Custom Local Search

Naren Datha[1], Tanuja Joshi[1], Joseph Joy[1], Vibhuti Sengar[2]

[1]Microsoft Research India
196/36 2nd Main
Bangalore, 560 080, India
{narend, v-tjoshi, josephj}@microsoft.com

[2]Microsoft Corporation
One Microsoft Way
Redmond, WA 98052, USA
vibhutis@microsoft.com

ABSTRACT

Many popular online services provide "local" or "yellow-pages" search, but none of them allow users to customize the search over user-specified data. Instead, searches are performed over spatial data provided by data aggregators. This paper describes a novel system for providing custom local search over user-provided custom spatial datasets logically combined with detailed street-level vector data. We present the algorithms and architecture underlying our prototype system, which can handle a range of queries, including complex queries with ambiguities and misspellings. We show that the system provides custom local search with precision and recall figures that often exceed that of a commercial local search service. Further, we show that our system scales gracefully with the number of individual custom data sets being served simultaneously.

1. INTRODUCTION

Online local search, or "yellow pages" search, is something many people take for granted these days. Popular services provide answers to queries such as "Seattle Pizza" and "Pediatric hospitals in Bellevue." Answers take the form of listings of relevant organizations along with contact information. We refer to the underlying data being served as "Yellow Page Data" or YP Data.

Online providers typically source YP Data from consolidators who in turn may rely on specialized YP Data collecting firms [8]. The aggregated data in turn is considered as an aggregate pool of data on organizations, over which local search is performed, either over the names of the organizations or, more commonly, over a set of textual attributes (such as "food" or "apparel") tagged to the organization.

It is not possible today to search over custom data sets; end users have to make do with whatever YP Data is fed into the systems of these online providers.

Consider the following scenarios:

- A community-driven site that maintains a user-provided list of pedestrian-friendly features in the city. The site would like to provide local search for these features, supporting freeform text queries like "water fountains along Sammamish River Trail, Redmond."

- A heritage-focused organization would like to provide local search over all the geocoded heritage sites in their database, using freeform text queries that are robust with respect to spelling variations, such as "Bellivue Main Street sculptures" (Bellevue is misspelled).

To our knowledge, no commercial or research platform supports these scenarios beyond providing basic text search. In fact, there are significant challenges to building such a system, especially one that scales well with the number of custom data sets being supported concurrently. One challenge is how to parse the freeform text query, given the possible large variations in the structure of the query with no clear demarcation of terms from different data sets. Another challenge is how to actually execute the query over what is essentially a fusion of the custom datasets with the (typically much larger) detailed street-level vector data, and to do so in a way that scales gracefully with the number of distinct custom data sets being supported.

In this paper, we present concepts and algorithms for implementing custom local search and present results from an evaluation of our prototype system that supports local search over multiple custom geographic data sets. Search can be constrained to one or more of these custom data sets, while leveraging detailed street-level vector data to emulate a full featured, but highly customized local search service, where the *users*, not service providers, have control over the data being searched over.

2. MOTIVATING EXAMPLE

We consider customized local search needs of three hypothetical organizations, and use their example to motivate the need for an integrated custom local search solution, as well as to illustrate the inherent challenges of providing such a service.

Consider a heritage preservation organization, A, that is tracking about 10,000 heritage sites, with fresh sites uploaded monthly. Each heritage site is geocoded, and has a set of textual attributes, such as its primary name, type of site (one among a set of categories that can include architecture attributes, religious/cultural affiliation if any, state of repair and so forth). The organization would like to provide dedicated custom local search functionality over their dataset, integrated with common geographic knowledge about the location of streets, localities and so forth. Sample queries they would like to support are listed in Table 1, rows 1-2.

A second organization, B, is maintaining a community site for pedestrian-friendly areas in a city. Members of the community submit spatial annotations, in the form of points, lines or polygons, along with textual attributes that further define the area

Table 1: Sample Custom Local Search Queries

No.	Custom Dataset	Sample Query
1	A (Heritage sites)	Fort ruins along the river Ganges
2		Ganesha Temples in Malleshwaram Bangalore
3	B (Pedestrian info.)	Seattle Kid friendly parks
4		Seattle water fountains near 4th ave and Pike St
5	C (Traffic 'cams)	Washington highway webcams
6		I-405 north webcams in Kirkland

(such as "kid friendly", "running trail" and "pets allowed"). The organization would like to allow the public to search for these areas using local search queries, such as those listed in Table 1, rows 3-4. A few thousand annotations are maintained, with daily updates. Figure 1 shows a mashup maintained at the site, showing results for the query in row 4 ("Seattle water fountains near 4th ave and Pike St") [1].

A third organization, C, a state highway department, would like to provide search over traffic web cameras that they maintain. They have a few hundred cameras under their jurisdiction. The list is updated every few months. They would like to support search, with sample queries as listed in Table 1, rows 5-6.

The three organizations' search needs, exemplified by the sample queries in Table 1, have the following characteristics in common:

- The organization controls the data (including frequency of updates) over which the search is performed: specific sets of heritage sites, pedestrian-friendly locations, and webcams, maintained by organizations A, B & C, respectively.
- Search queries include "contextual" terms that refer to street-level data such as streets and localities. This is natural for users to want to do, since the custom data is embedded within geographic regions.
- The street-level data is typically much larger than the custom data. For example, detailed street-level data for the state of Washington, USA, contains of the order of half a million entities (individual streets, localities, administrative boundaries, landmarks, etc), each with associated geometry.
- The search results must contain explicit references to custom dataset entities in order that the organizations can pull in auxiliary data it may maintain that is specific to each instance.

Queries of this type, which merge custom content with detailed street-level data, are not supported by any commercial or research system today, to the best of our knowledge. In fact, there are significant procedural and technical challenges to building custom search engines:

- Individual organizations typically will not have the financial and organizational heft to license detailed street-level data and manage the relationships with the providers of such data.
- Even if an organization does have access to street-level data, it can be non-trivial to parse out terms from the query that may refer to street-level vs. terms from the custom data set.

[1] Our prototype system provides exactly this kind of experience, for multiple custom datasets.

Figure 1: Results from a custom local search query

- Larger online map service YP search providers control their own sources of data, including frequency of updates.

Even for the YP data that commercial providers *do* index, to our knowledge, the flexible kinds of queries merging street-level landmarks with YP data as illustrated in Table 1 are simply not supported today. In particular, consider the queries in Rows 1 ("Fort ruins along the river Ganges"), and 4 ("Water fountains near 4th ave and Pike st, Seattle". The former query requests for YP data results along a linear feature (the river Ganges). The latter query request YP Data near an intersection of two streets (4th Avenue and Pike Street in Seattle). Most YP search services do not provide this level of precision in freeform text queries over their inbuilt data sources, let alone over custom datasets.

We describe a system and algorithms for building a search service that can concurrently support multiple individual custom search experiences, of the kind described in this section. Individual organizations simply have to provide their data sets, and each organization will get its own custom search experience. The system is very effective at dealing with ambiguity and errors in the queries, can parse the more complex queries like the queries in Rows 1 and 4 of Table 1, and scales well with the number of custom search datasets supported concurrently. Before we describe our system, we give a reasonably general description of the "Custom Local Search" problem in the next section.

3. PROBLEM DEFINITION

Consider a set of spatial datasets D_1, D_2, D_3, …, D_n. Each dataset consists of a set of spatial entities. Each entity consists of a *shape* and of a set of textual *attributes*. The shape can consist of one or more geometric primitives (points, polylines, polygons) specified in some world coordinate system. Textual attributes can be both category descriptors (such as "fort" or "ruins") and names (such as "Golconda fort"). Each entity has a unique identifier known as Entity Identifier or EID. The datasets are typically uploaded by multiple organizations acting independently. We consider performing textual queries over *subsets* of these datasets. In this paper, we focus on queries involving a *pair* of the data sets, (D_{cust}, D_{ctxt}), where D_{cust} is a *custom* dataset (one of many), and D_{ctxt} is a (typically much larger) shared, *contextual* data set. As mentioned in the previous section, the primary motivating scenario for this case is providing concurrent multiple custom local search experiences. Each search experience supports freeform local search queries over the fusion of a custom data set D_{cust} with a large "common knowledge" contextual data set D_s, which contains detailed geographic vector data, including precise geometry of streets, landmarks, localities and administrative

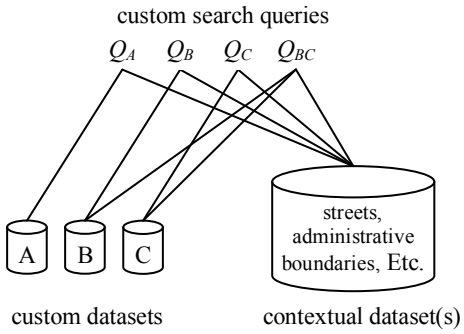

Figure 2: Queries over custom and contextual datasets

boundaries. A straightforward generalization of this is to consider queries over multiple custom datasets, and multiple contextual datasets. Figure 2 illustrates the relationship between custom datasets, contextual datasets and queries. Typically, any one organization would submit queries for a fixed subset of data sets.

3.1 Query Semantics

A custom local search query, *Q*, is the triple (*dataset-selection*, *geometric-scope*, *text-query*), where

dataset-selection specifies which datasets to use in query, including one or more custom datasets and one or more contextual datasets;

text-query is a *freeform* text query that contains embedded references to attributes(names) from both custom and contextual datasets, in no particular order and possibly with misspellings and embedded "nearness" operators, such as "*x* near *y*"; and

geometric-scope contains optional restrictions of scope to specific spatial regions.

The query result set is a ranked list of custom dataset entity references (Entity Identifiers or EIDs) that match the query. Along with results in the result set is additional contextual information, in particular, the geometric region and the list of contextual entity EIDs, whose intersection defines this region.

While the "freeform text query" can in principal be arbitrarily complex in scope, including queries such as "restaurants outside the Washington DC beltway", in this paper we focus on a useful and precise definition.

The freeform query can contain names or attributes from both the custom datasets and contextual datasets, in no particular order and possibly with misspellings and/or partial information and even possibly conflicting information (like the specification of a valid but wrongly specified postcode). There can be ambiguity whether a particular word or phrase refers to entities in the custom datasets versus the contextual datasets. For example, consider the query "Seattle's best coffee in Spokane." The word "Seattle" in this context could be part of the name of a custom dataset entity, or it could be the contextual City of Seattle.

Any system processing a custom local search query is expected to:

- Identify (map) which words and phrases correspond to specific names or attributes from both custom and contextual datasets.
- Extract a geometric Region of Interest from the contextual terms.
- Lookup and return entities from the custom datasets that match the attributes extracted from the queries and that fall within the Region of Interest. Entities should only be returned for the custom datasets that are specified in the query.
- In performing these operations, geometric restrictions specified are expected to be honored.

This process is illustrated in Figure 3, with an illustrative query taken from Table 1, row 4 (see also the hypothetical mashup in Figure 1.) Due to the ambiguity of interpretation, the expectation is that the system will return a ranked list of interpretations with associated set of custom dataset entities.

3.2 Challenges

There are multiple challenges to implementing a system for custom local search with the query capabilities described in Section 3.1:

Robust parsing with disambiguation – the text query contains terms from different datasets, possibly with misspellings; there could be genuine ambiguity as to which entity-name or textual attribute a word or phrase maps to. For example, it is hard to disambiguate which terms come from which datasets for the following queries: "St. paul St temples", "St Paul Church, Temple St", and "Marymoor park jogging trails".

Managing multiple datasets – it is challenging to provide a scalable solution that provides multiple custom search experiences that can leverage a large amount of shared

Tasks

1. Map query subsequences to either specific entities in contextual datasets or attributes from custom datasets (arrows in the example at right show contextual mappings; "water fountains" is mapped to a custom dataset attribute.)
2. Identify kernel Region of Interest by intersecting entities from contextual datasets (small red circle in the example.)
3. Expand Region of Interest by boundary width specified in the geographic-scope parameter of the query (dashed red circle in the example.)
4. Retrieve EIDs for custom dataset entities with matching attributes and that fall within the expanded region of interest (dark green dots in the example; the light gray dots are entities with matching attributes but which fall outside the region of interest.)

Example

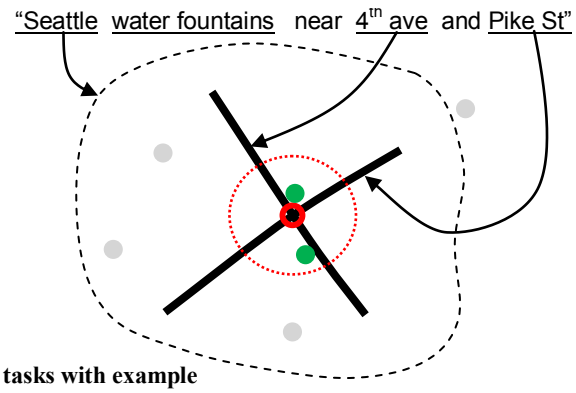

Figure 3: Query processing tasks with example

data in the form of contextual datasets, while enabling independence and flexibility in update schedule for the custom datasets.

Ranking results – Especially in the presence of ambiguity in interpreting the query, it is important to provide a reasonable ranking of the search results.

The next section presents our approach and algorithms to tackling these challenges. These are a basis for a prototype system we have built that can robustly handle custom local search queries over 45 independent custom datasets, containing an average of over 2500 entities each, along with a contextual dataset that contains over 250,000 entities providing detailed street-level vector data for the Greater Seattle area.

4. ALGORITHMS

We solve the Custom Local Search problem in three phases:

Query interpretation – in this first phase, a hybrid algorithm extracts entity names and attributes from the freeform query while simultaneously computing an approximate Region of Interest (illustrated in Figure 3). Due to errors and ambiguities in the query, multiple interpretations of the query may be generated in this phase.

Refinement & entity lookup – in this phase, specific sets of entities that make up the Regions of Interest are identified. This information is used to construct precise Regions of Interest, taking user-supplied, per-query geometric parameters into account. The precise Regions of Interest, along with textual attributes extracted from the query (in the previous phase), are then used to query a spatial database to retrieve entities from the custom datasets.

Result ranking – in this phase, multiple possible interpretations of the query are ranked, taking into account various factors including the textual similarity between terms in the query with those of the matched entities, as well factors computed from the specific entities that make up the result set.

All pre-computed indexes are maintained separately for each dataset, in order that individual custom datasets can be managed independently. We now examine each phase in turn.

4.1 Query Interpretation

The output of the first phase is a set of Partial Interpretations. Each Partial Interpretation (PI) includes a set of mappings from subsequences in the query to names or categories in specific datasets. The PI also contains an *approximate* Region of Interest, which we call the Focus of the PI. The "Partial" in Partial Interpretation stems from the fact that unlike the Interpretations introduced in [6], which map query subsequences to specific *entities,* PIs map query subsequences to *sets* of entities that share a particular name or attribute (the next section shows how PIs are converted to Interpretations.)

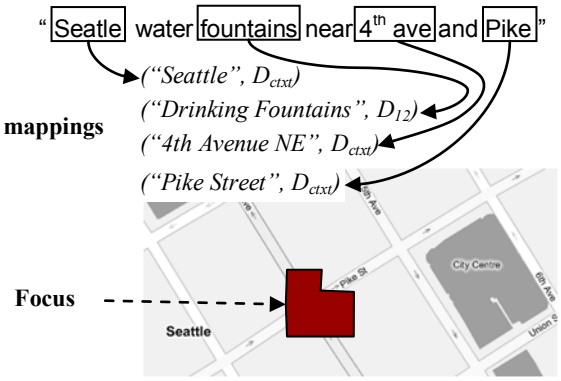

Figure 4: A Partial Interpretation (PI)

Figure 4 illustrates a particular Partial Interpretation for the query "Seatle water fountains near 4th ave and Pike." Note that the mappings may often contain inexact text matches. In the example PI, "Seatle" (presumably misspelled) maps to the name "Seattle", in contextual dataset D_{ctxt}, while "fountain" maps to the name "drinking fountain" in a particular custom dataset D_{12}. Some terms in the query may remain unmapped, such as "water" (in the latter case, "water fountain" was not a pre-defined category, while "drinking fountain" is a category). Note that the Focus contains the intersection of 4th Ave NE and Pike Street in Seattle.

Computing Partial Interpretations is challenging, especially in the custom local search domain, because it can be hard (and inherently ambiguous) to determine which terms in the query correspond to which datasets. This is especially the case when the same words or phrases occur in multiple datasets (for example "Temple" and "Temple St"), and when there are many entities in the query. It is therefore not surprising, that existing local search providers are somewhat restrictive in the form of queries they handle. Furthermore, when handling multiple custom datasets, it is quite infeasible to employ techniques that rely on hardcoded rules that reference specific keywords and attributes.

Figure 5 presents our algorithm, CLS-TEXSPACE, for computing Partial Interpretations. This is a version of the TEXSPACE algorithm [12], enhanced to support Custom Local Search. The CLS-TEXSPACE algorithm takes as input the query Q, a custom dataset D_{cust}, and a contextual dataset D_{ctxt}. Q consists of a sequence of tokens, ($q_1, q_2, q_3, \ldots, q_n$). The algorithm also takes an optional initial spatial region, F; if specified, the search will be restricted to this region. The return value is a list of Partial Interpretations. Each PI consists of a list of mappings and a final Focus, as illustrated in Figure 4. The algorithm can be easily extended to consider a *list* of custom and contextual datasets.

The hallmark of CLS-TEXSPACE is that it uses *spatial* processing, instead of text-based parsing techniques, to determine viable PIs. The algorithm does not make a priori decisions as to which subsequences are mapped to names/categories from which data sets, but rather explores many possibilities, using spatial intersection operations to prune non-viable combinations. This approach has been shown to be very effective in structured and unstructured address geocoding [12], and can also handle the added ambiguity introduced by cross-lingual matching [6].

CLS-TEXSPACE work as follows. Note that query subsequence q_{i-j} ($i \leq j$) represents a list of contiguous tokens, ($q_i, q_{i+1}, q_{i+2}, \ldots, q_j$), of query Q. The algorithm first constructs two lists of "Match Candidates" (MCs), one from the custom dataset D_{cust} and the second from the contextual dataset, D_{ctxt} (Figure 5, line 1). Each MC, (q_{i-j}, *name, D*), maps query subsequence q_{i-j} to name/attribute *name* in spatial dataset *D*. MCs are *candidate* Partial Interpretation mappings, such as the mappings illustrated in Figure 4. The two MC lists are obtained by looking up a *precomputed*, per-database fuzzy text indexes, *FI*(*D*). Any suitably robust fuzzy/approximate text lookup technology may be used, such as the Fuzzy Lookup system described in [2], which

CLS-TEXSPACE(Q, D_{cust}, D_{ctxt}, F)
// Return a ranked list of Partial Interpretations of query Q given
// custom repository $D_{cust}$and contextual repository D_{ctxt}
// Optional focus F provides an initial spatial scope. See text for explanation.
1. **for each** $n(n+1)/2$ subsequences q_{i-j} of Q
   a. MCL_{cust} ← set of match candidates (MCs) resulting from looking up q_{i-j} in precomputed fuzzy text index $FI(D_{cust})$
   b. MCL_{ctxt} ← set of MCs resulting from looking up q_{i-j} in precomputed fuzzy text index $FI(D_{ctxt})$
2. **for each** MC *CustMC* in MCL_{cust}
   a. *AnchoredMCList* ← **Filter**(*CustMC*, MCL_{ctxt}, F)
   b. **return TEXSPACE**(*AnchoredMCList*, F, {*CustMC*})

TEXSPACE(*MCList*, *Focus, WorkingPI*)
// Recursively grow partially constructed PI
// *WorkingPI*, and narrowing *Focus,* by trying
// successive elements of *MCList* in turn.
3. **if** *MCList* is empty
   then return {*WorkingPI, Focus*}
4. *NextMC* ← head of *MCList*
5. *NewWorkingPI* ← append *NextMC* to end of *WorkingPI*
6. *newFocus* ← geometric intersection of *Focus* and entities in *NextMC*
7. *CompatibleMCs* ← **Filter**(*NextMC, MCList, newFocus*)
8. // Perform recursive depth-first exploration
   return
   TEXSPACE (*CompatibleMCs*, *NewFocus, NewWorkingPI*))
   ∪ **TEXSPACE** (*MCList - NextMC*, *Focus, WorkingPI*)

Filter (*AnchorMC, MCList, Focus*)
// Return the MCs in *MCList* that are textually non-overlapping
// and spatially coherent with *AnchorMC*.
9. (q_{i-j}, *name, D*) ← *AnchorMC // Extract AnchorMC fields*
10. *filteredList* ← {}
11. **for each** (q_{k-l}, $name_y$) in *MCList*
    a. **if** (q_{k-l} does not textually overlap q_{i-j})
       ∧ (*Entities*($name_y$, *D*) are spatially coherent with *Focus*) // See text for explanation.
       then add (q_{k-l}, $name_y$, *D*) to *filteredList*
12. **return** *filteredList*

Figure 5: CLS-TEXSPACE Algorithm

we use in our implementation. The algorithm then calls TEXSPACE multiple times, each time seeding a Partial Interpretation with a Match Candidate, *CustMC*, taken from the list of custom dataset MCs (Figure 5, line 2). For each call to TEXSPACE, only the list of contextual MCs that are *compatible* (i.e., they spatially overlap and are derived from non-overlapping subsequences of query *Q*) with *CustMC* are presented. The call to the Filter function achieves this. This initial seeding and filtering step is crucial. It ensures that all PIs generated have exactly one mapping from the custom dataset and *no potential mappings are explored unless they are in the spatial neighborhood of entities from the specified custom dataset corresponding to the seeding Match Candidate*. This is a powerful source for robustness and disambiguation, while also eliminating fruitless exploration. The TEXSPACE algorithm is as described in [7]. The recursive algorithm's input includes a geometric region or scope, called *Focus*. The scope is successively narrowed as the working Partial Interpretation grows.

The Filter function needs to compute the boolean expression "*Entities*($name_y$, *D*) are spatially coherent with *Focus*" (Figure 5, line 11a) where *Entities*(*S*, *name, D*) is the (potentially large) set of entities in dataset *D* that share the name or attribute *name*. This may seem computationally expensive to be performing repeatedly, since this is asking the question: does the scope represented by *Focus* spatially overlap any of the entities with name $name_y$ (after first growing their respective boundaries by a predetermined amount). However, as explained in [7], by using precomputed linear quadtrees [4] to represent the spatial extents of the collection of spatial entities, as well as *Focus*, we can efficiently compute this Boolean expression. Since most entities in fact do not intersect with each other (a corollary of the "first law of geography" [13]), the filtering steps in Figure 5, lines 2 and 7 are extremely effective at pruning the exploration. In fact the cost of executing TEXSPACE is less than 10% of overall cost, which is dominated by the initial fuzzy-text lookup operation. Our Evaluation section has more details on performance.

Figure 6 illustrates the CLS-TEXSPACE algorithm converging on one particular Partial Interpretation for the query "restaurants near 148th ave and Northup," one of many combinations of Match Candidates explored by TEXSPACE. Figure 6 (A) shows the Footprint (a linear quadtree) of Match Candidate "Restaurants" taken from the custom dataset, overlaid on a map. Figure 6 (B) shows the Footprints of two Match Candidates, "148th Ave NE" and "Northrup Way," from the contextual dataset that overlap with the footprint of "Restaurants." Note that a single Footprint can contain multiple discontinuous regions, such as that of "148th Ave NE." Finally, Figure 6 (C) shows the Final Focus, the region of intersection of the two Footprints in Figure 6 (B).

4.2 Refinement and Entity Lookup

Recall that CLS-TEXSPACE produces a list of Partial Interpretations, each consisting of a set of mappings and a final focus ().Each mapping refers to a *set* of entities in a database that share a particular matched name or attribute. The next step is to identify the actual entities that make up what we call a fully-refined Interpretation. This is not trivial to compute. In a fully refined Interpretation, each mapping references a specific entity; and there could be multiple subsets of entities in cases that more than one entity corresponding to a name. Figure 7 illustrates how multiple sets of Interpretations can derive from a single Partial Interpretation, in this case one whose mappings contain the following names from the contextual dataset: "Greater Seattle", "I-90", and "Main St". In this illustrative example, there happen to be two entities named "Main St" (E_3 and E_4), and one each for "Greater Seattle" and "I-90" (E_1 and E_2, respectively). The Final Focus is a discontiguous region that covers two regions where three entities with the aforementioned names overlap. This Partial Interpretation is associated with two Interpretations, I_1 and I_2, corresponding to the two distinct entity triples that overlap: (E_2, E_3, E_1) and (E_2, E_4, E_1). In fact, the precise point of intersection (the Region of Interest as defined in Figure 3, before boundary expansion) are determined uniquely by the intersections of E_2 with E_3, and of E_2 with E_4. For this reason, tuples (E_2, E_3) and (E_2, E_4), which represent two distinct intersections of pairs of entities

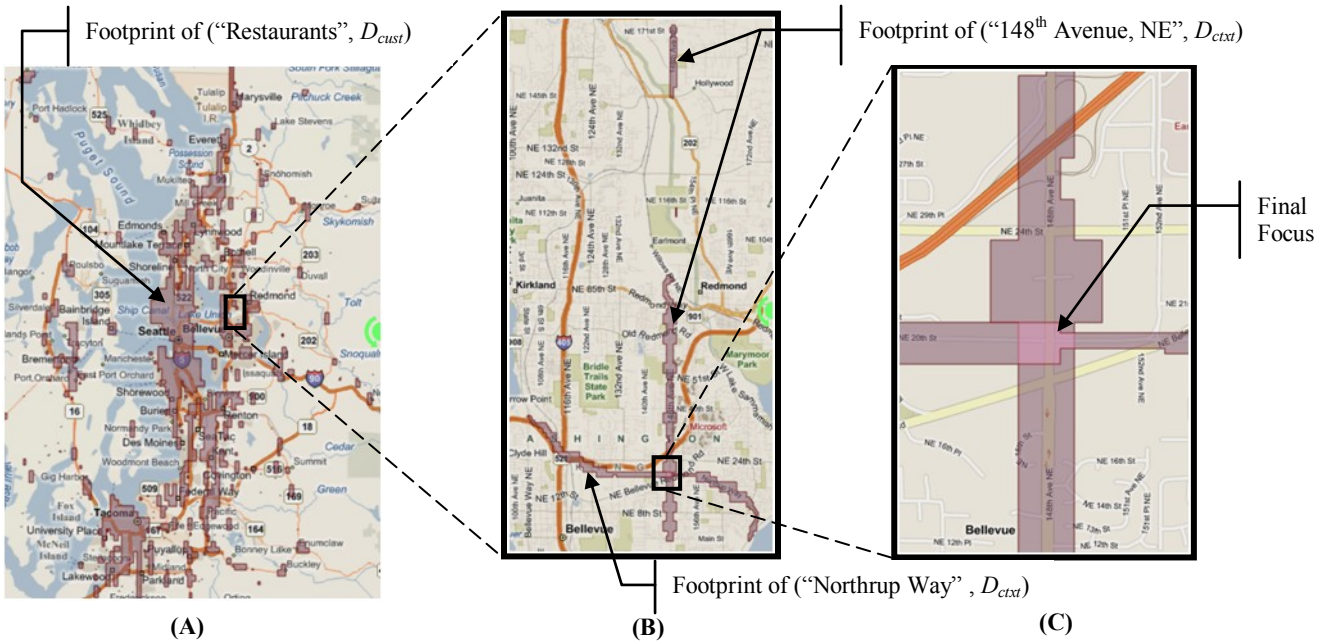

Figure 6: Illustrating CLS-TEXSPACE

with names "I-90" and "1st Main", are called out as "Intersectors." Other entities associated with an Interpretation, if any, are known as "Containers" (in this example, there is a single container entity, E_1, with name "Greater Seattle.") By convention, we represent the entities in an Interpretation as $((E_1, E_2, E_3, \ldots, E_n), E'_1, E'_2, E'_3, \ldots, E'_m)$, where the E_i s are Intersectors and the E'_j s are Containers. Note that these definitions apply even when entities are compound entities, each potentially composed of multiple primitive shapes (points, lines, and polygons with and without holes).

As illustrated in Figure 7, computing Interpretations from Partial Interpretations is not trivial in the general case. Algorithm FIND-INTERPRETATIONS (Figure 8) performs this refinement operation. First (Figure 8, line 1), the algorithm builds lists of

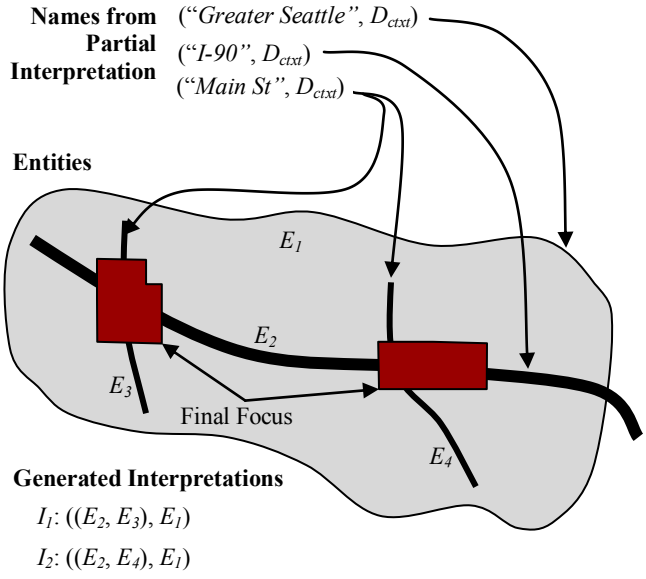

Figure 7: From Partial Interpretations to Interpretations

entities, one list for each name in the Partial Interpretation. Each list is constructed by looking up all entities with that name that come from its corresponding dataset and which are spatially within the Focus of the PI. The expression *SI*(*Name*, *D*, *Focus*) in Figure 8, line 1.b computes these lists. Precomputed spatial indexes *SI* (one maintained per dataset *D*) are used in this step. In most cases, it gets only one entity for each name because the Focus returned by CLS-TEXSPACE is quite discriminating. These lists are accumulated into *EntitiesPerNameList* (which is therefore a list of list of entities.) The algorithm then accumulates (potentially multiple) subsets of entities from the lists of entities produced by previous step. Each subset contains entities that mutually overlap. The overlap detection is done using spatial footprint of individual entityids. This functionality is implemented by recursive function RefineInterpretations in Figure 8. At each recursion depth level, it fans out, picking up individual entities (from successive elements of *EntitiesPerNameList*) that lie within a successively refined Focus. Each accumulated interpretation has exactly one entity for each name. It is interesting to note that RefineInterpration has a similar structure to TEXSPACE, but works at the level of entities and operates more surgically because TEXSPACE has previously shortlisted combinations of Names that are known to overlap. Classification of entities as Intersectors and Containers, not shown in the listing in Figure 8, is simply a matter of computing the minimal subset of entities per Interpretation that produce the refined Focus.

4.2.1 Custom Dataset Entity Lookup

The overall objective of Custom Local Search is to return the set of custom dataset entities that fall within the Region of Interest (ROI) for the query, as defined in Section 3.1. There are two options for computing the ROI:

1. A precise ROI can be computed for an Interpretation by intersecting the Intersectors, and then growing the resultant shape by the query-specific boundary width.

2. An approximate ROI can be computed by growing the Final Focus by the per-query boundary width. This ROI will be guaranteed to contain the precise ROI.

We use the latter technique in our implementation.There are two options for looking up custom dataset entities given the ROI:

1. FIND-INTERPRETATIONS *itself* will generate candidate entity IDs from all datasets, including custom datasets involved in the query. This list must then be filtered, pruning out those that do not fall within the ROI.
2. For some custom datasets, it may be preferred *not* to attempt to determine the custom dataset entities directly using FIND-INTERPRETATIONS, but rather use the ROI to lookup a separate spatial database (one maintained per custom dataset), to retrieve the entity IDs. One reason for choosing this method is that the custom datasets may be extremely dense. A second reason is to retrieve instantaneous results in case the dataset is changing rapidly.

FIND-INTERPRETATIONS (*PI*)
// Return a list of Interpretations given Partial Interpretation
// *PI*
1. // For each query subsequence-to-name match in *PI*,
   // prepare a list of entities that fall within the *PI* Focus
   // by looking up precomputed spatial index *SI* for the
   // corresponding dataset *D*. Accumulate these lists in
   // *EntitiesPerNameList*.
   a. *EntitiesPerNameList* ← {}
   b. **for each** (*Name*, *D*) in *PI*
      add *SI*(*Name*, *D*, *Focus*) to *EntitiesPerNameList*
2. // Accumulate Interpretations by refinement
   return RefineInterpretation (*EntitiesPerNameList*, *Focus*, {})

RefineInterpretation(*EntitiesList*, *CurrentFocus*, *WorkingInterpretation*)
// Recursively refine a Partial Interpretation by enumerating
// entities at each level and picking entities that
// overlap with the successively refined Focus.
3. **if** *EntitiesList* is empty
   then return {*Working Interpretation* }
4. *EntitiesAtThisLevel* ← head of *EntitiesList*
5. // Recursively accumulate Interpretations for each entity
   // at this level
   Interpretations ← *{}*
   for each entity *E* in *EntitiesAtThisLevel*
      a. *RefinedFocus* ← geometric intersection of *CurrentFocus* and *E*
      b. **if** *RefinedFocus* is non empty
         then // *E* lies within the refined Focus
            NewWorkingInterpretation ← append *E* to *WorkingInterpretation*
            Interpretations ← *Interpretations* ∪ **RefineInterpretation**(tail of *EntitiesList*, *RefinedFocus*, *NewWorkingInterpretation*)
6. **return** *Interpretations*

Figure 8: Algorithm FIND-INTERPRETATIONS

Our implementation uses FIND-INTERPRETATIONS to obtain entities.

4.3 Ranking

Our system as evaluated in this paper uses a basic set of heuristics that take into account the following information:

- Weighted edit-distance score between query subsequences and the matched entity names.
- Number of matched names.
- Fraction of the query that remains unmatched.

More sophisticated mechanisms are being considered as part of ongoing research.

5. RESULTS AND EVALUATION

We have built a fully functional prototype Custom Local Search system based on the algorithms described in the previous section. The system indexes a contextual dataset that contains over 250,000 entities providing detailed street-level vector data for the Greater Seattle area. Our system also indexes over 45 independent custom datasets. These datasets contain between 1000 and 8000 entities each, averaging over 2500 entities per dataset. The custom datasets were constructed from subsets of real data – commercial Yellow Pages data from the Greater Seattle area, comprising in aggregate over 60,000 distinct entities with over 2000 categories. One of the major benefits of using Yellow Pages data in our custom datasets is that it enables comparison of our system against commercial local search providers. It is worth noting here that it is very easy to add a custom dataset to our system. In fact the dataset indexes can be created (pre-computed) on a machine and then deployed on a different machine.

In order to demonstrate the utility, accuracy and robustness of our approach, and to analyze overall performance, we have built an evaluation framework that allows us to measure our system on a variety of queries. The next subsection describes our process for constructing challenging but realistic local search test queries, as well as reference result sets for each query.

Section 5.2 presents the precision and recall graphs for our results run on this set of queries. Section 5.3 compares our results with other commercially available custom search providers. We show that our system provides precision and recall figures that match or exceed those of commercial search providers, while serving queries over multiple custom datasets. Finally Section 0 demonstrates the performance advantage gained from loading multiple custom indexes that share a large common contextual dataset.

Figure 9 is a screenshot of the GUI front end of our CLS system, displaying results for the query "northup way coffe[sic] places close to 148th ave ne." Note that the system is correctly displaying coffee shops near the intersection of Northup Way and 148th Avenue NE as top-ranked results.

5.1 Test Query Sets and Reference Results

We constructed our local search test data in the following three-step process:

1. First we created a set of pure location queries as follows. We synthesized 90 location queries by selecting combinations of overlapping features from the underlying vector data. Since these descriptions were synthesized from specific spatial entities, the reference (golden) location is exactly known. In addition, we selected 85 well-formed address queries that

were collected from users. For each such location query, we determined the reference geographic region of interest by majority consensus among multiple commercial location search engines, augmented by manual examination of online maps.

2. Next we constructed test local search queries by randomly adding custom data attributes such as names of businesses (e.g., "Sears") or categories of interest (e.g., "Chinese Restaurants") to the above location queries. We inserted 1 to 6 categories per location query making up total of 426 custom local search queries. These queries were marked as "clean" queries, i.e., queries containing no errors. We also generated an additional 426 queries by adding varying amounts of errors, using techniques described in [12]. These include spelling variations, reordering of terms and the introduction of conflict and ambiguity.
3. For each of the test local search queries, we generated the reference (golden) result set by firing queries to a spatial database that picked the entities within the Region of Interest, according to the semantics defined in Section 3.1.

Table 2 shows some typical queries, along with the number of entities that make up the reference result set for each.

5.2 Precision and Recall Performance

Precision and recall are defined in terms of r, the number of correct entities among the first n results returned for a query, and t, the total number of correct entities in the reference (golden) result for this query. Precision is defined as the ratio r / n, and recall is defined as the ratio r / t [10]. A high precision value indicates that most results retrieved were useful, and a high recall value indicates that most of the useful results were retrieved. We are able to precisely compute precision and recall numbers because we know golden result set for each query.

Figure 10 presents average precision and recall results of our system executing the test query set, for queries with and without introduced errors. The x-axis is the value n/t, i.e., the number of results considered, normalized by the number of expected golden entities for that query. Thus at $x = 1.0$, the number of results considered is equal to the number of golden entities, and at $x = 2.0$, we consider twice as many results as there are golden entities. This normalization allows meaningful averaging even if there is wide variation in the number of golden entities per query. Note that the value of precision and recall at x=1.0 is equivalent to R-precision as described in [10].

The graph shows that our system performs quite well, obtaining a recall value of 0.75 at x=1.0, growing to 0.85 (85% of golden set entities found) at x=2.0, for clean queries. The graph also shows that our system has high precision values for x<1.0, indicating that the golden entities are returned at top ranks. This has also been our experience anecdotally from using our system.

Figure 10 also presents precision and recall values for queries with moderate errors added (such as misspellings and the introduction of extraneous terms, as explained in Section 5.1). The graphs show that our system exhibits relatively small degradation with the introduction of errors, showing that the CLS-TEXSPACE algorithm is robust to errors in queries.

5.3 Comparison with Commercial Services

Using bona fide Yellow Pages data to construct custom datasets has enabled us to also evaluate our system with respect to well established commercial local search providers. We evaluated our system by comparing our system with two popular online location search services (Google® and Microsoft®). To account for the

Table 2: Sample Queries

Query	Entities
Apartments around sea-tac airport	12
Find me insurance agents at S Judkins St and 13th ave intersection, seattle	22
Tukwila city, Burgers along S 136th St	4
Pacific Restaurants, 121st St S, 8th Ave Ct S,Pacific Lutheran University,Pierce	6
Japanese Restaurants near University of Washington campus, Union Bay Pl, 45th St,Seattle	4
Pizza places at 14th St and Mullen St,Tacoma	3
Cocktail Lounges near Mill Creek, at 132nd St SE & McCollum County Park intersection, Snohomish	1
Met Life at S Thistle St,42nd Ave S intersection,Seattle	1

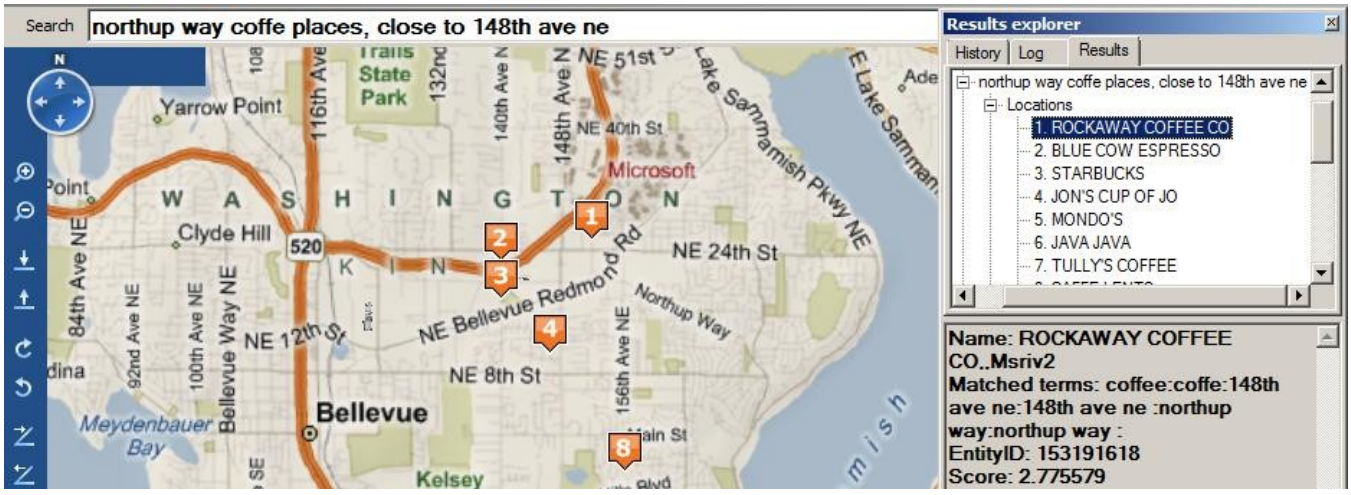

Figure 9: Screenshot from CLS prototype

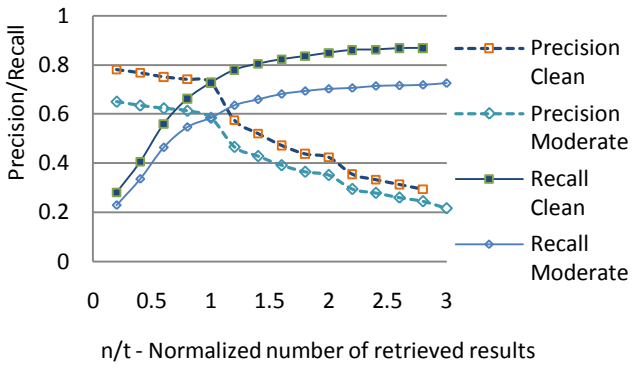

Figure 10: Precision and recall graph of our system

fact that the commercial services index data from a region much wider than ours, we modified the queries sent to the commercial search engines by appending each test query with "Washington, USA" before firing it to the commercial services. We also had to make modifications to several queries (such as inserting "AND" between street entities) to make them acceptable to one of the commercial services. Even with these modifications there were some queries from our test dataset that did not obtain any results from the commercial service (though they do work fine with ours) – these are eliminated in our quantitative comparisons below. Moreover, out of the many available services, we picked the service for which the underlying contextual as well as custom data was available to us.

Figure 11 presents precision-recall graphs for the set of simple queries for our service and the chosen commercial service. The graphs show that our precision and recall numbers exceed those of the commercial service. One of the prime contributors to this disparity is our system's more robust parsing capability, in particular when dealing with combinations of intersecting contextual entities. Another factor is our system's more precise notion of Region of Interest.

5.4 Effect of Pre-index Boundary Width

One of the parameters of our system is the boundary width expansion applied to the shape of each custom dataset entity during index building. This in effect results in expansion of the footprint of each entity. This expansion will enable our system to support queries with operators such as "near", "within x-miles of". Naturally larger pre-index boundary width would enable us to support larger distance queries. To analyze the effect of this boundary width expansion, we created a series of custom indexes with increasing width and observed the results for a set of test queries. We observed that Precision and Recall values degraded considerably with increased boundary width. This can be attributed to the fact that with increased width, larger number of entities are likely to intersect given focus and hence larger number of entities per spatial name are extracted at Step 1b in Figure 8. These are likely to be crowded with "noise" entities resulting in reduced Precision and Recall values. Ideally if all the intersecting entities were extracted in Step 1b, perhaps we would see that eventually Recall would reach value of 1.0, however it would result in a serious hit in query-processing time (as we observed in our experiments), therefore in practice we extract only a limited number of entities giving lower Recall values for larger boundary width. This shows that the effect of spatial constraints goes down with the increasing pre-index boundary width.

5.5 Performance

To evaluate the performance of our system and the scalability of the CLS-TEXSPACE algorithm, we evaluated our system under constant load while supporting up to 46 independent custom datasets, sharing a common contextual dataset – the same Greater Seattle contextual dataset used in the precision/recall evaluation. The custom datasets, comprising Yellow Pages data, contained between 1000 and 8000 entities each, averaging over 2500 entities per dataset.

Figure 12 shows the memory consumption of our system, when under full load, as a function of the number of custom datasets being simultaneously queried. To measure memory consumption for $x=N$, we initialize our system with N datasets, and fire queries at random to all N datasets. The first run starts with ~100MB of memory footprint as it loads the (large) context dataset. From that point, the memory usage gracefully goes up as additional datasets are loaded and queried in subsequent runs. This graph shows that our system handles increasing number of custom datasets with linear increase in memory usage. Note that had the contextual dataset not been shared among the custom datasets, but instead merged (duplicated) with each individual dataset, the memory requirements to support 46 independent custom local search experiences would be in excess of 4GB, a 10-fold increase. This demonstrates the performance benefit of using a shared contextual database.Queries take on average about 370ms per query, and the operation is CPU bound even under full load, on a 3GHz uniprocessor machine with 2GB RAM. Figure 13 shows the CPU usage time as the number of custom datasets loaded in each run increases. The relatively flat curve is expected because the processing is CPU bound (80% of the time spent in fuzzy text lookup), and the complexity of CLS-TEXSPACE does not depend on the total number of databases being served.

6. RELATED WORK

There has been recent work on spatio-textual databases. Notably, the STEWARD spatio-textual search engine [9] is a system for automatically geo tagging unstructured text documents such as news articles, and supporting querying and visualization over this document set. The system employs a number of techniques including named entity extraction and mechanisms for disambiguation that leverage the presence of geographically nearby place-names present in the document. In [3], Chen et al. present multiple geographic query processing techniques, while in , Zhou et al. present a hybrid index structure for combined text and spatial data. Our work is complementary in that it focuses on precise query semantics for text queries that have a complex structure that can include multiple intersecting entities with potential misspellings, and which include embedded local search

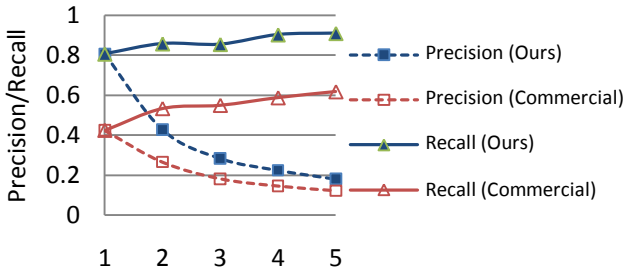

Figure 11: Comparison with commercial service

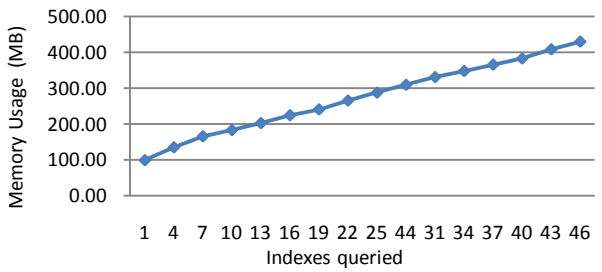

Figure 12: Physical Memory Usage of CLS

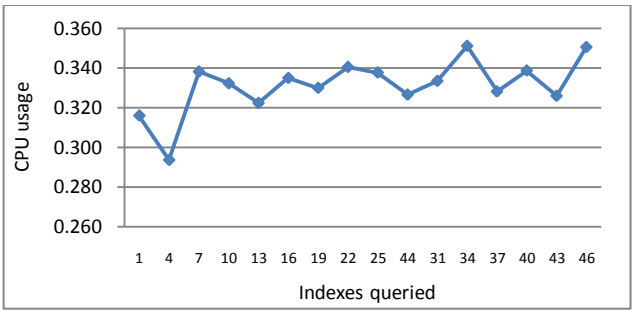

Figure 13: CPU usage per query (Seconds)

categories – queries of this nature are not handled by these systems, to the best of our knowledge. Furthermore, the focus of our work is on providing multiple custom local search experiences, each retrieving results from distinct sets of custom spatial datasets, something that none of these systems address. Search services, such as Google™ Custom Search [5], provide custom search functionality for web search. Third party sites can use this service to build a search experience where both the scope of the search and the presentation of the results can be customized. However this does not extend to custom *local* search. Google Custom Maps allows users to upload custom maps. However, as far as we can determine, search functionality is limited to full text search over this custom data.

Commercial spatial databases such as Microsoft SQL Server™ [11] support text search subject to spatial constraints. They do not have the sophisticated parsing capabilities of our system, but rather a complementary set of lower level primitives that can be leveraged in building a Custom Local Search system based on the algorithms described in this paper.

Several systems exist for geocoding postal addresses. The objective of these systems is to compute a geographic *location* from input text. These are typically rule based systems specific to a particular country or address format, such as the Australian address geocoding system described in [1]. The TEXSPACE algorithm, introduced in [12], does not require rules, and supports robust geocoding of both *structured* (postal) addresses, as well as informal, or *unstructured* descriptions of locations. None of these systems, however, support "yellow pages" style local search – where the objective is to return a list of specific spatial entities that fall within a region of interest, and furthermore, they do not have a concept of multiple custom datasets. However, TEXSPACE does function as an important building block for robust parsing of the query string, as we explain in Section 4.1. XL-QUERY [6] is a cross-lingual location search algorithm based on TEXSPACE. XL-QUERY computes the spatial scope of the query as accurately and as robustly as possible when the query is in a script that is *different* from that of the names of the underlying geographic entities in the geocoding system. There is the potential, presently unexplored, of using XL-QUERY to extend custom local search to support crosslingual location search queries.

7. CONCLUSIONS

The Custom Local Search algorithms and system we have described and evaluated in this paper provide a way for users and organizations to have the same or better level of freeform text search over their custom spatial datasets, as that provided by commercial search providers. We have found these techniques to be effective in practice and to scale well, supporting 100s of custom datasets which share common contextual data.There are opportunities to further research that builds on this work in several ways, for example, embedding more complex spatial operators into the free form text queries, and exploring index structures that are tailored to very frequently updated datasets.

8. REFERENCES

[1] Christen, P., Churches, T. and Willmore, A. A probabilistic geocoding system based on a national address file. In *Proc. of the 3rd Australasian Data Mining Conference*, 2004.

[2] Chaudhary, S., Ganjam, K., Ganti V. and Motwani R. Robust and efficient fuzzy match for online data cleaning. In *Proc. ACM SIGMOD*, 2003.

[3] Chen, Y.Y., Suel, T. and Markowetz, A. Efficient query processing in geographic web search engines. In *Proc.* ACM SIGMOD, 2006.

[4] Gargantini, I. An effective way to represent quadtrees. In *Comm. of the ACM*, 1982.

[5] Google Custom Search. See http://www.google.com/coop/cse/

[6] Joshi, T., Joy, J., Kellner, T., Khurana, U., A Kumaran and Sengar, V. 2008. Crosslingual location search. In *Proc. ACM SIGIR*, 2008.

[7] Joshi, T., Joy, J., and Sengar, V. Robust Location Search. Technical Report MSR-TR-2008-41, Microsoft Research, 2008.

[8] Laycock, J. "Still Together" – Search engines and Internet Yellow Pages. Online article, see http://www.searchengineguide.com/senews/005928.html.

[9] Lieberman, M. D., Samet, H, Sankaranarayanan, J., and Sperling, J. STEWARD: architecture of a spatio-textual search engine. In *Proc. ACM GIS*, 2007.

[10] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[11] Microsoft SQL Server. See http://www.microsoft.com/sqlserver/.

[12] Sengar, V., Joshi, T., Joy, J., Prakash, S., and Toyama, K. Robust location search from text queries. In *Proc. ACM GIS 2007*

[13] Tobler, W. R.. The first law of geography. See http://en.wikipedia.org/wiki/First_law_of_geography.

[14] Zhou, Y., Xie, X., Wang, C., Gong, Y. and Ma, W. Y. Hybrid index structures for location based web search. In *Proc.CIKM 2005*