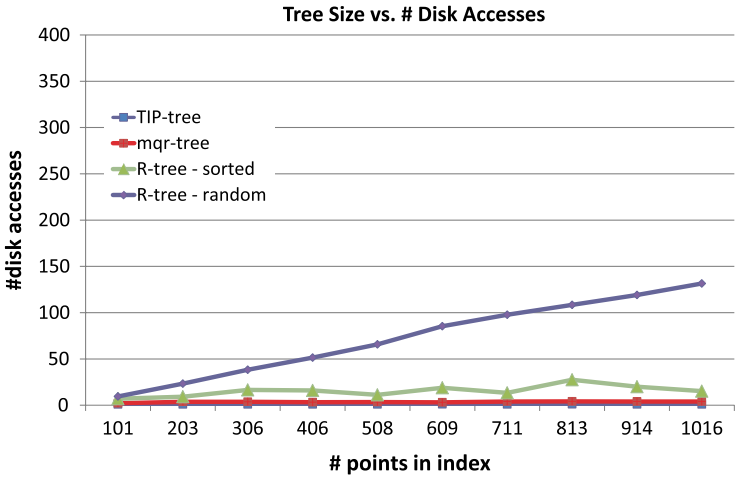

Fig. 18. Index size — real data (test 5).

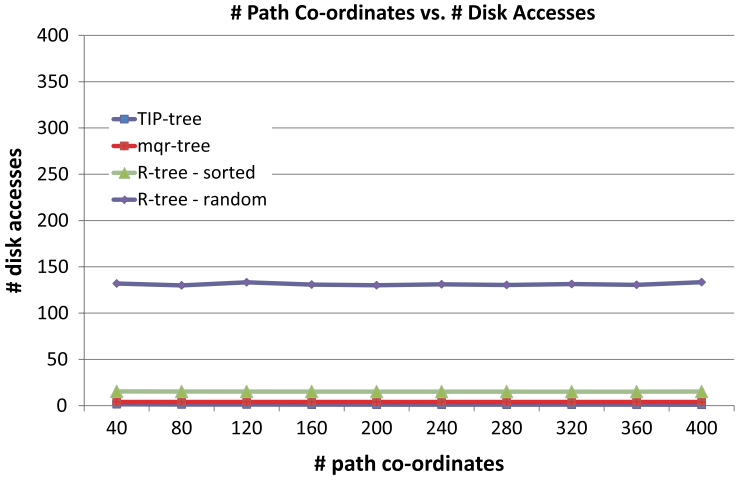

Fig. 19. Size of path — real data (test 6).

the number of disk accesses. Similarly in test 6, the number of points on the trajectory results in a constant number of disk accesses required to obtain data from any of the evaluated indexes.

8.6. Test 7: scaling

For the final test, we evaluated on the much larger synthetic data set PS4 on how a much larger number of co-ordinates in the index affects the average number of disk accesses. Here, for the overall index size, we ran 10 tests, which each again used 10%, 20%,…,up to 100% of the one million points.

Fig. 20 presents the results of test 7. Here, we present in the graph the results of the TIP-tree navigation algorithm versus the mqr-tree search only, because we found the results for the R-tree (both sorted and random) to be significantly higher in this case (approximately 100 and into the thousands, respectively) and would detract from the contribution of TIP-tree navigation. We find that our results compare with the trends found in previous tests involving index size. Specifically, the number of disk accesses required on average for the TIP-tree is still approximately one, while we find a slight increase in the number of disk accesses required by the mqr-tree. Although the initial number of disk accesses required to find a starting point for TIP-tree navigation does increase as the tree size increases, this is absorbed by the very few disk accesses required during navigation.

8.7. Further discussion

Execution time. Over all tests, we found that the TIP-tree and its traversal algorithm performed better than both the mqr-tree and the R-tree with respect to the average number of disk accesses per co-ordinate. Note that we performed a worst-case

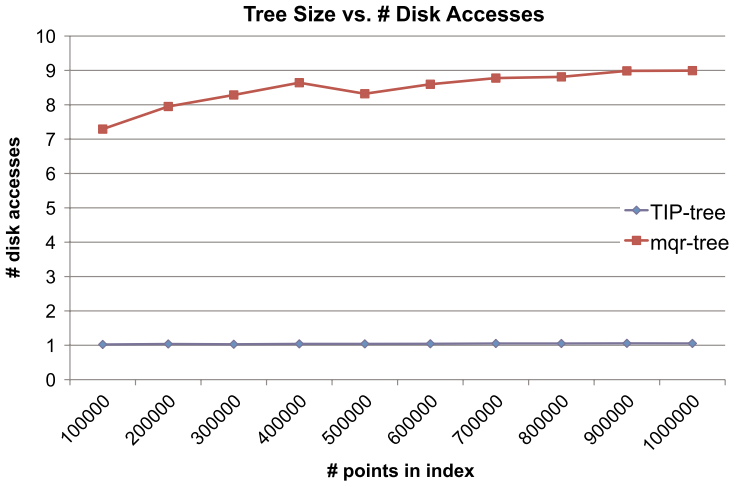

Fig. 20. Index size — large data sets (test 7).

evaluation — specifically, that every time a node is checked, a disk access is required. Average or best cases are influenced by both, the data distribution on a hard drive and the effects of caching. Both are largely independent of the tree structure used and are therefore not relevant for this evaluation.

The number of disk accesses is a standard measure for the quality of database indexes as it largely influences the query execution time. However, we also recorded the CPU execution time for each co-ordinate "search" (i.e., the time to execute the search once the data has been loaded into main memory). We found that the average execution time was less than 1ms in all cases.

The last factor influencing the overall query execution time is the network transmission time. However, as the amount of data to be transmitted back to the user remains the same, independently of the index used, the network transmission time is irrelevant for this evaluation.

Therefore, disk access time remains the crucial factor in how well this system performs. As we discovered during our evaluation, the average number of disk accesses required during navigation is approximately one, which includes the initial search. If the worst-case number of disk accesses required for the initial search (or a pull-mode search) is desired, the value can be estimated using the following formula:

$$NumberOfDiskAccesses = 1.5 * \log_5(n)$$

where n is the number of points in the TIP-tree. This heuristic refers to the estimated index tree height and a weighting factor. For $n = 1{,}000{,}000$, we would require approximately 13 disk accesses in the worst case to perform a pull-mode search or find the starting point for navigation.

Caching and pre-fetching. Depending on the user's location and surrounding, the same nodes may be accessed repeatedly (a situation we observed in our evaluation as well). For the TIP-tree, this would result in repeatedly accessing one node. For both the mqr-tree and R-tree, it would result in repeated access of the same path (or paths, in the case of the R-tree) of nodes to get to the node (or nodes) that several co-ordinates will require. If these nodes were cached in memory on server side, this could significantly reduce disk access time. Moreover, if caching were to take place on the mobile device instead of on the server, this would reduce the network transmission cost, the disk access cost and the execution time on the server. Similarly, predicting the user's trajectory enables pre-fetching of information from the server onto the mobile device. The use of an efficient index supports both caching as well as pre-fetching. Efficient strategies for mobile caching and pre-fetching have been investigated elsewhere [44,45] and are orthogonal to the problem considered here.

Tree initialization and update. Tree construction time is dominated by disk access time. Therefore, the time it will take to construct a TIP-tree with n points can be estimated with:

$$BuildTime = (SeekTime * 2.6 * n * FracWrite) + (TransferTime * 2.6 * n)$$

where SeekTime is the time required to move the read/write head into place, TransferTime is the time for transferring a 512-byte block of data, and FracWrite is the fraction of writes performed for the program by the operating system. The value of $2.6 * n$ is the number of blocks and is composed of: (1) records for each of the n points, (2) One list node (i.e., list of POIs) for each of the n points, and (3) the number of leaf and non-leaf nodes in the index, which is approximately $0.6 * n$. When evaluated assuming $n = 1{,}000{,}000$ randomly ordered points, a SeekTime of 18.5 ms, a TransferTime of 0.0046848 ms and a FracWrite value of 0.001,[5] we obtain an estimate of approximately 60 s. Note that this value will be lower with faster seek and transfer times.

[5] Values obtained through benchmark testing of server, not from factory specifications!

Updating with a new point or POI entry, however, does not require the entire TIP-tree to be brought into memory — only the non-list (i.e., non-POI) nodes. Therefore, the time required to insert a new point is:

$$InsertTime = 2 * ((SeekTime * 1.6 * n * FracWrite) + (TransferTime * 1.6 * n)).$$

Note that the entire calculation is multiplied by 2 to account for reading in the index as well as writing it back to disk. Given the values $SeekTime = 18.5$ ms, $TransferTime = 0.0046848$ ms and $FracWrite = 0.0002$, the time required to insert a new point into a tree containing a million points is approximately 26 s.

If insertion time becomes too long due to the number of points already in the tree, insertions could be done periodically in batch mode. Also, insertion can be done "off-line" while the existing TIP-tree is in operation, with the new tree taking over once it is built. Note that the tradeoff is really low search and navigation times for the user.

User communication: continuous querying vs user-activated search. The communication between system and end-user is implemented as push mode: whenever new information becomes available (e.g., because the user enters a new region) it is actively transferred to the user's mobile device. Push communication is particularly suitable for mobile applications so that the user is not required to repeatedly request new information. The internal implementation of this end-user push mode is a continuous pull (i.e., location data is regularly sent to the server to check for new data in the current region). So the continuous pulling of data is triggered by updates in user location. It may also be triggered by the user directly (e.g., when following a link).

Presentation of the information may take different forms depending on user preference (e.g., the POIs may be presented as icons on a map). The issue of data overload in areas of dense POIs can be addressed in several ways. One way is by maintaining a system-defined threshold of the number of POIs to display at a given time. As a consequence users are notified about some items later (i.e., once they are closer). This makes the notifications arrive later (from the user's point of view), but the user no longer needs to focus on so many items, only the ones that are very close by. Another measure to manage the high density of information is late filtering: the data presented to the user can be filtered further on-the-fly (e.g., by use of the spotlight radius, user profile or any other context). TIP also supports several presentation modalities: information may offered to the user as a list (as shown in Figs. 1 and 7), as pointers on a map or as audio information [46]. Large amounts of data may be easier to filter from a map and then selected items are viewed or listened to. In addition to the push mode, TIP supports pull-style communication. This is particularly appropriate if the user has to inspect the available information for some time (e.g., reading a text about a POI). In these cases the user may request more data once they have processed the previous ones, instead of being overwhelmed with continuously arriving new data.

User evaluation. User experiments can elicit feedback on user experience (interface navigation, usability etc.). However, the focus of the research presented in this paper is one of mobile access to spatial data. Unless the measured performance shows a noticeable lag close to seconds, it will not influence the user experience. Furthermore, network access and other issues may have a further influence on real-life performance. Thus the quality of the indexing is here measured purely in terms of database access. However, our TIP/Greenstone combination in a location-based service has been tested and studied on users recently, details can be found in [47,46].

9. Conclusion

In this paper, we presented a strategy for efficient, uniform, location-based access to digital library collections that are external sources to a context-aware mobile system. Using the TIP framework, we utilized a spatial index to manage the context of location. We showed how access to resources from within and outside of the tourist information system can be carried out seamlessly. We presented an algorithm for traversing the TIP-tree to continually provide information to the user, so that repeated searching from the beginning is not necessary. Finally, we executed an empirical evaluation in which we compared the TIP-tree with other spatial access methods. The results of our evaluations show that the TIP-tree and its traversal algorithm are the most efficient for handling a user's trajectory as they used the least number of disk accesses when searching for location-based data. Thus the TIP-tree navigation is very efficient and significantly outperforms traditional spatial search.

Other contributions of this work include the following. A one-pass access to digital library collections do not need to be "geographically aware" (i.e., the DL collections no longer need to carry their own location mark-up). Any additional resource can be added by specifying its GPS location. Furthermore, the TIP-tree can handle collections that reference multiple locations.

In conclusion, our work provides a strategy for context-aware mobile systems to co-operate with digital libraries in a seamless and efficient manner. This means access is no longer a constraint to information contained within the mobile information system. This further allows systems to keep a minimal amount of local information, which can be enhanced by external sources, such as co-operating digital libraries, as required.

Future work. From our current work, we identify several technical aspects that might further improve the TIP-tree.

- *continuous region search:* All information within a specified radius or rectangular region is fetched. An issue with the existing co-ordinate based search is the "oscillation" between two different result sets if the user crosses continuously over a node or location boundary. Incorporating a continuous region search will solve this problem by presenting either

both sets of information to the user, or a subset of it in the case of a dense set of POIs. In either case, the information that is presented to the user in this situation will not be in constant change. Although the TIP-tree supports region or radius (i.e. range) searching, it currently does not support the continuous navigation of a search region or radius. We plan to address this in the next software version.

- *region-sized POI support:* Currently, the TIP-tree supports POIs that are represented by co-ordinates (e.g. Auckland Museum). Although POIs of larger area (e.g. Lake Taupo) can be supported by the TIP-tree if access to them is restricted by our point searching strategy (Section 6.3), they do not work within our navigation strategy. It would be desirable for the TIP-tree to support the navigation of both point and region-based POIs.
- *context-based search:* Incorporating searching on other forms of context, such as user interests could easily be incorporated by performing an additional search on the results of the location context search. However, it is more desirable to include other types of context into the indexing structure so that only one search is required.
- *search refinement:* refining or widening a search means searching higher up versus lower down in the tree. Both of these require modifications to the current TIP-tree structure so that all leaf nodes are on one level of the tree and all data resides in the leaf level.

Finally, the software described here is a service-based collaboration between several software packages. The digital library packages are available for download on the internet (e.g., Greenstone can be obtained from www.greenstone.org). The TIP-tree code will be released in the near future once some housekeeping has been performed. The mobile TIP software is planned to be released as an android app.

Acknowledgments

We wish to thank the reviewers for their constructive comments that helped to improve our paper.

References

[1] W. Osborn, A. Hinze, TIP spatial index: efficient access to digital libraries in a context-aware mobile system, in: Proc. 23rd Australasian Database Conference, ADC, 2012, pp. 127–136.
[2] I.H. Witten, D. Bainbridge, D. Nichols, How to Build a Digital Library, Elsevier, 2009.
[3] R. Tansley, M. Bass, M. Smith, Dspace as an open archival information system: status and future directions, in: Proc. 10th European Conf. on Digital Libraries, 2006.
[4] C. Lagoze, S. Payette, E. Shin, C. Wilper, Fedora: an architecture for complex objects and their relationships, Int. J. Digit. Libr. 6 (2) (2006) 124–138.
[5] A. Hinze, A. Voisard, Location- and time-based information delivery in tourism, in: Proc. 8th Int'l Symp. Advances in Spatial and Temporal Databases, 2003, pp. 489–507.
[6] A. Hinze, K. Loeffler, A. Voisard, Contrasting object-relational and rdf modelling in a tour information system, in: Proc. 10th Australian World Wide Web Conf., 2004.
[7] A. Hinze, A. Voisard, G. Buchanan, TIP: personalizing information delivery in a tourist information system, J. IT & Tourism 11 (3) (2009) 247–264.
[8] A. Hinze, M. Rinck, D. Streader, Anonymous mobile service collaboration: quality of service, in: From Active Data Management to Event-Based Systems and More, in: LNCS, vol. 6462, Springer, 2010, pp. 141–158.
[9] A. Hinze, Q. Quan, Trust- and location-based recommendations for tourism, in: Cooperating Information Systems (Coopis), 2009, pp. 414–422.
[10] A. Hinze, X. Gao, D. Bainbridge, The TIP/Greenstone bridge: a service for mobile location-based access to digital libraries, in: Proc. 10th European Conf. Digital Libraries, 2006, pp. 99–110.
[11] S. Shekhar, S. Chawla, Spatial Databases: A Tour, Prentice Hall, 2003.
[12] M. Moreau, W. Osborn, mqr-tree: a two-dimensional spatial access method, J. Comput. Sci. Eng. 15 (2012).
[13] R. Finkel, J. Bentley, Quad trees: a data structure for retrieval on composite keys, Acta Inform. 4 (1974).
[14] P. Klante, J. Krshe, S. Boll, Accessignts — a multimodal location-aware mobile information system, in: Proc. 9th Int'l Conf. on Computers Helping People with Special Needs, Paris, France, 2004.
[15] A. Pashtan, R. Blatter, A. Heusser, CATIS: a context-aware tourist information system, in: Proc. 4th Int'l Workshop on Mobile Computing, Rostock, Germany, 2003.
[16] S. Poslad, H. Laamanen, R. Malaka, A. Nick, P. Buckle, A. Zipf, Crumpet: Creation of user-friendly mobile services personalized for tourism, in: Proc. IEE 3G2001 Mobile Communication Technologies Conf., London, UK, 2001.
[17] C. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton, Cyber-guide: a mobile context-aware tour guide, ACM Wirel. Netw. 3 (1997) 421–433.
[18] K. Cheverst, K. Mitchell, N. Davies, The role of adaptive hypermedia in a context-aware tourist guide, Commun. ACM 45 (5) (2002) 47–51.
[19] G. O'Hare, M. O'Grady, Gulliver's genie: a multi-agent system for ubiquitous and intelligent content delivery, Comput. Commun. 26 (11) (2003) 1177–1187.
[20] R. Meier, V. Cahill, On event-based middleware for location-aware mobile applications, IEEE Trans. Softw. Eng. 36 (3) (2010) 409–430.
[21] A. Holzer, P. Eugster, B. Garbinato, Alps — adaptive location-based publish/subscribe, Comput. Netw. 56 (12) (2012) 2949–2962.
[22] J. Viterbo, V. Sacramento, R. Rocha, G. Baptista, M. Malcher, M. Endler, A middleware architecture for context-aware and location-based mobile applications, in: Proc. 32nd Annual IEEE Software Engineering Workshop, SEW '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 52–61.
[23] J.H. Schiller, A. Voisard (Eds.), Location-Based Services, Morgan Kaufmann, 2004.
[24] J. Myllymaki, J. Kaufman, High-performance spatial indexing for location-based services, in: Proc. 12th Int'l Conf. on World Wide Web, WWW '03, ACM, New York, NY, USA, 2003, pp. 112–117.
[25] V. Gaede, O. Günther, Multidimensional access methods, ACM Comput. Surv. 30 (1998) 170–231.
[26] L. Balasubramanian, M. Sugumaran, A state-of-art in R-tree variants for spatial indexing, Int. J. Comput. Appl. 42 (2012).
[27] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proc. ACM SIGMOD Int'l Conf. Management of Data, 1984, pp. 47–57.
[28] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proc. ACM SIGMOD Int'l Conf. Management of Data, 1990, pp. 322–31.
[29] J. Hellerstein, J. Naughton, A. Pfeffer, Generalized search trees for database systems, in: Proc. 21th Int'l Conf. Very Large Data Bases, 1995, pp. 562–573.
[30] S. Berchtold, D. Keim, H.-P. Kriegel, The X-tree: an index structure for high-dimensional data, in: Proc. 22nd Int'l Conf. Very Large Data Bases, 1996, pp. 28–39.
[31] J. Qiu, X. Tang, H. Huang, An index structure based on quad-tree and R*-tree- R*Q-tree, J. Comput. Appl. 23 (2003).
[32] L. Arge, M. de Berg, H. Heverkort, K. Yi, The priority r-tree: a practically efficient and worst-case optimal R-tree, in: Proc. 2004 ACM SIGMOD Int'l Conf. on Management of Data, 2004.

[33] W. Osborn, K. Barker, An insertion strategy for a two-dimensional spatial access method, in: Proc. 9th Int'l Conf. Enterprise Information Systems, 2007.
[34] M. Sharifzadeh, C. Shabani, VoR-tree: R-trees with Voronoi diagrams for efficient processing of nearest neighbour queries, Proc. VLDB Endowment 3 (2010).
[35] G. Li, J. Tang, A new DR-tree K-nearest neighbour query algorithm based on directive relationship, in: Proc. Int'l Conf. Environmental Science and Information Application Technology, 2010.
[36] A. Al-Badarneh, A. Al-Alaj, A spatial index structure using dynamic recursive space partitioning, in: Proc. 2011 Int'l Conf. on Innovations in Information Technology, 2011.
[37] H. Wang, R. Zimmermann, W.-S. Ku, Distributed continuous range query process over moving objects, in: Proc. 23rd Int't Conf. Database and Expert System Applications, 2006.
[38] H. Jung, S.-W. Kang, M. Song, S. Im, J. Kim, C.-S. Hwang, Towards real-time processing of monitoring continuous k-nearest neighbour queries, in: Proc. 2006 Int'l Conf. Frontiers of High Performance Computing and Networking, 2006.
[39] H. Hu, J. Xu, W. Wong, B. Zheng, D. Lee, W.-C. Lee, Proactive caching for spatial queries in mobile environments, in: Proc. 21st Int't Conf. on Data Engineering, 2005.
[40] Y. Tao, D. Papadias, Q. Shen, Continuous nearest neighbor search, in: Proc. 28th Int'l Conf. Very Large Data Bases, 2002, pp. 287–298.
[41] Y. Park, K. Bok, J. Yoo, An efficient path nearest neighbour query processing scheme for location-based services, in: Proc. 17th Int'l Conf. Database Systems for Advanced Applications, 2012.
[42] K. Lee, W.-C. Lee, H. Leong, B. Unger, B. Zhang, Efficient valid scope computation for location-dependent spatial queries in mobile and wireless environments, in: Proc. 3rd Int'l. Conf. Ubiquitous Information Management and Communication, 2009.
[43] R. Bayer, E. McCreight, Organization and maintenance of large ordered indices, Acta Inform. 1 (1974) 173–188.
[44] Y. Michel, A. Hinze, Traditional pre-fetching and caching of limited use for mobile applications, in: Conference on Mobile and Ubiquitous Information Systems (Mobile und Ubiquitäre Informationsysteme — MMS), 2009, pp. 25–38.
[45] Q. Ren, M.H. Dunham, Using semantic caching to manage location dependent data in mobile computing, in: Proc. 6th Annual Int't Conf. on Mobile Computing and Networking, MobiCom '00, ACM, New York, NY, USA, 2000, pp. 210–221.
[46] A. Hinze, D. Bainbridge, Tipple: location-triggered mobile access to a digital library for audio books, in: Proc. Joint Conf. on Digital Libraries, JCDL'13, ACM, New York, NY, USA, 2013.
[47] A. Hinze, D. Bainbridge, Listen to tipple: Creating a mobile digital library with location-triggered audio books, in: Proc. 2nd Int't Conf. Theory and Practice of Digital Libraries, 2012, pp. 51–56.