Towards an Extensible Expert-Sourcing Platform

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Christopher Jonathan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Mohamed F. Mokbel

May, 2019

Acknowledgements

There are many people that have earned my gratitude for their contribution to my time in graduate school. Firstly, I am extremely thankful to my parents, Sandford Jonathan and Sutini Setiadi, for their support and motivation throughout the years of my graduate school career. Without their support, it would be very difficult to be able to reach the current stage of my graduate studies and career.

I would like to extend much gratitude and appreciation to my doctoral adviser Prof. Mohamed F. Mokbel for all of the accomplishment that I have achieved in the last five years of my Ph.D. career. In addition to his support and guidance, he has been also an excellent role model for research, advising, teaching, and career.

I would like to thank my DMLab colleagues for our collaboration on exciting research ideas that has greatly enriched my research and mentoring experience. In particular, Ahmed Eldawy, Amr Magdy, Ibrahim Sabek, Louai Alarabi, Rami Alghamdi, Harshada Chavan, Mashaal Musleh, Saif Al-Harthi, and Kwang Woo Nam.

I am also extremely thankful to both of my younger brothers, Albert Jonathan and Alvin Jonathan, for their support as well as for being my close friends in pursuing our Ph.D. career together. Finally, I would like to thank my close social circle that have supported me throughout my Ph.D. journey, especially Jessica Tulus.

Dedication

To my precious family.

Abstract

In recent years, general purpose crowdsourcing platforms, e.g., Amazon Mechanical Turk, Figure Eight, and ChinaCrowds, have been gaining a lot of popularity due to their capability in solving tasks that are still difficult for machines or computers to solve, e.g., labeling data, sorting images, computing skyline over noisy data, and sentiment analysis. Unfortunately, current crowdsourcing platforms are lacking a very important feature that is desired by many of the recent crowdsourcing applications, namely, recruiting workers that are expert at a given task. Being able to recruit expert workers will allow those applications to not only achieve a more accurate results but also higher quality results than recruiting general crowd for the applications. We call such crowdsourcing process as *expert-sourcing*, i.e., outsourcing tasks to experts. Without having any platforms to support them, developers of each expert-sourcing application needs to build the whole crowdsourcing system stack from scratch while, in fact, those systems share many common components with each other.

This thesis proposes Luna; the first extensible expert-sourcing platform. To instantiate a new expert-sourcing application out of Luna, one only needs to provide a few simple plug-ins that will be integrated with the core components of Luna to provide the expert-sourcing platform for the new application. This is possible due to the fact that Luna is able to identify the components that can be shared among many expert-sourcing applications and the components that need to be tailored for a specific application. In this thesis, we show the extensibility of Luna by instantiating six different expert-sourcing applications that are currently not well supported by the general purpose crowdsourcing platforms. Experimental evaluation with real crowdsourcing deployment as well as by using real dataset shows that Luna is able to achieve not only more accurate but also better quality results than existing general purpose crowdsourcing platforms in supporting expert-sourcing applications. Lastly, we also provide a more specialized expert-sourcing platform for image geotagging application that is initially deemed unfit to be solved by crowdsourcing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recently, crowdsourcing has been gaining a lot of popularity due to its capability in solving various computer-hard tasks, e.g., labeling data [1, 2], sorting images [3], and sentiment analysis [4]. The popularity of crowdsourcing can be seen by the existence of several commercial general purpose crowdsourcing platforms, including Amazon Mechanical Turk (AMT) [5], Figure Eight [6], and ChinaCrowds [7]. In such platforms, a requester submits a task, alongside its deadline and budget, to be solved by workers, i.e., the crowd. Then, when the task has reached its deadline or has been solved by the crowd, the requester will retrieve the workers' results from the platform. Unfortunately, current AMT-like platforms are lacking a very important feature, desired by many of the recent crowdsourcing applications, i.e., recruiting workers that are expert at a given task. Current crowdsourcing platforms basically post all of the available tasks, and interested workers can just do it without ensuring that those workers are the best workers to solve these tasks. While this may be acceptable for some simple tasks, this may not be the case for more complicated, and very common, tasks. For example, a translation task may need workers who are expert at the source and the output languages, a spatial task (e.g., ride-sharing or image-geotagging) may need workers who are expert at a certain geographic area, an image-labeling task may need workers with the expertise of the image contents, while a surveying task about a certain topic may need workers with matching expertise about the surveyed topic. We call such crowdsourcing process as *expert-sourcing*, i.e., outsourcing tasks to experts.

The popularity of expert-sourcing applications, hindered by the limitations of existing crowdsourcing platforms to support them, urges both industry and academia to provide several expert-sourcing solutions where each solution is tailored to tackle a specific type of application. This includes specific solutions for data labeling [1, 2, 8], ride-sharing [9, 10], and spatial crowdsourcing [11, 12]. The main problem here is that these are all tailored solutions, in a sense that each application needs to build its own crowdsourcing platform from scratch. For example, a ride-sharing application has to build its own crowdsourcing platform as the current general purpose ones cannot serve it. Such need becomes a major roadblock in taking the full advantage of crowdsourcing to support various expert-sourcing applications.

This thesis proposes Luna: the first extensible expert-sourcing platform. Luna overcomes the limitations of existing crowdsourcing platforms to support expert-sourcing applications as: (a) Luna ensures that a task is assigned to expert workers of that task, hence ensures higher quality results. (b) Luna is an extensible platform equipped with the main infrastructures that are needed by a wide spectrum of expert-sourcing applications. To instantiate a new application out of Luna, one only needs to provide a few simple plug-ins that will be integrated with the core of Luna to provide the expert-sourcing platform for the new application. Such extensible platform would act as a gateway for various applications that can take advantage of the idea of supporting expert-sourcing without the need to build a new full-fledged system from scratch. Also, from a system point of view, the extensible approach of Luna is very appealing to its wide spectrum of applications. Whenever a new idea or data structure is developed inside Luna to boost its performance, it is only developed once in the system, then, a myriad of expert-sourcing applications are immediately supported.

The core of Luna is composed of four modules. Three of these modules are configurable modules with a plug-in function that overrides their default functionality to be more specific to the application that is instantiated out of Luna while the last module provides an interface for the administrator to configure the three modules. In particular, the first configurable module is responsible in finding a set of expert workers for a given task, where it has a plug-in function on the criteria to select those workers. For example, this criteria could be the nearest worker in the case of ride sharing or the workers who have previously labeled similar images in the case of image-labeling. The second

configurable module is responsible in finding a single task for a worker, where it has a plug-in function on the criteria to select one of the available tasks, e.g., a task that is the closest to a deadline. The third configurable module is basically a plug-in function responsible in aggregating the results of all workers to form the final answer for the task. The underlying infrastructure of Luna is composed of two fixed index structures and one configurable application-based index structure. The fixed index structures are common for any expert-sourcing applications. Meanwhile, the configurable index structure is plugged into Luna based on the application needs, to present the set of expertise of that application. The expertise can be either flat, e.g., language expertise, or hierarchical, e.g., location expertise where an expert in Minneapolis can also be considered as an expert in a parent expertise Minnesota as well as the United Sates.

To configure a new expert-sourcing application using Luna, a system administrator will only need to write a simple SQL-like statement that identifies: (a) the expertise index that is needed in the application, (b) the function used to select a set of workers for a given task from a set of already identified expert workers, (c) the function used to identify a single task for a given worker out of a set of tasks with matching expertise to the worker, and (d) the function used to aggregate a set of results to a final result. Without defining any of these configurations, Luna will handle expert-sourcing applications similar to how a general purpose crowdsourcing platform handles those applications. In this thesis, we show Luna extensibility by instantiating six expert-sourcing applications out of Luna, namely, *domain-specific image-labeling*, *image-labeling*, *image-geotagging*, *spatial crowdsourcing marketplace*, *ride-sharing*, and *translation*.

Luna is extensively experimented with real crowdsourcing deployment as well as by using real dataset where we evaluate the results both qualitatively and quantitatively. In particular, for both domain-specific image-labeling and image-geotagging case studies, we show that by deploying both applications out of Luna, we are able to achieve not only more accurate but also better quality results in comparison to when deploying those applications on top of a general purpose crowdsourcing platform. As for the ride-sharing application, we show that Luna is able to provide an efficient ride-sharing application that is currently unsupported by any of the general purpose crowdsourcing platforms.

1.1 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 details the system overview as well as the main functionality of Luna. The extensibility of Luna is discussed in Chapter 3, Chapter 4, and Chapter 5 where we deploy six case studies out of Luna by using the four simple plug-ins that are provided by Luna. In particular, Chapter 3 discusses two case studies, namely, *domain-specific image-labeling* and *image-labeling*; Chapter 4 discusses three case studies, namely, *spatial crowdsourcing marketplace*, *ride-sharing*, and *image-geotagging*; while Chapter 5 discusses the *translation* case study. Figure 1.1 gives the summary of the configuration of Luna that we use for each case study. Then, Chapter 6 discusses a more specialized crowdsourcing framework to tackle image-geotagging problem where we provide multiple optimization to increase the result accuracy and the overall performance of the process. Lastly, Chapter 7 gives the related work to our work and Chapter 8 concludes the thesis.

Application	Expertise-Index	WorkerRank	TaskRank	Aggregate
Domain-specific Image-Labeling	WordNet	FIFO	expertise-first	expertise-aware-majority
Image-Labeling	iterates over Domain-specific Image-Labeling			
Spatial Crowdsourcing Marketplace	Spatial	submit range query	request range query	all
Ride-Sharing	Spatial	nearest-worker	nearest-task	all
Area-based Image-Geotagging	Spatial	FIFO	expertise-first	distance-aware-majority
Image-Geotagging	iterates over Area-based Image-Geotagging			
Translation	Language-Language	FIFO	deadline-first	translation-scoring

Figure 1.1: Summary of Luna Case Studies

Chapter 2

Luna System Overview

Figure 2.1 gives the system architecture of Luna as an extensible expert-sourcing platform. Luna is composed of four modules, Application Creation, Request Task, Submit Task, and Result Evaluation, as well as three index structures, Worker Index, Task Index and Expertise Index. Modules and index structures depicted in non-bold lines (Application Creation module, Worker Index, and Task Index) are oblivious of the underlying application of Luna. Meanwhile, modules and index structures depicted in bold lines are responsible on the extensibility of Luna as they are configurable based on the application that will be built using Luna.

2.1 Luna Users

There are three kinds of users who would deal with Luna namely, *requester*, *worker*, and *administrator*. The interaction between these users and Luna goes as follows:

- **Requester.** The requester is someone who has a task, and would like to use a Luna-based system to recruit experts who would complete the task. The requester submits a task to Luna, which would include: (a) the budget that the requester is willing to pay to get the task done, (b) a deadline where the task needs to be done before, and (c) the expertise that is needed to solve the task. First, Luna inserts the task information into the Task Index. Then, Luna involves its configurable Submit Task module to find available expert workers according to its

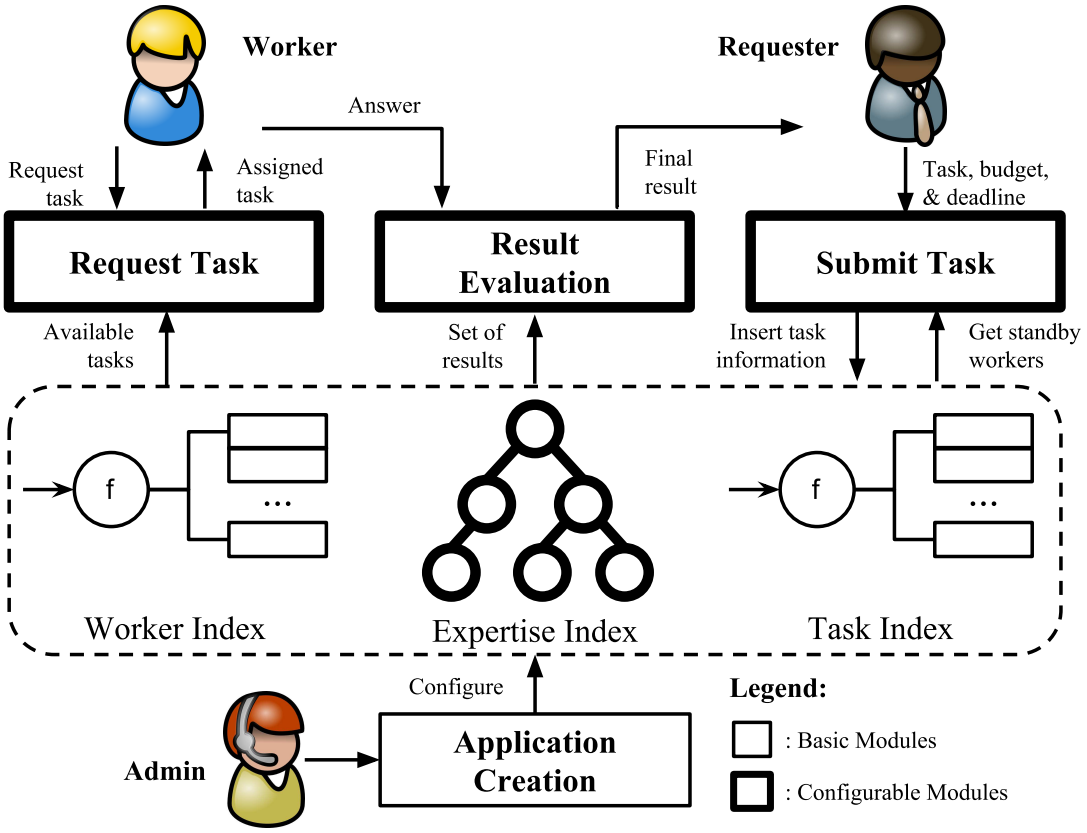


Figure 2.1: Luna System Architecture

configured criteria in that module. If there are enough available workers, the task will be completed by these workers, and the result will be sent to the configurable Result Evaluation module to come up with the final result to be sent back to the requester. If there are not enough available expert workers in the system, the task will be inserted into the Expertise Index, waiting for new workers that are expert at the task to show up. Once the task budget is all spent, or the deadline is reached, whatever results that are available will be sent to the Result Evaluation module. We will further discuss this work-flow in Chapter 2.3.

- **Worker.** The worker is someone with certain expertise who is willing to perform tasks that match the worker's expertise and get compensated for. When the worker is ready to do a task, Luna inserts the worker's information into the Worker Index

and involves its configurable Request Task module to find a matching task. If there is no matching task at the moment, the worker is inserted into the Expertise Index as standby waiting for a suitable task to do. We will further discuss this work-flow in Chapter 2.4.

- **Administrator.** The administrator is the one who instantiates various instances of expert-sourcing applications that are based on Luna. This is done by feeding the Create Application module with the information that is needed to customize the three configurable modules and the configurable index structure. We will discuss this further in Chapter 2.5.

2.2 Luna Data Structures

This section discusses the three index structures in Luna, namely, Task Index, Worker Index, and Expertise Index. Figure 2.2 gives an example of the three index structures.

Task Index

The Task Index is a hash table to store all *standby* and *active* tasks in Luna. A task is *standby* if it is still waiting for workers with matching expertise to solve it. A task is *active* when it is currently assigned to all the workers it needs, however, still waiting for the result to be concluded. An entry in the Task Index is composed of seven attributes: (*TaskID*, *Budget*, *Deadline*, *Expertise*, *Active Workers*, *Result Set*, *Task Properties*). The *Budget* and *Deadline* attributes are set by the requester. Here, we consider the *Budget* as the number of workers needed to solve this task. It could also be a monetary value that is split among the workers who are taking the task. The *Expertise* is the expertise domain needed by the worker to solve the task, e.g., a certain language or a certain location that the worker must be aware of. The *Expertise* could be either set by the requester, derived from another expert-sourcing of Luna, or derived from the nature of the task and its surrounding context. While the details of finding out the expertise of a task is beyond the scope of this thesis, we later provide two case studies to show on how to automatically derive a task expertise with Luna in Chapter 3.6 and Chapter 4.6. Without loss of generality, we assume that a task would only require one set of expertise.

Extending Luna to deal with tasks of multiple expertise is straightforward. The *Active Worker* is a list that is initially empty and then populated by workers who are currently taking the task. The *Result Set* is initially empty and then populated by the results obtained from each assigned worker in the form of (*WorkerID*, *Answer*). Once all results are obtained, this set is used to conclude the final result for the task. Finally, the *Task Properties* attribute is used to store optional additional information that is specific to a certain task in some instances of Luna.

Worker Index

The Worker Index is another hash table to store all *standby* workers, i.e., workers who are waiting for a task to be assigned to them, and *active* workers, i.e., workers who are currently working on a task. An entry in the Worker Index is composed of three attributes: (*WorkerID*, *Expertise List*, *Worker Properties*). Setting the workers' expertise can be either self reported from the worker or guided by a work-flow, e.g., where the worker needs to answer few questions or select among options to narrow down the workers' expertise, in a way similar to Quora [13]. The *Worker Properties* attribute is used to store additional information that may be specific to the worker, e.g., the worker's individual performance.

Expertise Index

In Luna, an expertise is defined as a set of *non-overlapping* domains, related to the expertise topic. For example, a *Language* expertise may be a set of known languages, where a worker may be an expert in one or more languages and a task may need a certain language expertise, e.g., translation tasks. An expertise may also be hierarchical. For example, a worker may be an expert in animals, but more specifically in mammals, and even more specifically in dogs. In that case, an *Animal* expertise may have a hierarchy of domains, where a worker who is expert at a certain species is also considered to be an expert at all of its upper hierarchy. Another example of a hierarchical expertise is the location expertise, where an expert at the Minneapolis area is also considered an expert at the Minnesota state, as well as USA. A task may need a certain expertise at a certain hierarchy.

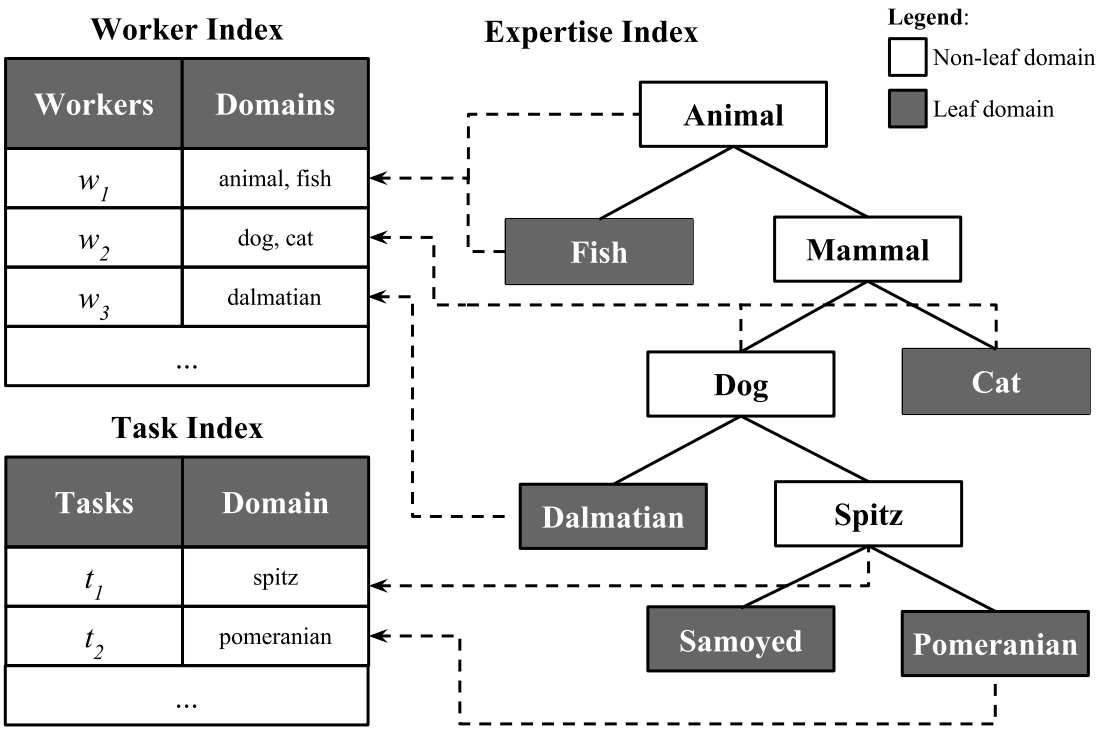


Figure 2.2: Example of Luna Indexes

The right side of Figure 2.2 depicts part of the Expertise Index in Luna for the *Animal* expertise. It is represented as a tree structure, where the root node represents the general expertise, i.e, "Animal". Nodes in the same level represent non-overlapping domains. Each node can have zero or more children. If the expertise is non-hierarchical, e.g., *Language* expertise, we create a dummy root node that acts as the parent node of all the non-overlapping nodes. Each node maintains two lists: (a) a list of Task IDs, representing the set of standby tasks that are waiting for workers with the node expertise domain, and (b) a list of Worker IDs, representing the set of standby workers with the node expertise domain and are looking for tasks matching their expertise. At any point of time, at least one of these two lists in each node should be empty. For an incoming task, it will be inserted in the task list, only if there are no available workers in the worker list to be assigned to it. Similarly, an incoming worker will be inserted in the worker list, only if there are no available tasks to take from the task list.

The Expertise Index is a *configurable* index in Luna, where each instantiation of an expert-sourcing application from Luna would either use one of the default available expertise index structures or upload its own new index structure in a specific format that is compatible with Luna. Currently, Luna has two default expertise index structures, namely (1) *Spatial*, a hierarchical spatial expertise that indexes the whole world in grids, and (2) *WordNet*, a hierarchical expertise index based on the WordNet synsets [14], i.e., a large lexical database of English language. Here, we use the nouns as the domains of the *WordNet* expertise index and the ISA-relationship between nouns as the parent-child relationship between the domains of the expertise. Meanwhile, examples of expertise index structures that can be uploaded include Yahoo Answers categories [15], which is a non-hierarchical set of general topics including Sports, Politics, and Health; or Wikipedia portals [16], which is a hierarchical set of main topics in Wikipedia including Culture, Health, and History. The details of uploading a new expertise index in Luna will be discussed further in Chapter 2.5.

2.3 Submitting a Task to Luna

This section discusses the process of submitting a new task to Luna by the requester. Algorithm 1 gives the pseudo code of this process. The input to the algorithm is the task T, with its budget B and expertise E. For simplicity, and without loss of generality, we are omitting the task deadline from this procedure. Handling the task deadline will be simply dealt through a background process that wakes up on the task expiration to conclude the result. The algorithm calls two configurable functions, namely `WorkerRank` and `Aggregate`, in Lines 6 and 21 in Algorithm 1, respectively. These two configurable functions are extensible plug-ins to the algorithm to be furnished by any expert-sourcing applications that will use Luna. The strength of our Luna platform lies in the fact that, except for the two configurable functions, Algorithm 1 is the same for any expert-sourcing applications. Researchers, developers, and practitioners, who would like to build a new expert-sourcing application, e.g., ride sharing, image labeling, and translation, do not really have to bother developing things from scratch. Instead, they will only need to furnish the two configurable functions in Algorithm 1 for their application. Later, a third configurable function will be discussed in Chapter 2.4.

Algorithm 1 starts by inserting the incoming task T with its budget B and expertise E into the Task Index, with an initial empty set for its list of workers and results. Generally speaking, Algorithm 1 is composed of three phases, as follows:

- **Phase I: Assigning workers.** The main objective of this phase is to find at most B workers to work on the task T. This is done by first locating the node N_E in the Expertise Index that corresponds to the expertise E that is required to solve the task T. Locating N_E is done by either simply scanning the Expertise Index from the root node until the node is located, or by using another hash index with the expertise as a key to locate the node directly. If the expertise node N_E has more than B workers waiting for a task to work on, i.e., *standby* workers, then Luna has to make its decision about which B workers to select to work on the task. Since the selection criteria would be application-based, this is where the configurable function `WorkerRank` needs to be plugged in Luna per application. `WorkerRank` takes the task T, budget B and list of workers that are already expert at this task, as input and returns a set of B workers that are more suitable to do the task per the logic furnished in this configurable function. For example, an image-labeling application may select the top-B workers who have done the most tasks within the task expertise, while, a ride-sharing application may select the nearest worker ($B = 1$) to the pickup location of the task. By default, and if not furnished by the application, Luna uses a FIFO approach as its `WorkerRank` to select the top-B workers who have been on standby the longest in order to maintain the fairness between workers. In case that the expertise node N_E has less than B workers, then, regardless of the application that is built with Luna, we just assign T to all workers in N_E without the need to use the `WorkerRank`.

- **Phase II: Workers working on the task.** This phase is the same for any expert-sourcing applications that are using Luna. In this phase, the list of workers identified in Phase I, i.e., $T.Workers$, will work on the task T. Once a worker w, $w \epsilon\ T.Workers$, finishes the task, then (a) the outcome is appended to the *Result Set* of the task, i.e., $T.Results$, (b) the worker w is removed from *every* node N_w in the Expertise Index where w is in the N_w's worker list, and (c) the worker w is removed from the Worker Index. If w needs to do another task, the worker

will request this again from Luna, with the procedure that will be discussed in Chapter 2.4.

- **Phase III: Concluding task submission.** If the number of workers who worked on the task T in Phase II is less than the required number of workers B, then, the task is not finished yet. In that case, and regardless of the application that uses Luna, we: (a) update the value of B to be the remaining number of workers to finish the task, and (b) add T to the task list of the node N_E in the Expertise Index that has the required expertise of T, waiting for more workers to show up and work on the task through the procedure in Chapter 2.4 in the future. In case that already B workers have worked on the task in Phase II, then the task is considered finished, and we need to aggregate the results received from all workers together, i.e., $T.Results$. This is done through the configurable function `Aggregate`, per application. For example, an image-labeling application may select the majority label for the image while a translation application may select the translation that is the most similar to professionally-produced translations. By default, and if not furnished by the application, Luna outputs all worker's answers, similar to what Amazon Mechanical Turk is doing. After solving the task, we remove the task from the Expertise Index and the Task Index, then return the result to the requester.

2.4 Requesting a Task from Luna

This section discusses the process of a worker requesting to start working on a task in Luna. Algorithm 2 gives the pseudocode of this process. The input to the algorithm is the worker w, with the workers' list of expertise E. The algorithm calls two configurable functions, namely `TaskRank` and `Aggregate`, in Lines 17 and 23 in Algorithm 2, respectively. As mentioned in Chapter 2.3, these two configurable functions are extensible plug-ins to be furnished by the expert-sourcing application that will use Luna. The algorithm starts by inserting the worker w into the Worker Index, then it is composed of two phases, as follows:

- **Phase I: Finding candidate tasks.** This phase is the same for any expert-sourcing applications that are using Luna, where the objective is to find all candidate tasks that worker w is eligible to do, i.e., the worker w has the expertise to perform on any of these tasks. Recall that: (a) a worker may have more than one expertise, and (b) for the case of a hierarchical expertise in the Expertise Index, if a worker is an expert at a certain expertise node, then the worker is also considered expert at all of its ancestor expertise nodes. Given this, we iterate over all expertise in E, and for each of them, we visit its corresponding node in the Expertise Index, along with and all its ancestor nodes. For each node that we visit, we append the list of tasks in this node to our list of candidate tasks. If it ends up that there are no candidate tasks in any of these nodes, then the worker will not be doing any work here. Instead, the worker will be added to the worker list of every node that we have visited as a *standby* worker. The worker will be picked up later when a new task arrives with a need of any of the worker's expertise through the procedure discussed in Algorithm 1 in Chapter 2.3.

- **Phase II: Solving a task, if found.** This phase is executed only if there is at least one candidate task that is identified in Phase I. The first thing that this phase would do is to select one of the candidate tasks and assign it to the worker w. Since the selection criteria would be application-based, this is where the configurable function `TaskRank` needs to be plugged in Luna per application. `TaskRank` takes the worker w and the list of candidate tasks as input, and returns one task T, based on the logic furnished in this configurable function. For example, a translation application may select the task with earliest deadline while a ride-sharing application may select the nearest task. By default, and if not furnished by the application, Luna uses an *expertise-first* ranking function where it selects the task that has the deepest height in the Expertise Index. If there is more than one task with the deepest height, we select the one with the earliest deadline among them. Once the task T is selected, then the worker w is assigned to it. Once completed, we reduce the task budget by one in the Task Index as well as remove the worker w from the Worker Index and the *Active Worker* list of the task in the Task Index. If all of the task budget is consumed, then the task is considered finished, and we aggregate the results that are received from all workers together.

This is done through the configurable function `Aggregate`, per application, which is the same function discussed in in Algorithm 1 in Chapter 2.3. After a task is solved, we remove it from the Expertise Index and the Task Index, then return the result to the requester.

2.5 Creating Luna-based Expert-Sourcing Application

Luna is equipped with an SQL-like administrator interface to instantiate a new expert-sourcing application out of Luna. We will study six of such applications as our case studies in the following three chapters. To create a Luna-based expert-sourcing application, the system administrator will need to issue the following SQL-like clause:

CREATE-APPLICATION *Application-Name*
EXPERTISE-INDEX `Expertise`
`[OPTIONAL]` **TASK-SUBMISSION** `WorkerRank`
`[OPTIONAL]` **TASK-REQUEST** `TaskRank`
`[OPTIONAL]` **RESULT-EVALUATION** `Aggregate`

With this clause, the system administrator will need to furnish the three modules, namely, `WorkerRank`, `TaskRank`, and `Aggregate`, and the Expertise Index structure `Expertise` for the application, described briefly as follows:

- `WorkerRank`. This is the plug-in function that is used in Line 6 of Algorithm 1 to select B workers for a given task, where B is the task budget, from a set of candidate workers who are all expert at the task. The logic of the function is application-dependent, and hence needs to be furnished by the system administrator who is creating the application. By default, if not furnished by the application, Luna uses its default `WorkerRank` function that employs a FIFO approach to select the top-B standby workers who have been waiting the longest for a task to work on. The main goal of this approach is to maintain fairness among workers.

- `TaskRank`. This is the plug-in function that is used in Line 17 of Algorithm 2 to select one task for a given worker among a set of candidate tasks where all of those

tasks have matching expertise with the worker, i.e., the worker is expert at each of them. As it is the case of the `WorkerRank` function, the logic of the `TaskRank` function is application-dependent. By default, if not furnished by the application, Luna uses its default `TaskRank` function that employs an *expertise-first* ranking function to select the task that has the deepest height in the Expertise Index that is used by the application. If there is more than one task with the deepest height or the expertise index is a non-hierarchical one, we select the one with earliest deadline among them.

- `Aggregate.` This is the plug-in function that is used in both Line 21 of Algorithm 1 and Line 23 of Algorithm 2 to calculate the final answer of a given task out of multiple workers' answers for that task. The input to this function is the list of results returned from a set of workers who have done the task. The output of the function, along with its logic, are application-dependent. By default, and if not furnished by the application, Luna outputs all worker's answers without any aggregation in a way similar to what Amazon Mechanical Turk does.

- `Expertise.` This is the Expertise Index, depicted in Figure 2.2, and discussed in Chapter 2.2. Luna is already equipped with the following two predefined expertise index structures: (a) *WordNet*, which is a publicly available lexical database (collection of words) of the English language, where words (nouns, verbs, adjectives or adverbs) are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. The relations among synsets is a super-subordinate relation that links more general synsets like to increasingly specific ones. In Luna, we have uploaded the 117K synsets of *WordNet* in a tree index structure, considering each synset as an expertise. (2) *Spatial*, which is a hierarchical spatial grid that starts from a root node covering the whole world. The next tree level is a division of the whole world into four quadrants. Each cell is further, and recursively, divided into four quadrants, where the lowest level is covering areas within few miles. Each spatial cell is considered an expertise by itself, where a worker may be an expert in a certain geographical area, and a task may need to be solved by workers who are experts in a certain geographical area. The administrator of a new Luna-based application can either pick one of the two available predefined

expertise index structures, i.e., *WordNet* or *Spatial*, or define a new expertise index structure. Once defined, the new index structure lives inside Luna and can be reused for other applications. To define a new expertise index structure, the administrator would need to upload a file containing a list of (E, E_P) tuples where E is an expertise and E_P is the parent expertise of E. If E is a root node, then we set $E_P = E$. For example, to create the *Animal* expertise index of Figure 2.2, the administrator will provide a file that contains tuples such as ("Animal", "Animal"), ("Fish", "Animal"), ("Mammal", "Animal"), ("Dog", "Mammal"), and so on. If the index is a non-hierarchical one, then the file will contain a list of (E, E) tuples. Note that if the index does not have a single root node, Luna will automatically create a dummy root node for the index.

Algorithm 1 Task Submission Procedure

```
1: procedure SubmitTask(Task T, Budget B, Expertise E)
2:     Insert T to the Task Index with B, E
3:     T.Workers ← ∅; T.Results ← ∅
       /* Phase I: Assigning T to at most B expert workers */
4:     N_E ← node in the Expertise Index corresponds to E
5:     if |N_E.WorkerList| > B then
6:         T.Workers ← WorkerRank (T, N_E.WorkerList, B)
7:     else
8:         T.Workers ← N_E.WorkerList
9:     end if
       /* Phase II: Workers working on the task */
10:    for each worker w in T.Workers do
11:        w works on T, the outcome is appended to T.Results
12:        for each node N_w that matches an expertise of w do
13:            Remove w from N_w.WorkerList
14:        end for
15:        Remove w from the Worker Index
16:    end for
       /* Phase III: Concluding task submission */
17:    if |T.Workers | < B then
18:        T.B ← T.B - |T.Workers|
19:        Insert T in N_E.TaskList
20:    else
21:        Result ← Aggregate (T.Results)
22:        Remove T from the Task Index
23:        return Result
24:    end if
25: end procedure
```

Algorithm 2 Requesting Task Procedure

```
1: procedure RequestTask(Worker w, ExpertiseList E)
2:     Insert w and E to the Worker Index
       /* Phase I: Finding candidate tasks */
3:     CandidateTasks ← ∅
4:     for each expertise e in E do
5:         N_e ← node in the Expertise Index corresponds to e
6:         CandidateTasks ← CandidateTasks ∪ N_e.TaskList
7:         for each parent N_p node of N_e up to the root do
8:             CandidateTasks ← CandidateTasks ∪ N_p.TaskList
9:         end for
10:    end for
11:    if CandidateTasks is empty then
12:        for each node N_E in the Expertise Index that
             corresponds to a task in CandidateTasks do
13:            Add w to N_E.WorkerList
14:        end for
15:        return
16:    end if
       /* Phase II: Solving a task, if found */
17:    Task T ← TaskRank(w, CandidateTasks)
18:    Insert w to T.Workers
19:    w works on T, outcome is appended to T.Results
20:    T.B ← T.B - 1
21:    Remove w from the Worker Index and T.Workers
22:    if T.B = 0 then
23:        Result ← Aggregate (T.Results)
24:        Remove T from the Expertise Index
25:        Remove T from the Task Index
26:        return Result
27:    end if
28: end procedure
```

Chapter 3

Domain-specific Data-Labeling

This chapter discusses our first case study of an expert-sourcing application that can be instantiated out of Luna, namely *domain-specific data-labeling*. Data-labeling is a process of attaching meaningful textual information to unknown data to get the sense of the data. Labeled data enable myriad of applications to better serve our needs, e.g., in providing a more accurate web search. Data-labeling application has been one of the most popular applications that are currently solved by crowdsourcing as it is still difficult for a machine to provide a label for unknown data. This is done by asking the crowd to provide a label for a given task. However, by using Luna as the platform for a data-labeling application, each labeling task will be assigned to workers that are expert at the task domain. We call such approach as *domain-specific data-labeling* where we assign workers who are expert at the domain of the task to label the task. Such approach allows us to provide not only a more accurate label but also a higher quality label than existing approaches. Experimental evaluation based on real crowdsourcing deployment shows that Luna-based domain-specific data-labeling application is able to provide better quality labels than existing crowdsourcing-based approach.

3.1 Introduction

Data labeling is a process of attaching meaningful textual information to unknown data in order to get the sense of the data. Examples include, but are not limited to, finding out the object that is depicted by a photo, the topic of an audio or video recording,

and the topic of a news article. Labeled data enable myriad of applications to better serve our needs, e.g., in providing a more accurate web search [17], point-of-interest recommendation [18], customer-product recommendation [19], and understanding city demographics [20]. In addition to those applications, machine learning-based applications, e.g., Netflix [21] and Tinder [22], need to learn from large as well as growing labeled data in order to do their jobs.

With the importance of data labeling, multiple research efforts use artificial intelligence techniques [23–25] to label their data. Unfortunately, these approaches not only able to consistently provide an accurate label for the data, but also require a large amount of already labeled training dataset to empower their labeling capability in the first place. To address such problem, crowdsourcing has been a popular solution for data labeling where it uses humans' knowledge to provide the label. An example of a labeling task is in asking the workers to provide a label of an image that depicts a husky dog. By using crowdsourcing, any workers should easily be able to identify that the label of the image is depicting a"dog" or an "animal".

With such popularity of labeling tasks in crowdsourcing, multiple research efforts have been conducted to improve the accuracy of labeling tasks [1, 8, 26, 27]. In general, these works propose a solution to mainly ensure that the resulting label is accurate. Using the above example of labeling an image of a "husky dog", these works try to ensure that inaccurate label, such as "cat", will not be selected as the resulting label of the image. This is done by either selecting workers who previously have done well in solving similar tasks as in [8,26] or by selecting workers who have the same domain with the task's domain, e.g., location domain in [1] and a predefined small set of domains in [27]. These approaches are instances of an expert-sourcing application that is called *domain-specific data-labeling*, i.e., assigning each task to workers that are expert in the domain of the task. For example, a requester may have an image of an animal, but needs to know which animal, an image of a city in Minnesota, but needs to know which city, or an image of a car, but needs to know the brand of the car. For such images, the requester would prefer to assign such tasks of image-labeling to workers who have expertise in animal, Minnesota, or car, respectively. The initial domain of each task can be known by the requester themselves or can be retrieved from other approaches. For example, it can be retrieved by matching the keywords of a task to find previously similar tasks as

in [8, 26], by matching the geo-location information of the task to worker's location as in [1], or by linking the keyword entities of a given task to a knowledge bases as in [27].

In this chapter, we deploy a domain-specific data-labeling application out of Luna by just using the four simple plug-ins of Luna. Without loss of generality, in this thesis, we will only focus one instance of data-labeling application, i.e., image-labeling application, as there are less works have been done in this application. However, our approach can be extended to label any data types. Throughout the rest of the chapter, we devise a new ranking function for the `TaskRank` plug-in for the Request Task module of Luna, namely *ED-first* ranking function, as well as an *expertise-aware majority* aggregation function for the `Aggregate` plug-in for the Result Evaluation module of Luna. Experimental evaluation based on real crowdsourcing deployment with real workers shows that Luna-based labeling approach is able to retrieve not only more accurate labels, but also higher quality labels than existing crowdsourcing-based labeling approach.

3.2 Luna Configuration

This chapter discusses the configuration of Luna for domain specific image-labeling application. In this application, requester submits a task as image to be labeled, along with the budget (i.e., the number of required workers for the task), the task deadline, and the known domain (i.e., expertise) of the image. Then, workers with matching expertise will provide the labels for the image and returns the final label of the image to the requester. Using Luna, one can simply use the following SQL-like command to instantiate an expert-sourcing domain-specific image-labeling application:

CREATE APPLICATION *Domain-specific Image-Labeling*
EXPERTISE-INDEX *WordNet*
TASK-SUBMISSION *FIFO*
TASK-REQUEST *ED-first*
RESULT-EVALUATION *expertise-aware-majority*

In this domain-specific image-labeling problem, the plugged-in index structure and the three plug-in functions are as follows:

- **Expertise Index = *WordNet*.** We use the *WordNet* expertise index structure, discussed in Chapter 2.5, which is one of the two default data structures in Luna. Given the known domain of the image (e.g., "animal" or "car"), the *WordNet* index will be used to locate the node in the Expertise Index that matches the exact word of the domain that is given by the requester. Given the richness of the *WordNet* index (117K synsets), we will be able to cover most of the possible domain expertise for any domain-specific image-labeling tasks.

- `WorkerRank` = ***FIFO*.** We use the default FIFO approach for the `WorkerRank` function of the Submit Task module. Recall that, for a task T with a budget B and domain expertise E, the `WorkerRank` function would be only used if there are more than B workers are waiting in the *WordNet* node that matches the expertise E. With the FIFO approach, we would return the top-B standby workers in terms of longest waiting time in a node that has a domain of E in the *WordNet* index to be assigned with T.

- `TaskRank` = ***ED-first*.** In this plug-in function, we are not using the default one of Luna. Instead, devise an *ED-first* ranking function for the `TaskRank` function of the Request Task module. In contrast to the default *expertise-first* ranking function, *ED-first* ranking function would consider both the height of each task domain in the expertise index and the task deadline when assigning a worker with a task. We further discuss this ranking function in Chapter 3.3.

- `Aggregate` = ***expertise-aware-majority*.** In this plug-in function, we are not using the default one of Luna. Instead, one can define a very simple function that returns the majority label for the image. This simple function would still return better result than that of Amazon Mechanical Turk as Luna recruits expert workers rather than random workers. However, this simple functionality is still not taking much advantage of the inherent expert-sourcing in Luna. Hence, we devise an *expertise-aware-majority* aggregation function for the `Aggregate` function of the Result Evaluation module which exploits the expertise of the workers who did the task. We further discuss this aggregation function in Chapter 3.4

3.3 ED-first TaskRank Ranking Function

This chapter describes the *ED-first* ranking function that we devise for the `TaskRank` function of the Task Request module of Figure 2.1. Recall that the input of the `TaskRank` function is a worker who requests for a task to solve, alongside the worker's profile, from the Worker Index, and a set of candidate tasks that the worker is expert at, retrieved from the Expertise Index, and the goal of this module is to select a task that is the *best fit* for the worker based on the function definition. While the default *expertise-first* ranking function will select a task that is located at the deepest node of the expertise index among the candidate tasks, we believe that we should also consider the deadline of each task in order to avoid starvation of tasks that are not located at the deeper nodes of the expertise index. As a result, we devise an *ED-first* (Expertise Deadline-first) ranking function where we score the importance of a task t, namely R_t with the following equation: $R_t = \alpha \times E_t + (1 - \alpha) \times D_t$, where $0 \leq E_t \leq 1$ is the task expertise score, $0 \leq D_t \leq 1$ is the task deadline score, and $0 \leq \alpha \leq 1$ is a parameter to weight the importance of each rank with a default value of $\alpha = 0.5$. Algorithm 3 gives the high level overview of the ED-first ranking function.

- **Expertise Score.** We calculate the expertise score of a task t, E_t, based on the following equation: $E_t = H_t / H_{max}$ where H_t is the height of the task domain in the expertise index while H_{max} is the maximum height of a task domain among the candidate tasks for the worker.

- **Deadline Score.** We calculate the deadline score of a task t, D_t, based on the following equation $D_t = 1 - (D_t - D_{current})/D_{max}$ where D_t is the time-stamp for the task t deadline, $D_{current}$ is the current time-stamp, and D_{max} is the maximum D_t among the candidate tasks for the worker.

Consider the an example of having five candidate tasks for a worker that have been retrieved by the Request Task module from the Expertise Index in Table 3.1. In this example, we are using the default α value of 0.5, to weight the expertise score and deadline score equally, and we set the current time-stamp $D_{current} = 0$. With the default expertise-first ranking function, we would assign the worker with t_2 as it is located at a node that is located the deepest level of the expertise index, namely at

Algorithm 3 ED-first Ranking Function

```
1: procedure ED-FIRST(Worker w, CandidateTasks T)
2:     t_w ← ∅
3:     α ← 0.5
4:     H_max, D_max, R_max ← 0
5:     for each Task t in T do
6:         if t.height > H_max then
7:             H_max ← t.height
8:         end if
9:         if t.deadline > D_max then
10:            D_max ← t.deadline
11:        end if
12:    end for
13:    E_t, D_t, R_t ← 0
14:    for each Task t in T do
15:        E_t ← t.height/H_max
16:        D_t ← 1 − (t.deadline − D_current)/D_max
17:        R_t ← α × E_t + (1 − α) × D_t
18:        if R_t > R_max then
19:            R_max ← R_t
20:            t_w ← t
21:        end if
22:    end for
23:    return t_w
24: end procedure
```

level 5, among the five candidate tasks. However, with our ED-first ranking function, we will also consider the deadline score of each task where we will assign t_5 to the worker as it has the highest scoring among the five candidate tasks.

3.4 Expertise-Aware Majority Aggregate Function

In this chapter, we devise an *expertise-aware-majority* aggregation function for the `Aggregate` function of the Result Evaluation module which exploits the expertise of the workers who did the task. Instead of evaluating the answer of each worker equally, the main idea of this function is to trust more those workers who provide a label that is among their expertise domains. For example, consider two workers w_1 and w_2 with their

Task ID	Domain Height	Task Deadline	E_t	D_t	R_t
t_1	3	5	0.6	0	0.3
t_2	5	4	1	0.2	0.6
t_3	0	4	0	0.2	0.1
t_4	1	3	0.2	0.4	0.3
t_5	4	1	0.8	0.8	0.8

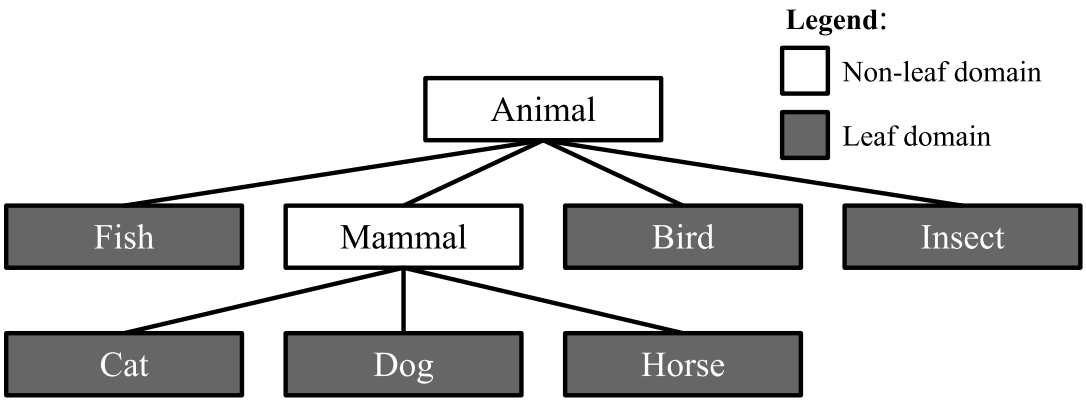
Table 3.1: ED-first Ranking Function Example

expertise domains of {"Bird", "Dog"} and {"Cat"}, respectively. While both workers may provide the same label, "Dog", we would give higher weight to w_1's answer than w_2's answer as the label is part of w_1's expertise. In doing so, we will assign a weight for each answer as the height of an *WordNet* expertise node that represents the *lowest common ancestor* node between the worker's expertise and the worker's answer. Then, the final result will be the label with highest weighted sum of the workers' results.

Figure 3.1 gives an example of a small part of the *WordNet* expertise index with height of 3 and a root node "Animal". The figure shows the answers of four workers w_1 to w_4, along with their expertise, the lowest common ancestor between the worker expertise and the answer, and the weight that is assigned to each answer. Both w_1 and w_4 have answers that are within their expertise, which means that the lowest common ancestor node is actually the answer node of each worker "Dog" and "Cat", respectively. Hence, both are assigned weight of 3, which is the height (depth) of the answer node. For w_2, the lowest common ancestor node between the answer "Dog" and the expertise "Animal" is the root node, which has height, and hence weight, of 1. For w_3, the lowest common ancestor node between the answer "Dog" and the expertise "Cat" is the "Mammal" node, which has height, and hence weight, of 2. Adding the weights of all workers, we end up having a weight of 6 to label "Dog" and weight of 3 to label "Cat", hence we return the former one as the final result.

3.5 Experiment

This experiment evaluates the *domain-specific image-labeling* application that is instantiated out of Luna. Since this application requires the requester to provide the domain of the image as a broad knowledge of the requester for each task/image, we only focus on



Worker	Worker's Expertise	Answer	Lowest Common Ancestor	Weight
w_1	Bird, Dog	Dog	Dog	3
w_2	Animal	Dog	Animal	1
w_3	Cat	Dog	Mammal	2
w_4	Fish, Cat	Cat	Cat	3

Figure 3.1: Expertise-Aware-Majority Example

a small part of the *WordNet* index structure, depicted in Figure 3.2, which includes the set of expertise domains that we are using in our experiments. In particular, we will label images with expertise domains of the index leaf nodes, i.e., the 13 gray-colored nodes in Figure 3.2. We compare our Luna-based approach against a general deployment of crowdsourced image-labeling in Amazon Mechanical Turk (AMT) where we randomly assign ten workers for each image and use majority voting to conclude the resulting label of each image. In this experiment, the performance of the Luna-based and the AMT-based approaches is compared qualitatively by looking at the output labels that are resulted from both approaches to check on their qualities. For both approaches, we use a budget of 10 workers to label the image in each domain expertise of Figure 3.2. Our prototype and experimental implementation run in Java 1.8.0-151 on a machine

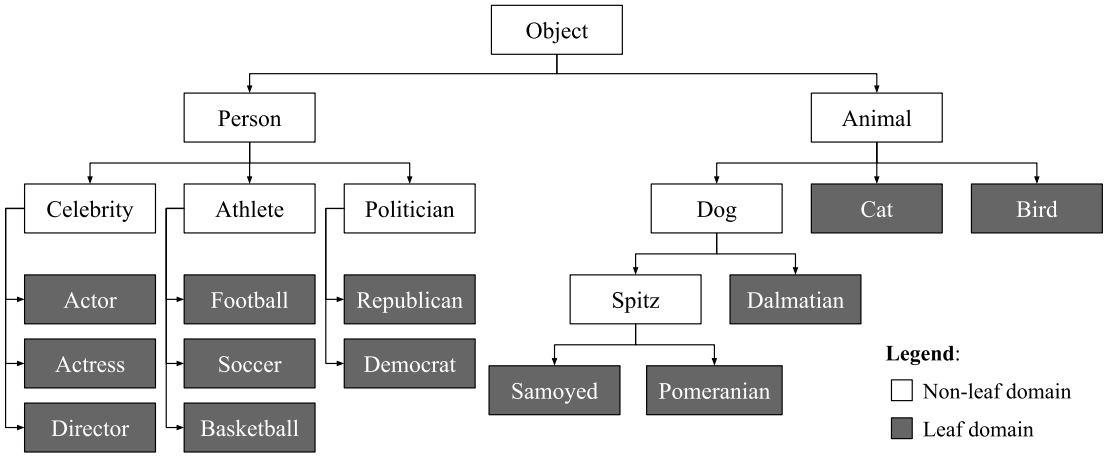


Figure 3.2: Image-Labeling Expertise

with Intel Quad Core i7-4790 3.6Ghz, two threads per core, and 32GB of RAM running 64-bit Ubuntu 16.04.

Recruiting Workers in Luna

As Luna is still a new platform, we do not have enough user base yet. Hence, we populate the Worker Index of Luna with Amazon Mechanical Turk (AMT) real workers, along with their expertise that they define themselves in a process guided by Luna. We do so by publishing a task in AMT which contains a link to our Luna platform. When a worker goes to the link, we profile the worker against our Expertise Index (e.g., *WordNet*) in a way similar to how Quora [13] profiles its new users. Once done, we insert the worker and the expertise to the Workers Index. Then, Luna does its own operations based on the requested application.

In particular, in this experiment, we used a set of 200 real workers populated in Luna from AMT workers. To ensure that we have the accurate expertise of the workers, we present the tree in Figure 3.2 to each of these 200 workers, asking the worker to mark the worker's expertise. Then, to ensure that the workers were serious in marking their expertise, we evaluate the their expertise with the *golden task* approach [28]. In particular, for each expertise that a worker selects, we give a test image that we already know the answer of and ask the worker whether the test image accurately depicts the

Table 3.2: Domain-specific Image-Labeling Results

Image	Domain	Luna	AMT
Matthew Perry	Actor	Matthew Perry	Tennis
Patty Jenkins	Director	Director	Actress
Stefon Diggs	Football	Running Back	Athlete
Paul Pogba	Soccer	Soccer Player	Artist
Kobe Bryant	Basketball	Kobe Bryant	Person
Ben Carson	Republican	Republican	Person
Elizabeth Warren	Democrat	Elizabeth Warren	Lady
Samoyed	Samoyed	Samoyed	Dog

label. If correct, we consider this is a confirmed expertise for the worker, otherwise, we remove this expertise from the worker profile.

Experimental Results

Table 3.2 gives a subset of the output labels for images that receive different labels in Luna and AMT. The table gives the ground truth of the image content, its known domain for the requester, and the output labels that we receive from both Luna and AMT. It is clear that Luna is able to achieve way more accurate and higher quality results than AMT. For example, consider the image that depicts "Elizabeth Warren", a Democratic party member in the United States. By using Luna, we are able to identify that the label of the image is "Elizabeth Warren" while using AMT gives the label "lady". While both labels are technically correct, it is clear that the label from Luna is of higher quality, and thus definitely preferred. The main reason here is the Luna was able to recruit workers who are expert at the Democratic party while AMT just give the picture to some random workers. Another example includes labeling an image that depicts "Kobe Bryant", a popular NBA player, where Luna is able to give a label of "Kobe Bryant" while AMT gives a label "Person". In other cases, the difference between Luna and AMT is even more severe, where AMT gives totally inaccurate results. For example, when labeling an image of "Matthew Parry", an actor, Luna is able to identify that the image contains a label of "Matthew Perry" while AMT gives a completely inaccurate label "Tennis". Another example is when labeling an image of "Paul Pogba", a soccer player, where AMT gives a label of "Artist" for the image.

Algorithm 4 Image-Labeling

```
1: procedure ImageLabeling(Tasks T, Budget B)
2:     E ← root node of the Expertise Index
3:     b ← B/ height of the Expertise Index
4:     while B > 0 do
5:         if E is a leaf node in the Expertise Index then
6:             b ← B
7:         end if
8:         E ← Luna Domain-specific Image-Labeling(T, b, E)
9:         B ← B − b
10:    end while
11:    return E
12: end procedure
```

3.6 Image Labeling

In the *image-labeling* problem, the requester submits an image along with the budget (i.e., number of required workers) to ask workers to label the submitted image. Unlike the case of domain-specific image-labeling problem, where the requester has a broad idea about the image content, in this case, the requester has no idea whatsoever about the image content. This makes it more challenging as we do not know which expertise is needed to solve the image-labeling problem. One way to deal with this is to treat this problem exactly as the domain-specific image-labeling problem, with the expertise E set to the root node of the *WordNet* expertise index. However, this ends up in a very poor quality labeling as we really cannot recruit expert workers here because all workers are considered experts in the root node.

What we want to do is to recruit those workers who are expert at the image content even though we do not have any idea about the image content. Luckily enough, a good administrator can use Luna to solve this dilemma and truly recruit those workers that are expert at the "unknown" image content. The main idea is to use an *iterative* approach that gradually understands the expertise of the image. This is done by splitting the labeling process (and its budget) into multiple iterations of domain-specific image-labeling tasks where the goal of each iteration is to retrieve a more specialized label for the image based on the previous iteration's result.

In particular, in the first iteration, we call the domain-specific image-labeling procedure with only a small part of the budget and the general expertise of the root node as the inputs. The result is returned as a specific label L_1 based on the `Aggregate` function used in Chapter 3.2. Then, in a second iteration, we again call the domain-specific image-labeling procedure with another part of the budget, however, now with the expertise is set to L_1 (the label that is returned from the previous iteration). This ensures that the workers in the second iteration are more expert at the image content, as they are all have the expertise in L_1. The result of the second iteration would be another more specific label L_2. In a third iteration, we again call another domain-specific image-labeling procedure with part of the budget and the expertise is now set to L_2. We continue doing so until we either reach a leaf expertise or the task budget is all consumed. The bottom line here is that in each iteration, we know more about the image content, and hence are able to recruit more expert workers.

Algorithm 4 gives the pseudo code that the Luna administrator would need to write to use domain-specific image-labeling to solve the more general image-labeling problem without having any prior domain knowledge. The algorithm only takes the task T and budget B as input, and returns an expertise node E from the *WordNet* expertise index as the image label. The algorithm starts by assigning the task expertise E to the root node of the *WordNet* index and limits the task budget only to b. b can be calculated by dividing the main task budget B by the number of levels in the index structure. Then, iteratively, the algorithm calls the domain-specific image-labeling with T, b, and E. After each iteration, E is updated to a more specific label for the image and the main task budget B is adjusted. We continue doing so until E is one of the index leaf nodes or B is all consumed.

For example, consider labeling an image of a dog. At the start, since we have no idea about the image content, we set the root node as the domain. Since most people can identify a dog, the first iteration will return the label "Dog". While existing crowdsourcing approach will conclude and return the label "Dog"; with Luna, we can go further to retrieve the species of the dog. This is done by running another domain-specific image-labeling task where we set the domain of the task to be "Dog" to assign the workers that are expert at "Dog" to label the image. By doing so, we will receive a more detailed label for the image, e.g., the species of the dog.

3.7 Related Work

This chapter discusses the existing work on data labeling applications in crowdsourcing. Data labeling has been one of the most widely studied crowdsourcing applications due to their popularity among other crowdsourcing problems. In particular, to solve the data labeling problem, many research efforts are focused on providing a more accurate labels for objects such as images and entities [1, 2, 4, 8, 29–33].

Most works [4, 8, 29–33] are focused on increasing the quality of the resulting labels by investigating the answer inference problem and task assignment problem of crowdsourcing. This is done by ensuring that they are assigning workers who have previously provided a label for a similar objects and performed well. Unfortunately, these existing works only work with a small set of answers or with a small set of expertise domains. For example, they limit the resulting labels into two labels, i.e., "true" or "false", for entity resolution problem. This is because their techniques rely on the outcome label in evaluating the worker's performance which will be used to assign workers with new tasks in the future. Meanwhile, [1, 2] consider the spatial distance between the workers and the tasks in order to achieve a better label.

In this chapter, we showed that we can configure Luna to solve the data labeling problem with any kinds of expertise without limiting the number of the output labels. In addition, we can also develop any of the existing techniques above by simply modifying the four plugins of Luna.

Chapter 4

Spatial Crowdsourcing Marketplace

This chapter discusses another case study of an expert-sourcing application that can be instantiated out of Luna, namely *spatial crowdsourcing marketplace*. In contrast to an existing crowdsourcing marketplace, e.g., Amazon Mechanical Turk or Figure Eight, where a task can be assigned to any workers randomly, each task in this application requires workers that are located at the spatial proximity of the task in order to complete the task. We call such tasks as *spatial task*. Several examples of spatial tasks include ride-sharing, delivery services, and crowd-sensing tasks. By using existing general purpose crowdsourcing marketplaces to solve spatial tasks will result in low quality results as they are unable to assign workers based on the spatial proximity of the tasks. Experimental evaluation by using real deployment of Luna as well as by using real dataset shows that Luna is able to provide an efficient platform for spatial crowdsourcing applications.

4.1 Introduction

In recent years, crowdsourcing has been gaining a lot of popularity due to its capability in solving various computer-hard tasks, e.g., filling surveys, image transcription, and audio transcription. The popularity of crowdsourcing can be seen by the existence of several famous commercial crowdsourcing marketplaces, including Amazon Mechanical

Turk [5], Upwork [34], and Figure Eight [6]. In such marketplace, task requester submits a task, alongside its budget and deadline, and wait for the task to be completed by workers. Meanwhile, when a worker decides to work on a task, the worker will browse for the available tasks in the marketplace and select the task that the worker decides to work on which will be rewarded by the requester. Database researchers have been using such marketplaces to employ the crowd to solve various database operations, including join operations [32, 35], aggregating and counting operations [36, 37], and to compute skyline queries over noisy data [38] among many other operations.

While these general marketplaces are widely used to solve lots of general tasks, many tasks were born to be spatially oriented, namely *spatial tasks*. In such tasks, the location of the worker plays an important role in solving the tasks efficiently. For example, ride-sharing and delivery services can only be done by workers who are located within the area of the tasks, image geotagging will be done more accurately by workers who live near the location of the image or have some knowledge about the location of the image, and rating a restaurant would be preferred to be done by local workers. Unfortunately, using the general crowdsourcing marketplaces to solve spatial tasks results in low quality results since these marketplaces are not spatially-aware where they randomly select workers to solve the tasks regardless of the tasks' and the workers' location.

The popularity of spatial tasks, hindered by the limitations of existing general crowdsourcing marketplaces to support them, urges both industry and academia to provide several *spatial crowdsourcing* solutions where each solution is designed to tackle a specific type of spatial task. This includes specific solutions for ride sharing [39,40], another solution for crowd sensing [41], another solutions for asking workers to go to a certain location to do a task [11, 42], and so on. For surveys of such tasks, see [43, 44]. Despite all such efforts on providing solutions for different kinds of spatial tasks, these approaches are not scalable from a system point of view. Each technique requires its developers to rebuild the same system components that have been built by other spatial tasks' solutions while, in fact, most of them are sharing the same execution logic and system components with each other. For example, both ride-sharing and crowd-sensing solutions try to find workers within a specific location to be recruited. As a result, by creating a single module that has the capability in finding such workers will be able to empower both solutions at the same time.

In this chapter, we deploy a spatial crowdsourcing marketplace out of Luna by just using the four simple plug-ins of Luna. In spatial crowdsourcing marketplace, a task requester submits a spatial task which includes the spatial proximity of the task where only workers who are located at that area can work on the task. Here, we discuss the detail of each plugin that is used by Luna to empower a spatial crowdsourcing marketplace. Later in the chapter, we also provide examples on how to instantiate a more specialized deployment of two spatial crowdsourcing applications out of Luna, namely *Ride-Sharing* in Chapter 4.4 and *Image-Geotagging* in Chapter 4.6, in order to provide a more optimized solution for each of the applications. Experimental evaluation based on real deployment of Luna as well as real dataset show that Luna can be used as a platform to instantiate a spatial crowdsourcing marketplace.

4.2 Preliminaries

This chapter discusses a set of preliminaries for the spatial crowdsourcing marketplace problem, namely *spatial tasks*, *workers*, and *spatial crowdsourcing*.

Spatial Tasks. A spatial task is defined as $T = \{loc, y, B, D\}$ where loc is the spatial location of the task, y is the type of the location where it can either be *physical* or *knowledge*, B is the number of workers that are required to complete the task, i.e., the task budget, and D is the deadline for the task. A spatial location of a task $T.loc$ is further defined as a circle with the form of (lat, lon, r) where lat and lon is the latitude and longitude, respectively, and r is the radius of the task spatial proximity from $loc.lat$ and $loc.lon$. The length of the radius and the location type depend on the type of the spatial task. For example, a ride-sharing task will have a smaller radius with $T.y$ is *physical* while a translation task will have a larger radius with $T.y$ is *knowledge* as we do not require workers to be currently physically located at the task location in order to solve the task.

Workers. A worker is defined as $w = \{loc, K\}$ where loc is the *physical* location of the worker currently is located and K is a set of m *knowledge* location(s) that the worker is expert at, i.e., $K = \{loc_1, loc_2, ..., loc_m\}$. A worker has exactly one *physical* location, however, the worker may have more than one *knowledge* locations. For example, a worker that is *physically* located at Minneapolis, MN, may also have *knowledge* locations

of Seattle, WA and New York, NY if the worker have lived in both locations before. Note that a location where the worker is physically located will also be considered as one of the knowledge location that the worker is expert at.

Spatial Crowdsourcing. In *spatial crowdsourcing*, requester submits a *spatial task* to the platform. Then, depending on the location type of the task, workers who are located within the spatial boundary of the task, either *physical* or *knowledge* boundary, are able to browse for the task and perform the task. For example, a ride-sharing task will require workers to be *physically* located within the pick-up location of the task in order to complete the task. Meanwhile, a translation task only requires workers who have the *knowledge* location of the two languages that they are translating from/to in order to browse for the task. Then, workers will report to the platform when they have finished to task and the result is reported back to the requester.

4.3 Luna Configuration

Using Luna, one can use the following SQL-like command to instantiate a spatial crowdsourcing marketplace.

CREATE APPLICATION *Spatial Crowdsourcing Marketplace*
EXPERTISE-INDEX *Spatial*
TASK-SUBMISSION *submit spatial range query*
TASK-REQUEST *request spatial range query*
RESULT-EVALUATION *all*

In this spatial crowdsourcing marketplace, the plugged-in index structure and the three plug-in functions are as follows:

- **Expertise Index = *Spatial.*** We use the *Spatial* expertise index structure, discussed in Chapter 2.5, which is one of the two default data structures in Luna. A task is mapped to the deepest grid cell that can cover the task location. Then, the task exact location and the location type are stored in the Task Properties field in the Task Index. That grid cell is considered as the expertise domain of the task. Meanwhile, for each of the worker's location, we map the location to every

grid cells that include the worker's location, i.e., one cell per each index level. The worker's exact location and the location type are stored in the *Worker Properties* field in the Worker Index.

- `WorkerRank` = ***submit spatial range query.*** In this plug-in function, we are not using the default one of Luna. Instead, we define a *submit spatial range query* as the `WorkerRank` ranking function of the Submit Task module. This is done by checking each of the standby workers that is inputted to the function while filtering out workers that cannot be assigned with the task. There are two requirements that each worker needs to fulfill in order to be eligible for the task. Firstly, we need to check each worker's exact location whether or not the worker's location is located within the spatial proximity of the task. This is because the grid cell that contains the task may cover a larger spatial area than the task spatial proximity. Secondly, if the task requires physical location of the workers, then we need to filter out workers that are only located based on knowledge. Then, we select B workers that fulfill those two requirements. If there are more than B workers, we select the top-B workers in terms of longest waiting time.

- `TaskRank` = ***request spatial range query.*** In this plug-in function, we are not using the default one of Luna. Instead, we define a *request spatial range query* function for the `TaskRank` ranking function of the Request Task module. Similar to the submit spatial range query of the `WorkerRank` function of the Submit Task module, there are two requirments that we need to check for each task that is inputted to the `TaskRank` function: (1) filtering out tasks which spatial proximity do not contain the worker's location and (2) for each of the worker's knowledge location, we filter out tasks that require physical location type for the worker. If there are more than one task for the worker, then we select one with the earliest deadline.

- `Aggregate` = ***all.*** In this plug-in function, we use the default `Aggregate` function for the Result Evaluation module to return every worker's result back to the requester.

4.4 Ride-Sharing

In this chapter, we study one of the spatial crowdsourcing marketplace application, namely *ride-sharing*, and see how we can provide a more specialized solution by using Luna. In the ride-sharing problem, the requester submits a task asking to be picked up from a certain location and dropped off at another location. The task will have a budget of one, asking for only one driver to complete the task. As previously mentioned, we consider the task budget in terms of number of workers needed for the task. The pricing of the task per worker is out of the scope of the thesis. Though ride-sharing is a crowdsourcing problem, it cannot make use of general purpose crowdsourcing platforms such as Amazon Mechanical Turk, as they do not take the location proximity into account. As a result, and due to the importance of the problem, a whole industry came out with their own special purpose ride-sharing applications, e.g., Uber [45], Lyft [46], and Didi [47]. Note that we only consider the physical location for both worker and task in this problem.

Using Luna, one can use the following SQL-like command to instantiate an expert-sourcing ride-sharing application, where requesters submit their requests for ride sharing, and Luna would take care of finding a suitable driver (i.e., expert) with the same properties that a commercial ride-sharing application would do.

CREATE APPLICATION *Ride-Sharing*
EXPERTISE-INDEX *Spatial*
TASK-SUBMISSION *nearest-worker*
TASK-REQUEST *nearest-task*
RESULT-EVALUATION *all*

In such ride-sharing application, the plugged-in index structure and the three plug-in functions are as follows:

- **Expertise Index = *Spatial.*** We use the *Spatial* expertise index structure, discussed in Chapter 2.5, which is one of the two default data structures in Luna. A task is mapped to the deepest grid cell that includes its exact location, which is stored in the task properties field in the Task Index. That grid cell is considered

Algorithm 5 Nearest-Worker Ranking Function

```
1: procedure NEAREST-WORKER(Task T, Workers W, Budget 1)
2:     Worker w ← ∅
3:     Distance d ← ∞
4:     for each Worker w_temp in W do
5:         if dist(T.loc, w_temp.loc) < d then
6:             d ← dist(T.loc, w_temp.loc)
7:             w ← w_temp
8:         end if
9:     end for
10:    C ← Circle(T.loc, d)
11:    G ← overlapping cells in the Expertise Index with C
12:    for each Grid cell g in G do
13:        for each Worker w_temp in g.WorkerList do
14:            if dist(T.loc, w_temp.loc) < d then
15:                d ← dist(T.loc, w_temp.loc)
16:                w ← w_temp
17:            end if
18:        end for
19:    end for
20:    return w
21: end procedure
```

as the expertise domain of the task. Meanwhile, a worker is mapped to h grid cells that include the worker's exact location (stored in the *Worker Properties* field in the Worker Index), where h is the height of the index structure, as one cell per each index level.

- `WorkerRank` = ***nearest-worker.*** We use a function to return the nearest worker to the task location for the `WorkerRank` function of the Submit Task module. The input to the function would be the task, a list of workers W who are located in the same grid cell C as the task location, and the task budget set to one. The function would return the nearest worker w among W to the task location among the workers in C. This is done by comparing the task exact location from the *Task Properties* in the Task Index to every worker's, in W, exact location that is retrieved from the *Worker Properties* in the Worker Index. However, this may not be the absolute nearest worker to the task, especially if the task location is close

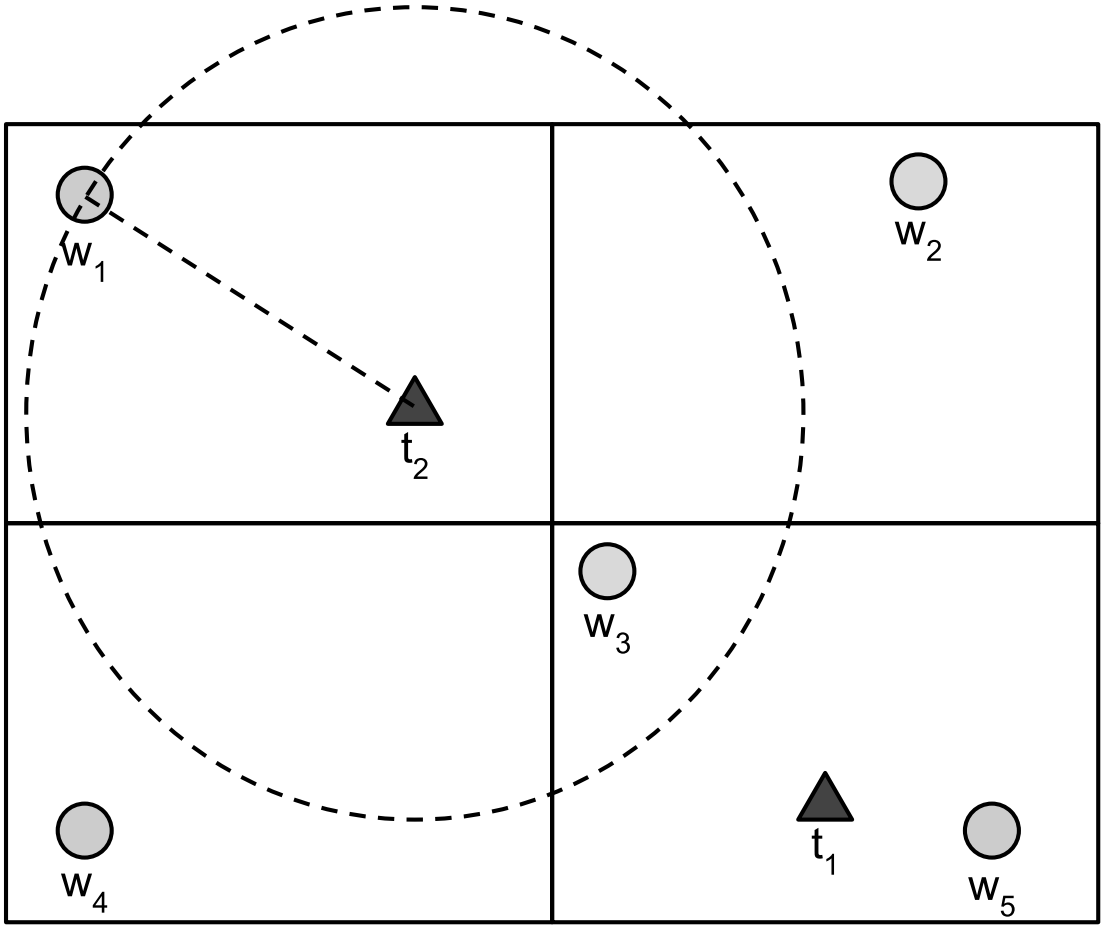


Figure 4.1: Example of Nearest-Worker Ranking Function

to the cell boundary. To ensure this, we would make a simple test by drawing a virtual circle centered at the task location with a radius to the nearest worker in W. If this circle is fully enclosed inside the grid cell C, then we conclude that w is the absolute nearest worker to the task. Otherwise, there is a probability that there is another nearest worker in any grid cells that overlap with the virtual circle. In that case, we will check on the workers in the *Spatial* index nodes that correspond to the cells that overlap with the virtual circle to see if there is another nearest worker. To minimize the need for comparing against the virtual circle, we can either: (a) use cells with larger area, as this will minimize the probability that the task is close to the boundary, and/or (b) have a threshold of area overlap that

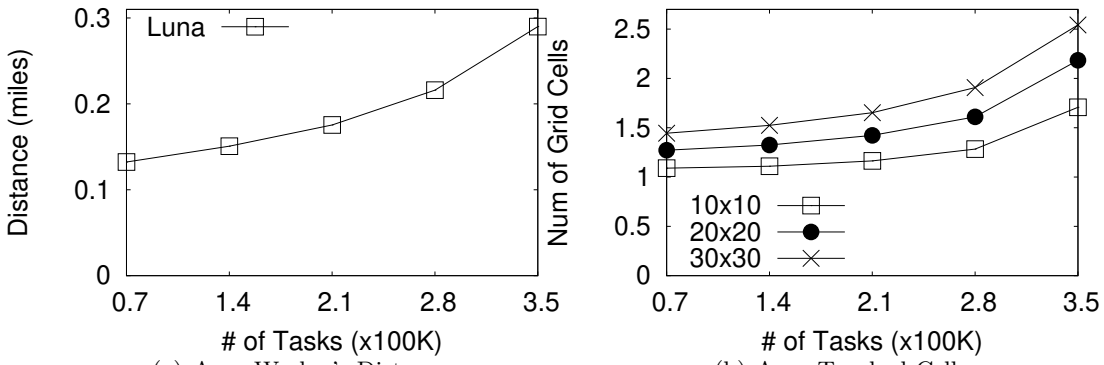
if not exceeded, we just go with the current nearest worker. Algorithm 5 gives the high level overview of this function. Figure 4.1 gives an example of the nearest-worker ranking function. For task t_1, the function will first check the standby workers in the bottom right grid cell, i.e., w_5 and w_3, and set w_5 as the closest worker to the task. In this case, we will not check the neighboring cells because the distance between t_1 and w_5 will not overlap with other grid cells, and thus the function will return w_5. For task t_2, the function will initially set w_1 as the worker for the task as w_1 is the only worker that is located with the task. However, when we draw a circle where the center of the circle is t_2 and the radius is the distance between w_1 and t_2, the circle overlaps with the other three grid cells. As a result, the function will need to check every worker in those three cells, i.e., w_2, w_3, w_4, and w_5, to select the closest worker with t_2, namely w_3.

- `TaskRank` = ***nearest-task.*** We use a function to return the nearest task to the worker location for the `TaskRank` function of the Request Task module. The input would be the list of tasks in either the same grid cell C as the worker location, or in any of C's ancestor gird cells. Similar to the case of the *nearest-worker* function, we may need to have a virtual circle centered at the worker location with radius until the nearest task. If that circle overlaps other grid cells, we would need to check there too.

- `Aggregate` = ***all.*** We use the default aggregate function of Luna for the `Aggregate` function of the Result Evaluation module, which basically returns all the answers it get to the user. In this case, the returned answer is just one tuple including the worker who is willing to share the ride.

4.5 Ride-Sharing Experiment

This experiment evaluates the *ride-sharing* application that is instantiated out of Luna, as discussed in Chapter 4.4. In this experiment, we evaluate its performance based on two metrics: (a) the average distance that the assigned driver needs to travel in order to pick up a requesting rider, and (2) the number of grid cells of the expertise index that are touched for each query. The first metric shows how *expert* Luna can assign a worker



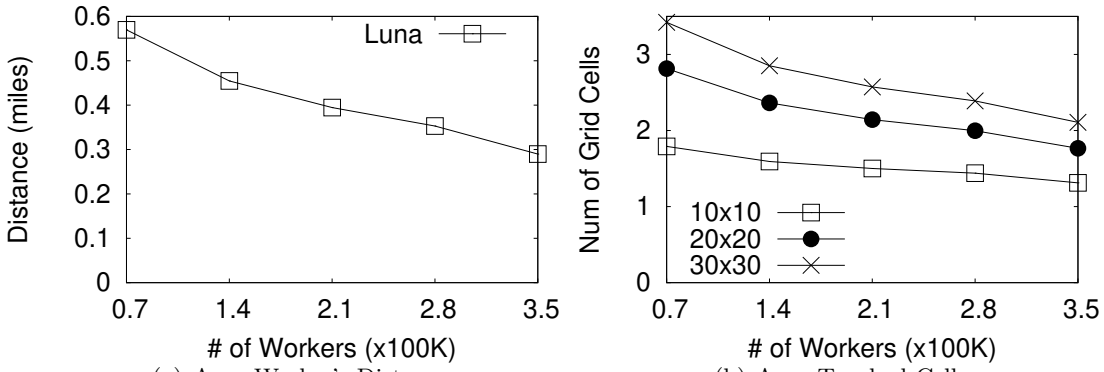
(a) Avg. Worker's Distance (b) Avg. Touched Cells

Figure 4.2: Varying Number of Tasks

with a task. The closer the task is to the worker the much better and more accurate result we can get. The second metric gives the query performance of Luna, where less visited cells means more efficient performance. Unless mentioned otherwise, we use a *Spatial* expertise index structure with 30×30 grid cells in its lowest level which indexes the location of New York, USA (with latitude from $40.325°$ to $41.15°$ and longitude from $-74.405°$ to $-73.44°$). We do not compare our approach against AMT-like deployment as AMT is completely unsuitable for this application. Our prototype run in Java 1.8.0-151 on a machine with Intel Quad Core i7-4790 3.6Ghz, two threads per core, and 32GB of RAM running 64-bit Ubuntu 16.04.

Recruiting Workers

We use the publicly available New York city taxi trip data [48] as a source to simulate ride sharing task requesters and working drivers. In particular, we use a total of 700K trips from this data, out of which we use: (a) 350K trips from January 21^{st}, 2015 as trip requests timestamped with the time of the trip requests, and a fixed deadline of 15 minutes for each trip. The trip start locations are mapped to our default expertise index, used in this application, to identify the expertise needed for this trip as the workers located in the same grid cell, (b) The other 350K trips from January 28^{th}, 2015 are used as workers, where the start time of the trip indicates a time when the worker is available. The workers locations are set as the start locations of the trips, and are also mapped to the expertise index indicating the worker expertise.



(a) Avg. Worker's Distance (b) Avg. Touched Cells

Figure 4.3: Varying Number of Workers

Experimental Results

Figure 4.2 gives the result of Luna when varying the number of the submitted tasks from 70K to 350K, while fixing the number of workers to 350K. In Figure 4.2(a), we see that Luna is able to assign workers with an average distance of 0.2 miles from the task, which is much reasonable distance and matches the expectations of a ride sharing application. However, we see that the distance between task and worker increases with more tasks being submitted. The reason is that the workers that are originally closer to a given task may have been assigned with other tasks that are submitted beforehand. Figure 4.2(b) also confirms this finding where we touch more grid cells when we increase the number of submitted tasks due to the increase of distance between worker and task. In this figure, we also see that Luna touches more cells when it indexes the location with more grid cells due to the smaller area that is covered by each cell. However, in a worst case with 350K and 30×30 gird cells, we still below 2.5 average grid cells per request, which is still very reasonable. In the case of 70K task requests and a grid cell size of 10×10, we have an average of one grid cell per requests, which means that Luna was always able to find an experts drivers who is located in the same cell as the ride task request.

Figure 4.3 gives the result of Luna when varying the number of available workers from 70K to 350K, while fixing the number of tasks to 350K. In contrast to the results of varying the number of tasks, we are able to assign tasks to closer workers when there are more available workers. This is because with more available workers, each

task has more candidate workers to select from, and thus has a higher probability of a closer worker to be available. The distance that the driver needs to travel to pick up a rider ranges between 0.3 and 0.6 miles, which is very reasonable for a ride sharing task. Figure 4.3(b) also confirms this finding where we touch less grid cells when we increase the number of available workers.

4.6 Image-Geotagging

In this chapter, we study another example of spatial crowdsourcing marketplace application, namely *image-geotagging*, and see how we can provide a more specialized solution by using Luna. In the *image-geotagging* problem, the requester submits an image that needs to be geotagged and a budget that the requester is willing to pay. The answer is returned as the location of the image. The main challenge of the geotagging problem is that not every worker is familiar with the exact location of the image, especially with images that are not widely popular. For example, given an image of the University of Minnesota campus, most people will not be able to determine the location of the image. Hence, using an Amazon Mechanical Turk (AMT)-like platform would not help here. An alternative approach suggests that workers located closer to the image location are able to provide more accurate geotagging than those located far from the image location [2].

Using Luna, one can instantiate an image-geotagging application that recruits experts at the image location as a means of ensuring accurate location of the image. We will describe this process in two applications. In the first application, termed *area-based image-geotagging*, we assume that the requester has a broad idea about the image location, but wants to enhance it more. For example, the requester knows that the image is in Minnesota, but needs to get a more detailed location, hence need to recruit workers in Minnesota. In the second application, termed *image-geotagging*, we consider the general case that the requester has no idea whatsoever about the image location, yet, the requester needs to recruit workers who are located close to the "unknown" image location, as they are considered more experts. The relation between these two applications is similar to the relation between the domain-specific image-labeling (Chapter 3.2) and the general image-labeling (Chapter 3.6).

Area-based Image-Geotagging. One can use the following SQL-like command to instantiate an area-based image-geotagging application, where requesters submit their requests as images to be geotagged, along with the budget (i.e., number of required workers), and the known area (i.e., expertise) of the image.

CREATE APPLICATION *Area-based Image-Geotagging*
EXPERTISE-INDEX *Spatial*
TASK-SUBMISSION *FIFO*
TASK-REQUEST *expertise-first*
RESULT-EVALUATION *distance-aware-majority*

In such area-based image-geotagging application, the plugged-in index structure and the three plug-in functions are as follows:

- **Expertise Index = *Spatial.*** We use the *Spatial* expertise index structure, discussed in Chapter 2.5, which is one of the two default data structures in Luna. Given the known area of the image, the *Spatial* index will be used to locate the most specialized (or deepest) grid cell that fully covers the area given by the requester.

- `WorkerRank` **= *FIFO.*** We use the default FIFO approach for the `WorkerRank` function of the Submit Task module. This function would return the top-B workers, where B is the task budget, in terms of the longest waiting time in the most specialized grid cell that fully covers the area of the task provided by the requester.

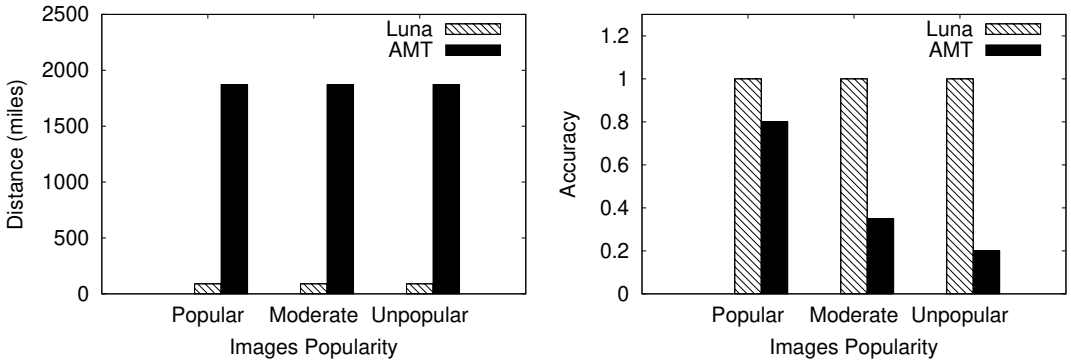
- `TaskRank` **= *expertise-first.*** We use the default *expertise-first* approach for the `TaskRank` function of the Request Task module. Given a worker w with multiple location expertise, all mapped into various grid cells in the *Spatial* index structure, the `TaskRank` function would return only one task, if any, from the set of tasks that are waiting in every the grid cell that w is expert at. With the *expertise-first* approach, the task that is located at the deepest grid cell would be returned and assigned to the worker. In case of having more than one task with similar deepest grid cell, the one with earliest deadline would be returned.

- `Aggregate` = ***distance-aware-majority.*** In this plug-in function, we use the *distance-aware-majority* function as the `Aggregate` function of the Result Evaluation module. For each worker's result, in the form of (*latitude*, *longitude*), we first project the worker's answer into a grid cell C that is located at one level below the known area of the task. Then, we weight the worker's answer with the following function: $d_{max} - d_w + 1$ where d_w is the cell distance between C and the grid cell where the worker is located on that level and d_{max} is the maximum cell distance on that level. Then, the final result will be the cell with the highest weighted sum. If the area is already the leaf node of the index, then we return the *minimum-bounding-rectangle* of the workers' results.

Image-geotagging. In the image-geotagging problem, the location of the image is not available to the requester in advance. Hence, we will use an iterative approach similar to the one in the image-labeling problem (Algorithm 4). In particular, we start at the root node of the *Spatial* index structure, which covers the whole geographical space. With part of the budget, we can narrow down the search space to USA, for example. Then, in a second iteration, we use another part of the budget with expertise node that covers USA. The result would be something around the US midwest area. Then, a third iteration will call for those workers who are living or have expertise in the US midwest area. A fourth iteration may lead to Minnesota, and a fifth one may give the absolute accurate result. The bottom line is that we are able to recruit expert workers even thought the location was completely unknown at the start.

4.7 Image-Geotagging Experiment

This experiment evaluates the *image-geotagging* application that is instantiated out of Luna, as discussed in Chapter 4.6. In this experiment, we set the index structure that is used by Luna into a three-levels hierarchical index structure as follows: (1) The first level of the index is a grid cell that covers the whole United States. (2) The second level of the index splits the United States into four non-overlapping regions. (3) The third level of the index has 48 cells where each cell is a state in the United States (we do not include the state of Alaska and Hawaii). With such index structure, for each image, we will iterate over the Luna-based *area-based image-geotagging* 3× to narrow down the



(a) Avg. Worker's Distance (b) Accuracy

Figure 4.4: Image-Geotagging Experiment

area of the image. We compare our Luna-based approach against a general deployment of image-geotagging in Amazon Mechanical Turk (AMT) where we randomly assign workers. We manually select 20 images to be geotagged and we use a budget of 30 workers to geotag each image for both approaches.

Recruiting Workers

As Luna is still new platform, we follow a similar approach to our domain-specific image-labeling application in recruiting workers for this experiment. In particular, we populate the Worker Index of Luna with real AMT workers. However, in this experiment, we used a total of 600 real workers populated in Luna from AMT. To get the location of each worker, we ask the workers to provide their home zip-code. Then, based on the worker's answer, we insert the worker to every cell that contains the worker's reported zip-code in the Expertise Index.

Experimental Results

To ease the readability, based on the results, we categorize every image into three categories: *popular*, *moderately popular*, and *unpopular* images. A popular image is an image that more than 50% of the workers agree on its exact location, thus, AMT can geotag the image accurately with majority voting. A moderately popular image is an image that more than 50% of workers are aware with its location, e.g., they are aware in which state of the US that the image is located. We categorize the rest as unpopular.

Figure 4.4(a) gives the average distance between the assigned workers with the images. In the case of Luna, the average distance is calculated based on the workers that are assigned with each image at the last iteration since these workers are the ones that provide the final location of the image. For any image popularities, we see that Luna is able to recruit significantly more expert workers than AMT. Specifically, Luna and AMT are able to recruit workers with an average distance of 88 miles and 1,879 miles, respectively. This is due to the ability of Luna to narrow down the search space of each image via its iterative approach.

Figure 4.4(b) gives the accuracy of both approaches in geotagging images across various image popularities. In the case of AMT, we calculate the accuracy based on the ratio of workers that provide the accurate city of each image. The main reason is that if we use majority voting to calculate the final city of each image, AMT will have 0% accuracy for both moderately popular and unpopular images since the majority of the workers are unable to pinpoint the accurate city of the image. As shown in the figure, Luna significantly outperform AMT for all image popularities. For popular images, while 80% of AMT workers are able to provide the accurate city of the images, the rest of the workers provided locations that are completely inaccurate. Furthermore, the accuracy of AMT is worsen for both moderately popular and unpopular images where it has an accuracy of 35% and 20%, respectively. In contrast, Luna is able to maintain its accuracy across all image popularities by being able to keep narrowing down the area of the image by iterating through the expertise index.

4.8 Related Work

This chapter discusses the related work of the spatial crowdsourcing marketplace case study. Several research efforts have been conducted in studying the spatial crowdsourcing problem where each crowdsourced task contains a spatial information about the task and/or the workers [9–12, 40, 42–44, 49–57]. In addition, several spatial crowdsourcing frameworks require the workers to be physically located around the task location and able to physically travel to the task location in order to complete the task, e.g., taking a picture of an object, delivery service, or ride-sharing applications. With Luna, instead of creating the whole system stack to support such spatial crowdsourcing frameworks

from scratch, in this chapter, we show that we can configure Luna to provide a spatial crowdsourcing platform for those frameworks by just using the four plugins of Luna.

Many efforts have studied the task assignment problem in the spatial crowdsourcing environment due to its importance in increasing the overall performance of the framework. This is especially important for tasks that require workers to physically travel to the tasks location since the physical traveling of a worker can incur high latency and cost. In particular, there two different modes of spatial task assignment that have been studied in the literature: (1) *batch* mode [1, 42, 56, 57] where the spatial crowdsourcing platform periodically assign the available workers to the available tasks at the current timestamp; (2) *online* mode [58, 59] where the spatial crowdsourcing platform immediately assign suitable tasks to the worker whenever the worker joins the platform. In both modes, the main goal of the spatial task assignment is usually to reduce the distance of the assignment. In Luna, by default, a task will be assigned to workers who are located around the task area based on the cell size of the expertise index that is used. However, any of these approaches can be easily implemented within the `TaskRank` and `WorkerRank` plugins.

Chapter 5

Translation

This chapter discusses another case study of an expert-sourcing application that can be instantiated out of Luna, namely crowd-powered *translation* application. Translation is one example of a crowdsourcing application that currently cannot be solved by using existing crowdsourcing marketplace, e.g., Amazon Mechanical Turk or Figure Eight. The main reason is that each translation task requires the workers to be familiar in both the source and the output languages in order to provide the correct translation, while, general crowdsourcing marketplace will assign each translation task to random workers. In this chapter, we discuss the four plug-ins that we use to configure Luna in order to act as the platform for a translation application.

5.1 Introduction

Translation takes three inputs, i.e., the text to be translated, the source language, and the output language, and the goal of this process is to convert the text from the source language to the output language. Translataion has been one of the main common features that have been widely used by many of the recent applications. For example, Google Translate [60] has more than 200 million active users monthly [61], many web pages currently have multilingual support, and many translation applications are currently widely available for travelers to use, e.g., iTranslate Voice [62] and SayHi [63]. The translation capability of those applications is currently empowered by using natural

language processing (NLP) technique which is trained by using large bilingual sentence-aligned parallel corpora. An example of such corpora is the Canadian Hansard data [64] which contain the transcripts of Parliamentary Debates in Canada that are stored in both French and English (required by law), and thus, such data can be used as the training dataset for translation techniques between French and English. Unfortunately, large bilingual parallel corpora only exist for relatively few languages pairs.

With the limited amount of available bilingual parallel corpora, multiple research efforts have been introduced to produce new training dataset [65–71]. While those techniques were able to help in producing new training dataset, unfortunately, they are still limited to an existing data source to create the new dataset. One way to create a new bilingual parallel corpora is by hiring professional translators, however, such approach is very expensive as shown by [72] that it costs $0.36/word. As a result, several research efforts have been trying to embrace crowdsourcing to hire non-professional workers in translating one language to another [73–77]. These crowdsourcing approaches have proposed techniques that are geared towards increasing the translation quality for non-professional workers. Unfortunately, without having any expert-sourcing platforms to support them, they deploy their techniques on top of general crowdsourcing marketplace where they will recruit random workers for each task and hope that those workers are fluent in both source and resulting languages, i.e., expert in both languages. In doing so, such approaches will not be able to utilize the provided budget effectively as there is a large probability that the workers who decide to pickup the task are not fluent in both languages.

In this chapter, we deploy a platform for crowdsourcing translation application out of Luna by just using the four simple plug-ins of Luna. In particular, we provide a platform for existing crowdsourcing techniques to recruit only workers who are expert at the source and resulting languages. With Luna, this allows those techniques to send the task to only expert workers and then evaluate their results. Here we provide two different configurations that we can use to empower such platform. The first configuration will modify both the `WorkerRank` function of the Submit Task module and the `TaskRank` function of the Request Task module. In the second configuration, we show how we can achieve a more efficient performance out of Luna by just changing the Expertise Index that is used by the translation application.

5.2 Luna Configuration 1: Ranking Functions

In the *translation* problem, the requester submits a text, task budget, the source language and the output language. This is one example of the tasks that has been solved by the general purpose crowdsourcing platforms like Amazon Mechanical Turk. However, such platforms do not provide any guarantees on the expertise of the workers with respect to the source and/or the output languages. As a result, existing works manually put the required expertise of each task, i.e., the source and the output languages, into the description of the task, hoping that the workers that decide to work on the task are really expert at those languages. In contrast, using Luna, workers who are experts in both the source and output languages are automatically assigned.

There are two different approaches that we can use in configuring Luna to handle the translation problem. In this chapter, we will discuss our first approach where we create two ranking functions, i.e., one for the `WorkerRank` function and one for the `TaskRank` function, that can be used for this problem. In our second configuration (Chapter 5.3), we show that we can just change the expertise index that is used and the translation problem is automatically supported by Luna.

Using Luna, one can use the following SQL-like command to instantiate an expert-sourcing translation application.

CREATE APPLICATION *Translation*
EXPERTISE-INDEX *Languages*
TASK-SUBMISSION *worker-language-filter*
TASK-REQUEST *task-language-filter*
RESULT-EVALUATION *translation-scoring*

In such translation application, the plugged-in index structure and the three plug-in functions are as follows:

- **Expertise Index = *Languages*.** For a translation task, a worker would need to be expert in two languages of the task, i.e., the task source language and the task output language. As a result, we upload a list of languages, e.g., English, Chinese, and Urdu, as the expertise index for the application as a flat index

Algorithm 6 Worker-Language-Filter `WorkerRank` Function

```
1: procedure FILTER(Task T, Workers W, Budget B)
2:     Workers R_w ← ∅
3:     Language L ← T.Properties
4:     for each Worker w in W do
5:         if B = 0 then
6:             break
7:         end if
8:         if L ⊂ w.expertise then
9:             R_w ∪ w
10:            B = B − 1
11:        end if
12:    end for
13:    return R_w
14: end procedure
```

structure with one dummy root node. Then, a worker who is expert at a set of languages will be stored in every expertise node, where each node represents each of those languages, in the index. For example, if a worker is fluent in both English and Chinese, then we will store the worker's information in the English expertise node and the Chinese expertise node. Meanwhile, a translation task will be stored into the expertise node of one of the two languages that are needed to complete the task, i.e., either the source language or the output language. The decision on which language that is going to be used to store the task is decided by the administrator. Then, we store the other language will be stored in the Task Properties of the task.

- `WorkerRank` = ***worker-language-filter.*** Instead of using the default FIFO approach for the `WorkerRank` function of the Submit Task module, we develop a worker-language-filter function. Recall that the Language expertise index that is used by the application will only store one of the two languages for each task, i.e., either the source or the output languages. As a result, when we have retrieve the standby workers that are available for the task, we need to do additional filtering of those workers for the other language. Algorithm 6 gives the pseudocode of the worker-language-filter function. Note that it is possible that the multiple standby

workers that are inputted into the functions may not be expert in the other language. In this case, we will ignore those workers. If we haven't used all of the budget yet, then we will store the task as a standby task into the expertise index so that it can be picked up later by future expert workers.

- `TaskRank` = ***task-language-first.*** Instead of using the default expertise-first ranking function for the `TaskRank` function of the Request Task module, we develop a task-language-filter function that acts similarly to the worker-language-filter function of the `WorkerRank` function. In the task-language-filter function, for a given worker who is looking for a task to complete, we will retrieve the standby tasks from the expertise index. The worker is guaranteed to be expert at one of the two languages of these standby tasks. Then, we will scan through each of the standby tasks to check whether or not the worker is expert at the task other required language that is stored in the Task Properties field. Similar to the `WorkerRank` function, the worker may not be expert at any of the standby tasks. In this case, the function will not return any tasks.

- `Aggregate` = ***translation-scoring.*** There are already well known approaches of evaluating the quality of a certain translation in the crowdsourcing environment, e.g., see [73]. This is done by comparing each translation against existing professionally-produced translations where the more the selected translations resemble the professional ones, the higher the quality. The scoring is based on a set of information features, e.g., country of residence and the number of years that the worker has spoken the languages. We encapsulate this functionality in our `Aggregate` plug-in function of the Result Evaluation module to score each translation, and return the one with the highest score.

5.3 Luna Configuration 2: Expertise Index

While the first configuration of Luna that we used to tackle the translation problem in Chapter 5.2 is able to serve the translation applications better than any existing general crowdsourcing platforms in supporting them, such configuration still has one main weakness. In particular, for each task (or worker) assignment that we are going to

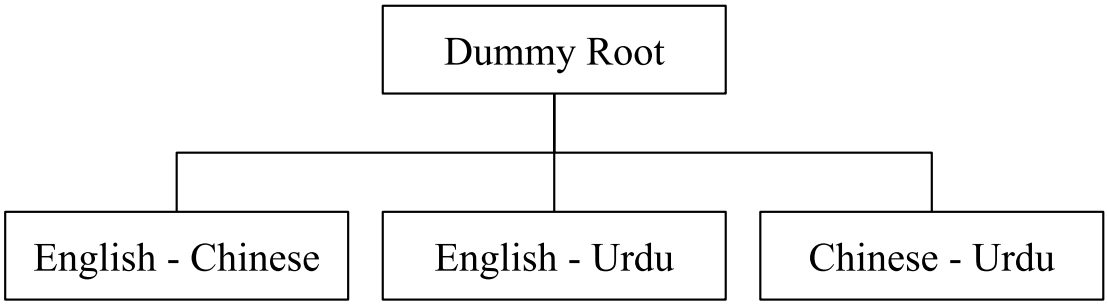


Figure 5.1: The Language-Language Expertise Index

make, we need to further filter-out the standby workers (or tasks) in the `WorkerRank` (or `TaskRank`, respectively) function. This is due to having two expertise domains that are required in the translation problem, i.e., the source language and the output language. As a result, there will be cases where all standby tasks (or workers) that are inputted to the ranking function will not match the required expertise. In this chapter, we provide the second configuration of Luna that can tackle the above issue and provide a more efficient expert-sourcing platform for the translation problem.

In particular, we use the following SQL-like command to instantiate an expert-sourcing translation application.

CREATE APPLICATION *Translation*
EXPERTISE-INDEX *Language-Language*
TASK-SUBMISSION *FIFO*
TASK-REQUEST *deadline-first*
RESULT-EVALUATION *translation-scoring*

In such translation application, the plugged-in index structure and the three plug-in functions are as follows:

- **Expertise Index = *Language-Language*.** For a translation task, a worker would need to be expert in both the source and output languages. So, instead of uploading a list of languages as the expertise index for the application, we create a new *Language-Language* expertise index as a flat index structure with one dummy root node. Figure 5.1 gives an example of the Language-Language index structure

with three different languages, namely English, Chinese, and Urdu. Each node in the index represents an expertise in a certain pair of languages, i.e., "English-Urdu", "English-Chinese", and "Chinese-Urdu". With this index, an incoming translation task is mapped to its corresponding nodes based on its required source and output languages. Similarly, a worker who is an expert in more than one languages is placed in all expertise nodes of all possible pair of languages within the worker's expertise. As described in Chapter 2.5, a new index structure can be uploaded to Luna by uploading a file containing a list of (E, E_P) tuples where E is an expertise and E_P is the parent expertise of E. In this case, we will upload tuples that represent every possible pair of languages that will be supported, where E is a pair of languages and E_P is a dummy root for all tuples. By using this index, we do not need to further filter out standby tasks and workers since each worker or task in each node of the index will be expert at both languages.

- `WorkerRank` = ***FIFO.*** Similar to the case study of the domain-specific image-labeling problem (Chapter 3.6), we use the default FIFO approach for the `WorkerRank` function of the Submit Task module. This means that we will assign each translation task to those B workers who are experts in the required pair of languages and have been waiting the longest to perform a task. Note that we do not need to use the worker-language-filter function (discussed in Chapter 5.2) anymore since every standby worker that is inputted to the `WorkerRank` function will be expert in both source and output languages of the task.

- `TaskRank` = ***deadline-first.*** Instead of using the default expertise-first ranking function, we add a new raking function here to select the task with the earliest deadline among all the candidate tasks for the `TaskRank` function of the Request Task module. This is done by simple scanning over all candidate tasks that the worker is expert at and select the task wit the earliest deadline. There is no reason to use the expertise-first ranking function since the expertise index that we use is a flat one. Similar to the `WorkerRank` function, we do not to do additional filtering like the one in the previous configuration as the worker will be expert at both languages of every standby task that is inputted to the function.

- `Aggregate` = ***translation-scoring.*** We are using the same `Aggregate` function

of the Result Evaluation module as the one in Chapter 5.2 where we compare each translation against existing professionally-produced translations.

5.4 Related Work

This chapter discusses the related work of the crowdsourcing translation case study. Most research efforts in translation have been focused on generating a bilingual parallel corpora that will be used as the training dataset for Natural Language Processing (NLP) techniques. Initially, researchers have tried to harness the human power in creating such dataset by recruiting professionals [72]. However, such approach incurs a very high cost, i.e., $0.36/word, which could potentially exceed half a million dollars just to build a small parallel corpus of 1.5 million words in Urdu-English languages. However, with the recent popularity of crowdsourcing, several research efforts have tried to use crowdsourcing in creating such dataset due to the low cost of hiring a worker in crowdsourcing [73–81]. However, as it currently stands, these works are mostly built around the idea that crowdsourcing is unable to recruit workers that are fluent in the source and output languages since there are no crowdsourcing platforms that can assign workers that are fluent in both languages. As a result, these works are providing techniques to filter out inaccurate results. With Luna, each task will now be assigned to workers who are fluent in both languages and any of these techniques can be implemented as one of the four plugins of Luna to further increase the accuracy of the translation.

Chapter 6

Stella: Geotagging Images via Crowdsourcing

Geotagged data (e.g. images or news items) have empowered various important applications, e.g., search engines and news agencies. However, the lack of available geotagged data significantly reduces the impact of such applications. Meanwhile, existing geotagging approaches rely on the existence of prior knowledge, e.g., accurate training dataset for machine learning techniques. This chapter presents Stella; a crowdsourcing framework for image geotagging. The high accuracy of Stella is resulted by being able to recruit workers near the image location even without knowing its location. In addition, Stella also return its confidence about the reported location to help users in understanding the result quality. Experimental evaluation shows that Stella consistently geotags an image with an average of 95% accuracy and 90% of confidence.

6.1 Introduction

Geotagging is the process of attaching a geographic location to an object (e.g., image or news item). Geotagging enables a myriad of important applications, e.g., web search engines use geotagged websites for enhanced search experience [82], location-based services analyze geotagged tweets to extract points-of-interest (POI) [83], and news agencies place geotagged news items on a map for enhanced user experience [84].

Figure 6.1: Example of Geotagging Images

Meanwhile, several research efforts use geotagged data to discover local events [85], identify scenic routes [86], find out flooded areas [87], track food poisoning incidents [88], understanding city demographics [20], and many more. Due to the importance of such applications, academic and commercial systems were built to query, analyze, and visualize geotagged contents, e.g., see [89–91]. Unfortunately, the lack of available geotagged data significantly reduce the impact of such applications, e.g., only 0.7% of tweets [92] and 4.8% of Flickr images [93] are geotagged.

Motivated by the importance of applications that need geotagged data, there were several commercial and academic efforts for geotagging, e.g., see [23,94]. Unfortunately, most efforts are geared toward text-based data using natural language processing technique [94]. Meanwhile, there have been lack of research in image geotagging, where the state-of-the-art technique [23] suffers from low accuracy due to its over reliance with accurate training dataset. This can be seen from using major web search engines, e.g., Google Images [95], to geotag an image. Among the images in Figure 6.1, we could only geotag the first one. This is because web search engines rely on something like Google PlaNet [23], which geotag an image by training a convolutional neural network using millions of geotagged images, thus, unsuitable for new images. This is why it was successful only in geotagging our first image, a popular landmark in Chicago, USA, while other images are not that popular.

As the case with other machine-hard operations, researchers have turned to crowdsourcing to seek the wisdom of the crowd to solve those operations, e.g., POI labeling [1] and image search [96]. Following a similar approach, we tried to use crowdsourcing for

(a) 1^{st} Iteration (b) 2^{nd} Iteration (c) 3^{rd} Iteration

Figure 6.2: Overview of Stella Geotagging Process

image geotagging by running an experiment on Amazon Mechanical Turk [5] where we recruited 60 workers to identify the city of each image in Figure 6.1. It ends up that we were only able to identify the first image. This was expected as the workers were randomly recruited, thus, it would be difficult to geotag the other two images.

Should a crowdsourcing platform is smart enough, it would recruit workers who are familiar with these images. We believe that the closer a worker is to an image location, the more likely that he is able to geotag the image. We call such workers as *domestic* workers. To test our hypothesis, we ran another experiment where we recruited workers only from the state of each image. It ends up that a large majority of the workers were able to identify the city of each image. However, the challenge here is how to recruit those domestic workers, if we do not know the image location itself.

In this chapter, we present Stella; a crowdsourcing-based geotagging framework. Without loss of generality, we describe Stella in the case of image geotagging. Stella pushes the boundaries of crowdsourcing platforms to support geotagging by being able to recruit domestic workers without knowing the image location. Stella overcomes this dilemma by using an *iterative* approach to gradually understand the image location. Figure 6.2 shows an example of Stella in geotagging the second image of Figure 6.1 with a total budget of 60 workers. In the first iteration, Stella recruits only 20 worldwide workers out of its budget of 60. Figure 6.2(a) shows the answers we get from those 20 worldwide workers, where we ended up getting 20 different locations. While an existing crowdsourcing framework (e.g., Amazon Mechanical Turk) would conclude no answer here, Stella takes it further. Stella notices that the majority of the answers are within USA though they are from different locations and thus, concludes that this image must be somewhere in USA. In the second iteration, Stella recruits another 20 workers, all

from USA rather than worldwide workers. Figure 6.2(b) shows the answers we get from those 20 USA workers, where again, we ended up getting 20 different locations within USA. However, Stella notices that the majority of the answers are within the Washington state. Hence, in the third iteration, Stella recruits its last 20 workers, all from the Washington state. Figure 6.2(c) shows the answers from those 20 workers, where there is a clear agreement on the whereabouts of the image. As a result, Stella can safely conclude the location of the image. The main idea is that Stella was finally able to recruit 20 domestic workers, i.e., those workers who live close to the image, even though the image location was unknown.

Stella does not only return the image location as an answer. Instead, it goes beyond to report on how confident it is in the answer. The main idea is to calculate the confidence based on the spatial diversity of the answers. The less spatially diverse answers we have, the more confident Stella is. Also, the more domestic workers we can recruit, the higher the confidence in the answer is. Extensive experiments of Stella using real deployment on Amazon Mechanical Turk shows that Stella is consistently able to geotag images with an average of 95% accuracy and 90% confidence.

6.2 System Overview

Figure 6.3 gives the system architecture of Stella. A user submits an image O that needs to be geotagged and a budget B that the user is willing to pay. The answer is returned to the user as the location of O and a confidence value C that tells how much Stella is confident about the reported location of O. Meanwhile, workers who are registered in Stella indicate their willingness to participate in a given crowdsourcing task and are willing to share part of their location information. Studying the workers' location privacy and incentives is beyond the scope of this thesis (see [54] and [97] for a study on worker's location privacy and incentives, respectively).

For efficient retrieval of workers within a certain area, Stella indexes the workers' locations in a multi-resolution non-overlapping spatial index structure. A worker's location is taken from the worker profile upon registering in Stella. Then, it is updated only when the worker explicitly updates it to get more matching tasks. So, there is no need to track any kind of worker's movement. The index can be a predefined spatial

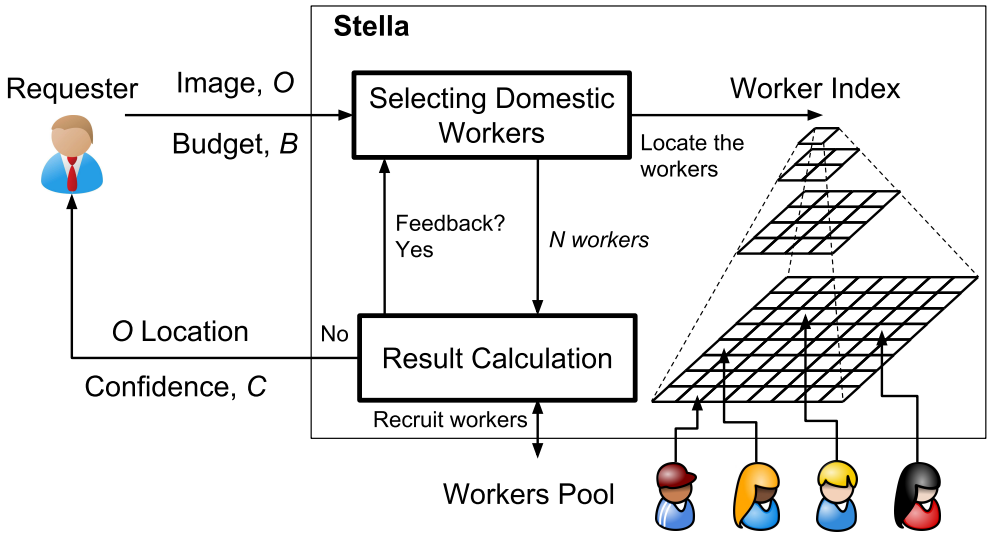


Figure 6.3: Stella System Architecture

regions, such as the whole world in the first level, country in the second level, state in the third level, and so on, or a well-established space partitioning index structure, e.g., a pyramid index structure [98]. For ease of understanding, we use a pyramid structure as our multi-resolution index structure with height H where each worker's location information is stored in one of the cells on the lowest level of the pyramid.

Internally, the main challenge that Stella faces is how to find and recruit domestic workers without knowing the image location. The hypothesis is that domestic workers are more capable of geotagging an image than non-domestic workers. We have verified our hypothesis by running 150+ real crowdsourcing tasks with 600+ workers on Amazon Mechanical Turk (will be discussed further in Chapter 6.6). Stella addresses this challenge by introducing the idea of *adaptive crowdsourcing*. Unlike conventional crowdsourcing platforms that assign a given task to all workers at once, Stella assigns the geotagging task to only a subset of the workers. Then, based on the result, Stella learns more knowledge about the image whereabouts and recruits another subset of the workers that are more domestic than the previous ones. This process is depicted in Figure 6.3 by the iterations over two main internal modules in Stella, namely, the *Selecting Domestic Workers* module and the *Result Calculation* module. The first module receives an amount of budget that it can spend on one iteration, N, as well as a

Algorithm 7 Basic Stella

```
1: procedure GEOTAG(Index P, Image O, Budget B)
2:     S ← P.root; C ← 100%; N ← B/P.H
3:     while S do
4:         W ← SelectDomesticWorkers(N, S)
5:         (S, O.loc, C) ← ResultCalculation(W)
6:     end while
7:     return O.loc, C
8: end procedure
```

hint about O location, provided as a feedback from running the second module from the previous iteration. The goal of this module is to find N domestic workers located within the provided hint, and send their information to the second module. Then, the second module sends the geotagging task to the assigned workers. Unless this is the last iteration, the objective is to find a smaller search space for the next iteration. If this is the last iteration, the module will provide the final answer, along with a confidence value that is computed incrementally across iterations.

6.3 Basic Stella Framework

This section presents our *basic* Stella approach; the simplest form of the Stella framework. The high level overview of this approach is shown in Algorithm 7. This is to focus on the main idea of Stella without digging into the optimization detail on every internal decision of Stella. Stella takes image O, budget B, and pyramid index structure P as its input, and outputs the image location $O.loc$ and a confidence value C. Then, Stella goes through H iterations, where H is the pyramid height, with $N = B/H$ workers assigned in each iteration. Each iteration is composed of two steps: (1) *Selecting domestic workers*, where the objective is to select N workers within a search space, and (2) *Result calculation*, where the objective is to predict the image location along with computing the confidence.

Step 1: Selecting Domestic Workers

This step takes a current search space S and the number of workers N as its input. The goal is to find N workers, inside and uniformly distributed over S, and output those workers to the second step. We do so by exploiting the structure of the pyramid index for the cells rooted at S and recruit $N/4$ workers from each child cell of S. For each cell, we recursively traverse its children until we do not have any workers left or until we reach the lowest level of P. For example, if $N = 4$, we select one worker from each of S children. If $N = 16$, we select one worker from each of the 16 cells that are two levels below S. In general, we will traverse $\lfloor \log_4 N \rfloor$ level(s) to uniformly assign workers. If N is not divisible by four, we assign the remainder of the workers randomly among the cells. When we decide to get workers from a certain cell, we randomly select them.

Step 2: Result Calculation

This step takes the set of N workers from the previous step and sends them the image O. The answer is received from each worker in the form of (*latitude*, *longitude*). Unless this is the last iteration of the geotagging process, we are not trying to infer the exact location of O. Instead, the goal is to identify which cell among the children of S that will become the new search space of the next iteration. So, instead of checking the exact (*latitude*, *longitude*) of every answer, we check which cell among the children of S that these coordinates are located. Then, the answer is seen as if each worker is voting on which child of S contains O. By deploying a simple majority voting, we decide that the quarter that takes the most votes becomes the new S. If this iteration is the last one, we infer the exact location of O. There are many ways to do so, including, reporting the location as the minimum bounding rectangle (MBR) that includes the N results, finding the centroid of the N results, or using a density-based clustering technique such as DBSCAN [99] on the N results. Without loss of generality, Stella chooses to report back the MBR of the answers as the location of O.

Meanwhile, the confidence value of each iteration, i.e., the *local confidence*, is computed as the ratio of workers who vote for the new search space over N. With a pyramid of height H, there will be a total of H local confidence values. At the final iteration,

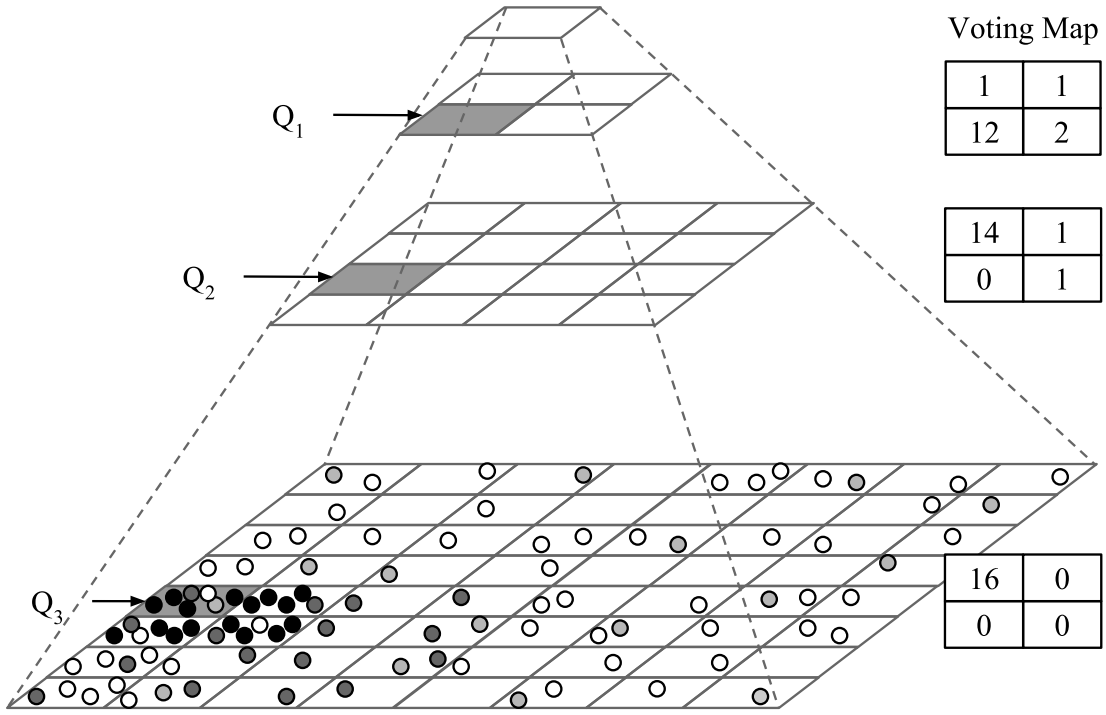


Figure 6.4: Basic Stella Example

we calculate the *overall confidence* as the geometric mean [100] of the H local confidence values. We decide to use geometric mean as it has been widely used to compare values that has different properties, e.g., in calculating the citation impact of research articles [100].

Detailed Example

Figure 6.4 gives an example of our basic Stella approach, where the set of available workers are depicted by circles (regardless of their colors) in the lowest pyramid level. With a pyramid height of four levels ($H = 4$) and budget $B = 64$, there is a total of four iterations with 16 workers in each iteration ($N = 16$). At the start, the initial search space S is set as the root of the pyramid. The first iteration selects 16 workers uniformly distributed over S by picking one from each of the 16 cells at the third level (depicted as light gray circles). Then, these workers vote on which quarter O is located. An example of the voting result is depicted on the right of the second pyramid level. We select

quarter Q_1 (depicted as gray cell) as the new S with local confidence of $12/16 = 75\%$.

In the second iteration, we select another 16 workers all within Q_1 by selecting one worker from each of the 16 cells in the lowest pyramid level that are the grandchildren of Q_1 (depicted as dark gray circles). The voting result is shown next to level 3 with a local confidence of $14/16 = 87.5\%$. In the third iteration, we select another 16 workers within Q_2. Since we only have one level left at the end, we end up selecting four workers from each of the four children of Q_2 (depicted by the black circles). Again, all the 16 workers agree that O is located in Q_3 with a local confidence of 100%. In the final iteration, we recruit our last batch of 16 workers (making a total of 64 workers) which all selected randomly from Q_3 (these workers are not depicted in the figure). Then, we report back the image location as the MBR that includes all exact locations returned from these last 16 workers with an overall confidence value of $(12/16 \times 14/16 \times 16/16 \times 16/16)^{\frac{1}{4}} = 90\%$.

6.4 Optimizing Stella

This section presents our *optimized* Stella; where we deploy four techniques to optimize our basic approach. All optimizations are generally geared towards ensuring that the recruited N workers are more domestic. Algorithm 8 gives the pseudo code of our optimized Stella, which follows the same skeleton of our basic Stella with the added optimizations annotated by comments.

Optimization 1: Domestic Workers

Objective. The objective of this optimization is to fully deploy the domestic worker concept behind Stella. Recruiting all workers uniformly distributed over the whole space will give low quality answer as there are only a small part of workers that are domestic to the image while other workers have a far distance to the image. That is why we only have subset of our workers chosen uniformly, then, we narrow down the search space, and recruit workers uniformly from the new smaller search space. In this optimization, we avoid recruiting workers uniformly from the new smaller search space in every iteration. We would like to always have our workers distributed in a skewed way biased towards the image location.

Main Idea. The main idea is that after retrieving the detailed workers' answers of

Algorithm 8 Optimized Stella

```
1: procedure GEOTAG(Index P, Image O, Budget B)
2:     S ← P.root; C ← 100%; N ← B/P.H
3:     L ← Empty List;          /* OPTIMIZATIONS 1, 2 */
4:     while S do
5:         if SkipIteration(L, S) then /*OPTIMIZATION 2*/
6:             (S, C, N) ← OfflineCalculation(S, L)
7:             continue;
8:         end if
           /* Step 1: OPTIMIZATIONS 1, 4 */
9:         W ← SelectDomesticWorkers(N, S, L)
           /* Step 2: OPTIMIZATIONS 1, 2, 3, 4 */
10:        (S, O.loc, C, L) ← ResultCalculation(W)
11:    end while
12:    return O.loc, C
13: end procedure
```

exact (*latitude*, *longitude*) from an iteration, Stella passes this information to the next one. Then, the next iteration will use this information to select N workers that match the spatial distribution of the exact locations from the previous iterations by mapping them into the children of S rather than selecting them uniformly. With this, we are taking advantage of every bit of workers' answers that we have to predict the location of O to recruit workers where O is more likely to be.

Algorithm. We make the following three modifications which are annotated by OPTIMIZATION 1 in the algorithm: (1) We first initialize an empty list L to be used as a buffer to store the N workers' (*latitude*, *longitude*) answers. (2) We modify Step 1 to take an additional parameter L, and use it as a guide for the distribution of the N workers to be recruited within S. Since L is initially empty, the first iteration will select the N workers in a uniform way. (3) We modify Step 2 to return a fourth output, which is an updated list L that contains the N current iteration results. The list L and new search space S will be passed to the next iteration. The first step of the next iteration will traverse $\lfloor \log_4 N \rfloor$ level(s) and make use of L to decide on number of workers to get from each pyramid cell.

Example. Figure 6.5 shows the effect of this optimization on the example of Figure 6.4 with $H = 4$ and $N = 16$. In the first iteration, the first step of the algorithm is similar

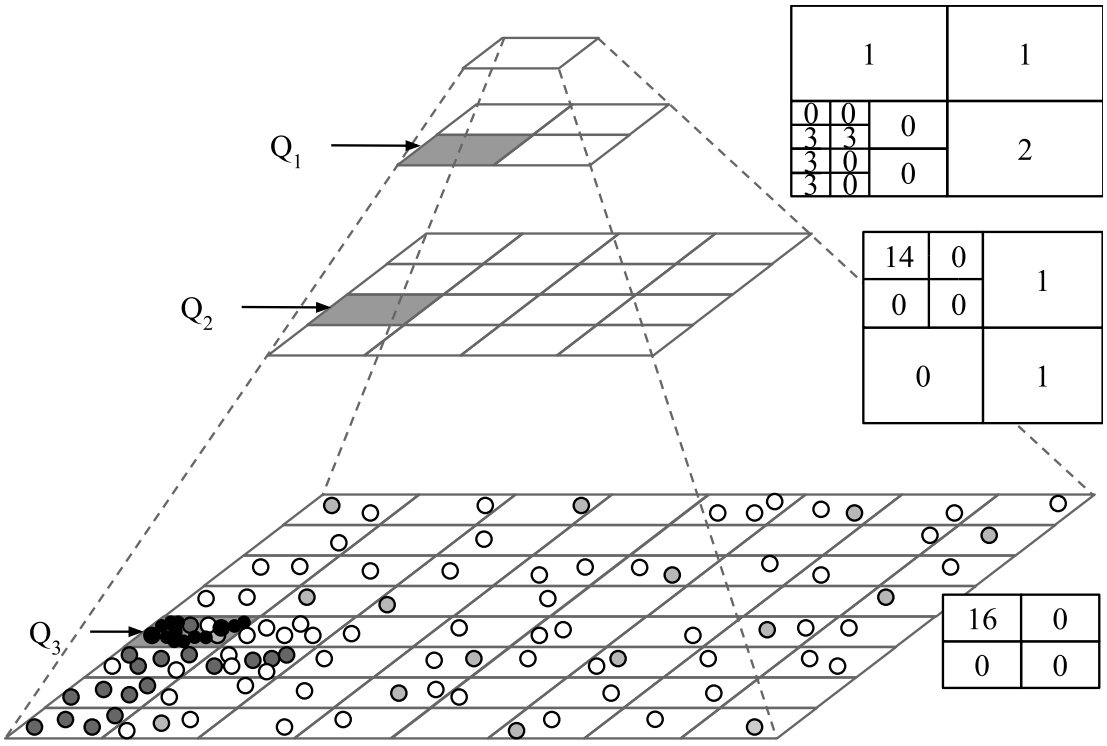


Figure 6.5: Optimized Stella Example

to basic Stella as we do not have any prior results. The second step of the algorithm comes up with a more detailed voting box that votes on the grandchildren of Q_1 instead of the children of Q_1. In the second iteration, we recruit four workers from the four non-zero voted cells within Q_1 and none from other cells (depicted by dark gray circles). By contrasting Figures 6.5 and 6.4 for the dark gray circles, it is very clear that the recruited workers in the optimized Stella are much more localized than the ones in the basic Stella. Then, another detailed voting box is produced from this iteration, yet, with only votes on the children of Q_2, as there is only one pyramid level below it. In the third iteration, we recruit $\lfloor 14/14 \times N \rfloor = 16$ workers from top left child (depicted by black circles). It can be visually seen that there are more black circles in Q_3, namely 16, in Figure 6.5 than that of Figure 6.4, namely four.

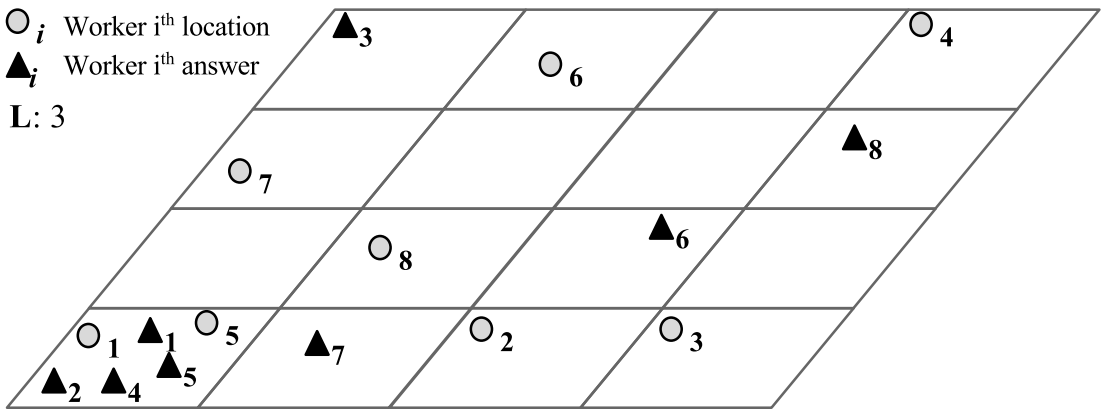


Figure 6.6: Optimization 3 Example

Optimization 2: Skipping Iterations

Objective. The objective of this optimization is to take advantage of cases where there are kind of an agreement on the whereabouts of the image O in one of the iterations and skip the iteration. Then, we reuse the budget from every skipped iteration to recruit more domestic workers in further iterations. For example, in geotagging an image of the Statue of Liberty in New York City, USA, it is likely that we can infer the city directly regardless of its exact location only from the first iteration. Thus, we can skip multiple iterations and use the skipped budget to recruit more domestic workers directly within the New York City to find the image exact location.

Main Idea. The main idea behind this optimization is to check on whether or not the previous iteration results agree on a certain child of S. To do so, we identify if there exists an outlier cell among S children that contains most workers' answers from the previous iteration. In Stella, we go with a simple high majority formula where, among the four children of S, we consider that a certain cell Q is an outlier cell if it has more than $x\%$ of the votes, with x is a system parameter with a default value of $x = 80\%$. If there is such an outlier quarter Q, we reuse the results from the previous iteration as the results that are coming from workers in Q in the current iteration. Reusing these results saves us a budget of N workers that we can use in further iterations. However, we would still need to make *offline* computation to update the local confidence value for every skipped iteration. The number of workers of any skipped iterations will be

distributed equally over the forthcoming iterations, which will make these workers more domestic than the case of having them in their original skipped iteration(s).

Algorithm. We make the following three modifications which are annotated by OPTIMIZATION 2 in the algorithm: (1) We initialize an empty list L to store N workers' answers, which we will use to decide on whether or not we can skip an iteration. (2) We start each iteration by performing a test that takes L and S as the inputs to check whether or not we can skip the current iteration. In the first iteration, the test will return *false* as L is still empty. In further iterations, the test will return *true* if and only if it finds an outlier cell. In that case, we skip the current iteration. Instead, we will only make offline calculation to update the search space S, confidence C, and number of workers N. (3) We will need to return the updated list L as the fourth output to be used in the next iteration.

Example. Consider the example in Figure 6.5 with $H = 4$ and $B = 64$. In the first iteration, the algorithm will proceed the same way as before. In the second iteration, we have the number of workers voting for each child of Q_1 where the two left children of Q_1 receive 6 votes each. Applying our threshold value of $x = 80\%$, we find that no child in Q_1 has more than $x\%$ of the total votes, so we proceed as usual. In the third iteration, the number of workers voting of each child of Q_2 is (14, 0, 0, 0), with the top left child of Q_2 having a clear higher majority, i.e., $14/16 = 87.5\%$ of the total votes which is more than 80%. So, we skip this iteration and do not hire any workers here. Instead, we only do offline computation to update S to be Q_3, N to be 32, the local confidence value of 87.5%. As the next iteration is the final one, we recruit 32 workers from Q_3. This will clearly give a more accurate result as we will be recruiting workers that are more domestic than the original ones with the new overall confidence of: $(12/16 \times 14/16 \times 14/16 \times 32/32)^{\frac{1}{4}} = 87\%$.

Optimization 3: Weighted Confidence

Objective. The main objective of this optimization is to increase the confidence of Stella. Based on our hypothesis of domestic workers, there is a higher probability that a worker will return an accurate location when the worker selects his nearby locations rather than further locations. Thus, instead of valuing all workers' answers equally, we should assign a higher weight to an answer that is closely located with the worker's

location, thus increasing the confidence of Stella. For example, when a worker who is living in Minneapolis geotags O to be in Minneapolis, it's more likely that the worker is more confident than geotagging O to be in Seattle.

Main Idea. The main idea behind this optimization is to weight each worker's answer based on the distance between the worker's location and his answer. The closer the distance is, the higher the weight will be. Without loss of generality, we use a simple cell distance to calculate the distance between two cells. For each worker w, the weight of w's answer is calculated as: $d_{max} - d_{actual} + 1$ where d_{actual} is the cell distance between the worker and his answer and d_{max} is the maximum cell distance at any pyramid levels, e.g., $d_{max} = 2$ in level 2 and $d_{max} = 6$ in level 3. Then, these weights are used to calculate the local confidence in each iteration as the ratio between the weighted sum of answers in the selected cell to the total weighted sum of all answers in this iteration.

Algorithm. The only modification for this optimization (annotated by OPTIMIZATION 3 in the algorithm) is that Step 2 of the algorithm will make use of the location of every worker it received from Step 1. We will retrieve the exact location of each worker that is stored in the index P and use it to calculate the weight of her answer. Similarly, the local confidence and overall confidence values of the iteration is calculated based on the total weighted answer on the new search space over the total weight of the iteration.

Example. Figure 6.6 gives an example that illustrates this optimization, where we only show the third level of a pyramid index. Here, we are recruiting eight workers, depicted by the gray circles, who have reported eight locations, depicted by the black triangles. For example, worker w_4 is physically located in the top right cell and has reported the image location in the bottom left cell. In this level, we have $d_{max} = 6$. Then, each worker answer will be weighted differently. For example, w_1's answer is located at the same cell as its location, i.e., $d_{actual} = 0$, hence, w_1's answer is weighted as $d_{max} - d_{actual} + 1 = 7$. Going on, w_2, w_3, w_4, w_5, w_6, w_7, and w_8 answers will be weighted as 5, 1, 1, 7, 4, 4, and 4, respectively. Out of these 33 total weights, 20 of them are for the bottom left cell (answers of w_1, w_2, w_4, and w_5). Hence, the local confidence value is $20/33 = 60\%$, which is higher than basic Stella (50%).

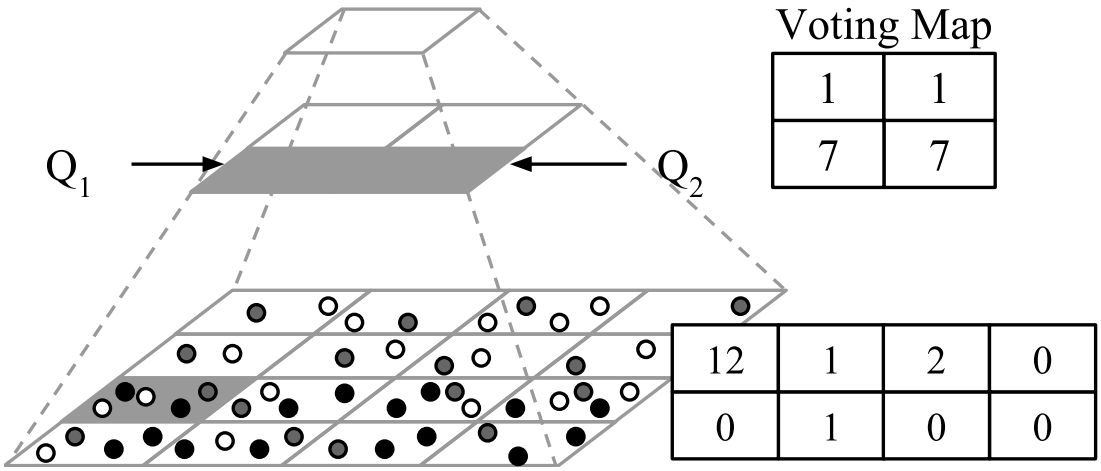


Figure 6.7: Optimization 4 Example

Optimization 4: Widening Search Space

Objective. The objective of this optimization is to avoid the case where there is low confidence when identifying an image location. For example, consider the case where we have 100 workers who cast their votes on the four quarters that are the children of a cell as follows: (50, 45, 3, 2). In basic Stella, we will just select the first quarter as it has the majority of answers, namely 50, though the confidence is only 50%. For a case like this, it will be better if we expand our search horizon to also include the second quarter that receives 45 votes, as this quarter is still promising to contain the image. Should we be able to widen our new search space to include the first two quarters, we would be able to have a 95% confidence instead of the 50% confidence as in basic Stella.

Main Idea. The main idea of this optimization is to go beyond the idea of narrowing down the search space S to only one quarter cell. Instead, we select all quarter(s) that contain more votes than the average votes per quarter as the new search space S, i.e., a quarter will be included in the new S if it has more than 25% of the votes.

Algorithm. We make the following three modifications which are annotated by OPTIMIZATION 4 in the algorithm: (1) Step 1 is modified to select its workers from multiple cells rather than from only one cell as it was the case in basic Stella. In particular, the first action of this step is to divide the number of workers N by the number of cells in S to set the number of workers to be recruited from each cell in S. Then, selecting

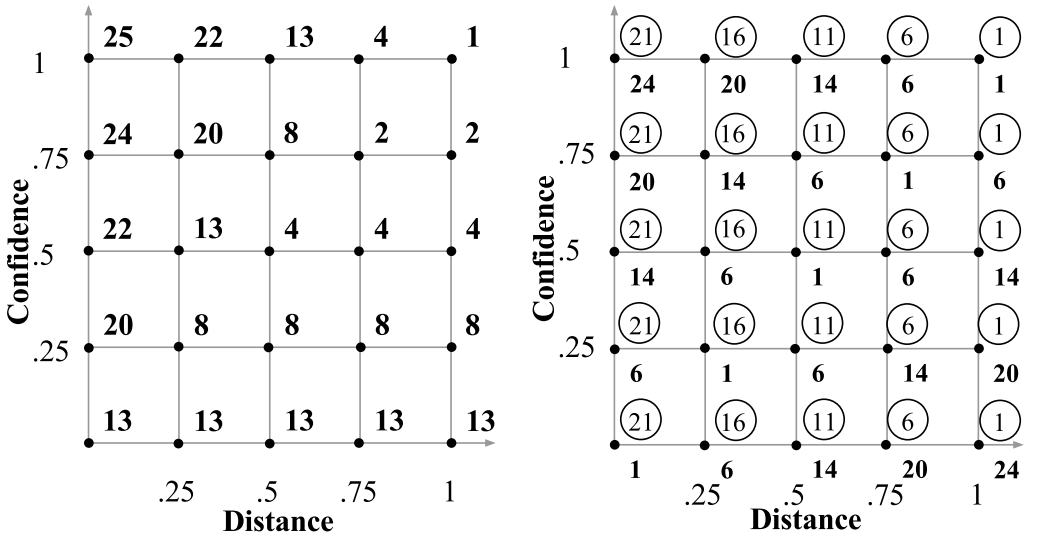
the set of workers from each cell in S is done in the same way as in basic Stella or in Optimization 1 of our optimized Stella. (2) Step 2 is modified to return back multiple cells inside S, rather than only one cell as it was in basic Stella. Those multiples cells in S are the ones that have more than the average votes per cell.

Example. Figure 6.7 gives an example of this optimization by using a pyramid of $H = 3$ and $N = 16$. The first root iteration will have 16 workers, depicted as gray circles, who will vote on quarters as (7,7,1,1). Since two of these quarters have more than the average votes per quarter (i.e., 4), we select these two quarters, namely Q_1 and Q_2, for the new S with a local confidence of $(7 + 7)/16 = 87.5\%$. In the second iteration, we allocate only eight workers to each of Q_1 and Q_2, i.e., two workers from each of the eight children of Q_1 and Q_2, depicted as black circles. The voting result is shown in the figure next to the lowest level, where only one cell, Q_3, is selected since it is the only one with above average votes. The third iteration finds the exact location of the image and returns the overall confidence of $(14/16 \times 12/16 \times 1)^{\frac{1}{3}} = 87\%$.

6.5 Evaluating Stella

One may evaluate the quality of a geotagging process based on how close is the resulting location to the actual image location. The closer the distance is, the better the accuracy. However, Stella returns the confidence of the answer in addition to the answer itself to help users to get a good idea on how much they can trust the answer. Yet, this makes it more challenging to evaluate the outcome of Stella. For example, it is clear that the best possible outcome of Stella is to have the exact correct location of an image with 100% confidence. Meanwhile, having a high confidence is not valuable, unless it comes with a highly accurate location as it can mislead the user to trust inaccurate result. For example, it is preferred to have both inaccurate location and confidence rather than inaccurate location with high confidence.

Figure 6.8(a) illustrates a spectrum of 25 possible answers and how we would evaluate each of them. The x-axis represents the distance accuracy of Stella answer, where 0 is the worst and 1 is the best. The y-axis represents the confidence returned from Stella, also ranging from 0 to 1. The 25 answer points depicted by black circles represent all the 25 combinations of distance accuracies and confidence values. On top of each



(a) Overall Accuracy Chart (b) Dist & Confidence Chart

Figure 6.8: Stella Accuracy Chart

point, we plot its ranking, with 1 is the best and 25 is the worst. For example, the top rightmost answer (1,1) has the best distance accuracy and confidence. The worst ranked answer (rank 25^{th}) is at (0,1), i.e., a very inaccurate answer with 100% confidence as the confidence misleads the user to trust inaccurate result. The rest of the section describes the concept of *distance accuracy*, *confidence accuracy*, and how we use both accuracies to come up with the *overall accuracy* of Stella, which is basically depicted by the ranking order in Figure 6.8(a).

Distance Accuracy

Distance accuracy evaluates the spatial location of an image O returned by Stella to the $truth$ location of O where the closer the distance is, the better the accuracy, regardless of the reported confidence. In the case when an answer is returned as an area, we consider $O.loc$ as the centroid of the area. We use a simple linear regression model to calculate the distance accuracy as: $1 - dist(O.loc, O_{actual})/d_{max}$ where $dist()$ is a euclidean distance function between two points and d_{max} is a system parameter that acts as a normalization upper bound distance. If $dist(O.loc, O_{actual})$ is greater than d_{max}, then the distance accuracy is set to 0. Figure 6.8(b) replicates the same setting

of Figure 6.8(a) while the ranking of the possible answers is only based on the distance accuracy (depicted as numbers inside circles).

Confidence Accuracy

Given a distance accuracy and a confidence, confidence accuracy measures how accurate the confidence reflects the distance accuracy. For example, a high/low confidence with an accurate/inaccurate distance accuracy should result in a good confidence accuracy. Hence, the confidence accuracy is calculated as: $1 - |DistAccuracy - C|$, where C is the output confidence. This means that if a distance accuracy of 1 with 100% confidence, the confidence accuracy will be at its maximum value of 1. Similarly, with every combination that has equal distance accuracy and confidence. Meanwhile, the worst confidence accuracy is when a distance accuracy of 0 and C is 100% or when a distance accuracy of 1 and C is 0%. Figure 6.8(b) also shows the ranking based only on the confidence accuracy (depicted as bold number).

Overall Accuracy

Relying solely on distance accuracy allows two answers with the same distance from the image location to have the same quality, regardless of the resulting confidence. On one hand, if Stella reports an inaccurate location, it is strongly preferable that it gives low confidence. On the other hand, relying solely on confidence accuracy would equate two answers with same accuracy regardless of their distance accuracy. For example, in Figure 6.8(b), the points (0,0) and (1,1) have the same confidence accuracy, however, the latter one is strongly preferred as it provides the accurate location. We evaluate the *overall accuracy* as a linear combination of both accuracies as: $\alpha \times$ *DistAccuracy* + (1 - α) $\times$ *ConfAccuracy*, where $0 \leq \alpha \leq 1$ is a system parameter to weight the importance of each accuracy. The overall ranking depicted in Figure 6.8(a) is calculated based on having $\alpha = 0.5$.

6.6 Experiment

This section presents our experiment of Stella: basic Stella (termed Stella) and the optimized Stella (termed Stella+). To the best of our knowledge, Stella is the first

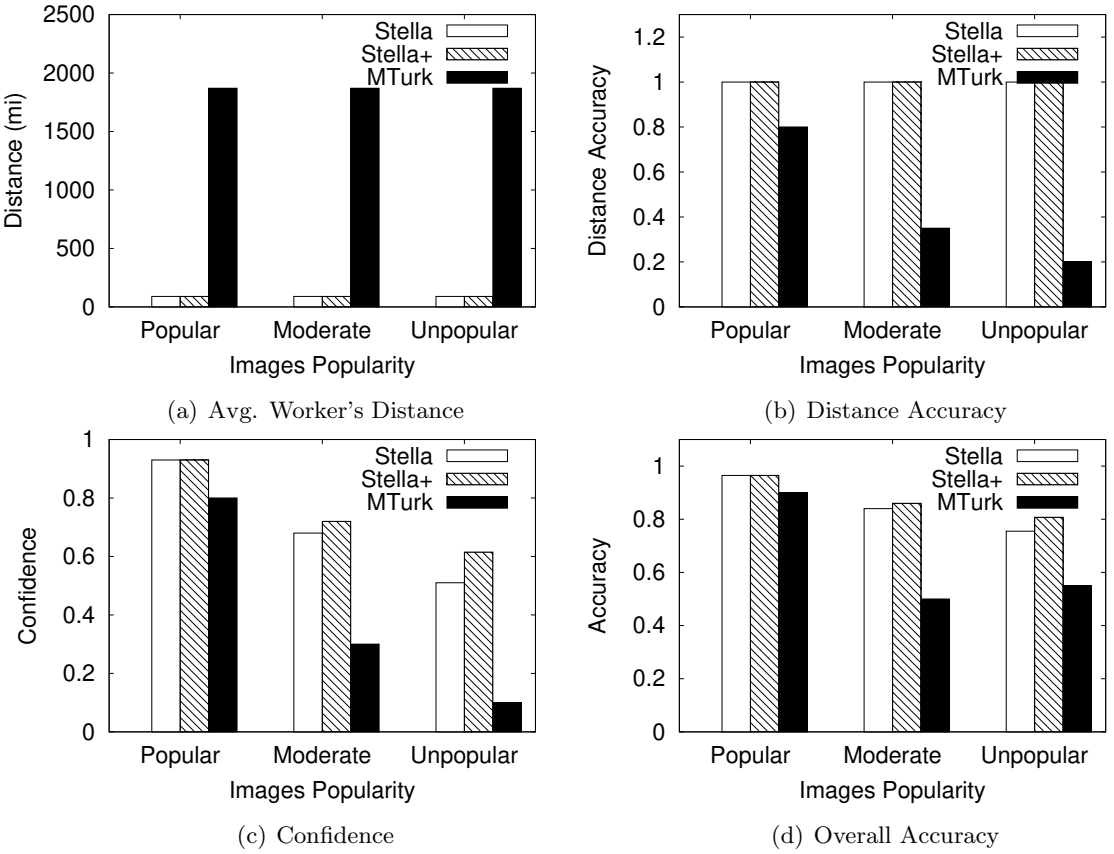
framework that solves the geotagging problem without relying on having a rich set of training dataset. Hence, it is not comparable to other geotagging frameworks. Instead, we first evaluate the idea of Stella by deploying it on top of a commercial crowdsourcing platform, namely Amazon Mechanical Turk (termed MTurk), to compare its performance to the general deployment of Amazon Mechanical Turk. Then, we evaluate the impact of each optimization of Stella with real dataset.

In all our experiments, we use four evaluation metrics: (1) The average distance between the worker location and the true location of an image O. This gives an indication on how successful each technique is in recruiting domestic workers. (2) The distance accuracy, i.e., how far is the reported location of an image O from its true location. (3) The confidence value, where the higher the confidence is the better. This gives an indication of how each approach is confident with its result. (4) The overall accuracy, which combines both the distance and confidence accuracies.

Note that we do not include latency in our evaluation metric. The main reason is that Stella is more about batch processing of geotagging and is not a real-time geotagging framework. While Stella may incur extra penalty in latency in comparison with other techniques, i.e., *upper-bounded* by the number of iterations $\times$ the latency of a single crowdsourcing deployment, it is currently the only framework that is able to geotag images with high accuracy while other approaches fail to do so. From our real experiment with Amazon Mechanical Turk, Stella often completes the geotagging task much earlier than its upper bound as we only deploy a subset of the total workers in each iteration, thus, resulting in less turn around time. Furthermore, for some images, Stella+ is able to minimize the latency further due to its second optimization.

Amazon Mechanical Turk Deployment

Experimental setup. We selected 20 images and geotagged them in a total of 150+ crowdsourcing tasks by using 600+ workers with a reward of \$0.05 per assignment. To ease the readability, based on the results, we categorize every image into three categories: *popular* image, *moderately popular* image, and *unpopular* image. A popular image is an image that more than 50% of the workers agree on its exact location, and thus MTurk can geotag the image accurately by using majority voting on the results. A moderately popular image is an image that more than 50% of workers are aware with its location,



(a) Avg. Worker's Distance

(b) Distance Accuracy

(c) Confidence

(d) Overall Accuracy

Figure 6.9: Amazon Mechanical Turk Deployment

e.g., they are aware in which state of the US that the image is located. We categorize the rest as unpopular images.

We first try to geotag every image using a major web search engine. However, it was only able to find the location of popular images, thus, having 0% accuracy for both moderately popular and unpopular images. To ensure a fair comparison, we simulated both Stella approaches using a real deployment on top of MTurk. Since MTurk only allows us to select workers from a certain country or state if the workers live in the US, we simulated Stella in a total of three iterations: (1) We recruit workers within the US and map their results into four non-overlapping regions. (2) We recruit workers all from the majority region to find the state of the image. (3) We recruit workers all from the majority state to find the city of the image. We also ask the workers to provide their

zip-code to see how domestic the recruited workers are. For each image, we use a total of 30 workers to geotag with all three approaches.

Experimental Results. Figure 6.9(a) gives the average worker's distance for each approach. Both Stella and Stella+ use the average distance of the workers that we recruit at the last iteration as they are the ones that provide the final location of the images. For every image popularity, Stella and Stella+ are able to recruit significantly more domestic workers than MTurk. Specifically, both Stella variants and MTurk are able to recruit workers with an average distance of 88 miles and 1,879 miles, respectively. There is no performance difference between Stella and Stella+ as the interface of MTurk only allows us to recruit workers within a state, thus, they recruit a similar subset of workers for every image.

Figure 6.9(b) gives the distance accuracy in geotagging images across various image popularities. We set the maximum distance threshold, d_{max}, as the diagonal length of the state where each image is located. Instead of finding the centroid of an MBR that covers all workers' answers, we calculate the distance accuracy of MTurk as the average distance accuracy among the recruited workers. The reason is that the MTurk answers often give different locations which increases the size of the MBR that covers all of those answers. In such case, the centroid of the MBR is no longer represents the workers' answers accurately. For example, if 9 out of 10 workers choose Chicago, IL, and 1 worker chooses Boston, MA, as the location of an image, then the centroid of the MBR that covers all 10 answers is located somewhere around Cleveland, OH. As shown in the figure, both Stella and Stella+ significantly outperform MTurk for all image popularities. For popular images, while 80% of MTurk workers are able to provide the accurate city of the images, the rest of the workers provided locations that are very far. Furthermore, the distance accuracy of MTurk is worsen for both moderately popular and unpopular images where it has an accuracy of 35% and 20%, respectively. In contrast, Stella and Stella+ are able to maintain their distance accuracy across all image popularities. There is no difference in the distance accuracy between Stella and Stella+ as they are able to narrow down the accurate state of the images and recruit domestic workers to provide each image final location.

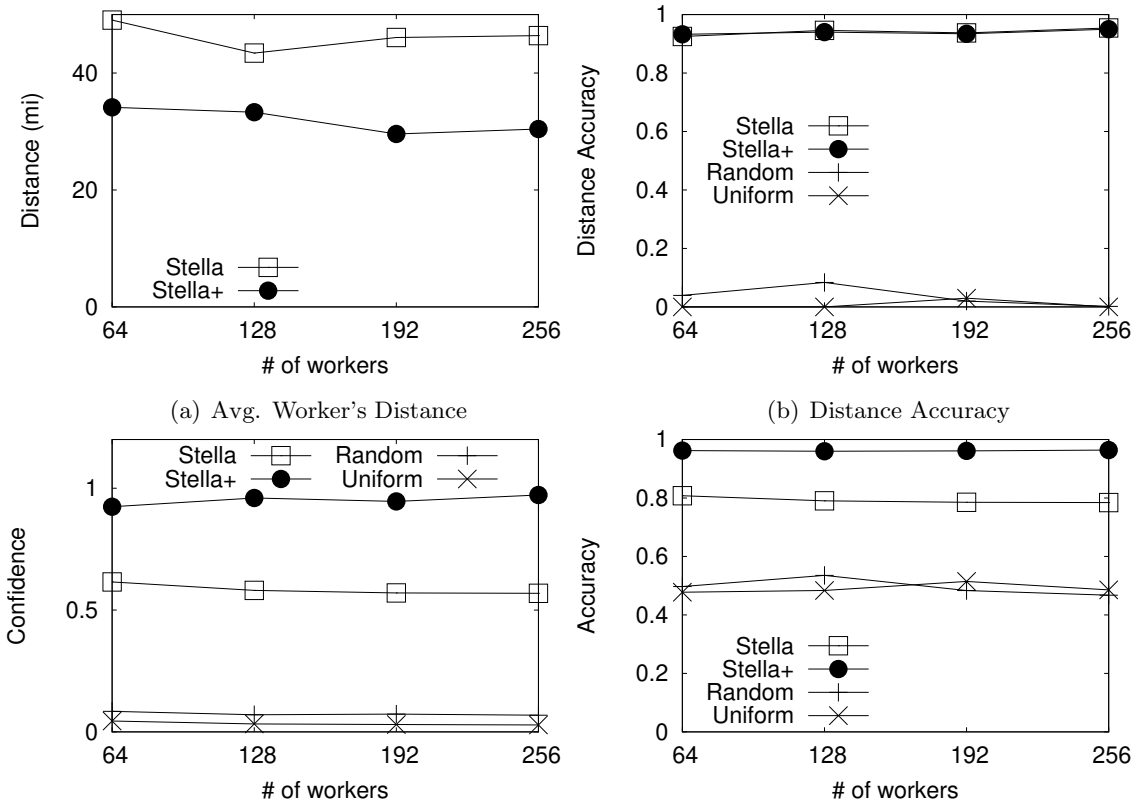
Figure 6.9(c) and Figure 6.9(d) give the confidence and overall accuracy of each approach across different image popularities, respectively. The confidence of MTurk

is calculated as the ratio of workers who agree on the final answer within a city level over the total number of answers. Both Stella and Stella+ consistently outperform MTurk, as they have more agreement among workers. The improvement of Stella+ in all three image popularities is mainly due to its third optimization that assigns a different weight for each worker's answer. In Figure 6.9(d), we can see that Stella and Stella+ consistently have higher overall accuracy than MTurk. However, we also see that the overall accuracy of MTurk is better than its distance accuracy (Figure 6.10(b)) and confidence (Figure 6.10(c)). The reason is that MTurk has a good confidence accuracy where it reports a low accuracy result when it is not confident in its result. Contrast such result to both Stella approaches where they have a high overall accuracy due to its high distance accuracy and confidence.

Standalone Stella Deployment

Experimental setup. Due to the limitation of the Amazon Mechanical Turk interface, we need to run our detailed evaluation of Stella in a different environment. Similar to prior research that used real non-crowdsourcing datasets as workers' locations [1], we use Foursquare dataset [101], consists of 1.7+ million users, as the workers that we can recruit. Then, we follow the *distance-aware quality* model, introduced in [1], which models the accuracy of a worker based on an exponential function. In particular, the input to the function is the distance between a worker w to an image, d_w, and the function will output the distance between w's answer to the image, a_w. $a_w = 0$ means that the answer of w is located at a zero distance from the image location, thus, is highly accurate. We generate such function by first plotting the accuracy of the 600+ workers from our Amazon Mechanical Turk experiment where the x-axis is the d_w and the y-axis is the a_w. Then, we generate an exponential trend line that fits the plot which results in an exponential function of: $a_w = 1.431e^{0.059d_w} - 1$.

Unless mentioned otherwise, we use a total of 128 workers, a pyramid index of six levels which covers the mainland of USA, and a default $\alpha = 0.5$ for our accuracy evaluation to weight distance and confidence accuracy equally. For our second optimization, we set $x = 80\%$. To minimize the inconsistency that may be resulted, we run each experiment 100× on a machine with Intel Quad Core i7-4790 3.6Ghz, two threads per core, and 32 GB of RAM running 64-bit Ubuntu 16.04.



(a) Avg. Worker's Distance

(b) Distance Accuracy

(c) Confidence

(d) Overall Accuracy

Figure 6.10: Stella's Overall Performance

Overall Performance. We compare the end to end performance of Stella and Stella+, with two basic crowdsourcing techniques: *random* and *uniform*. Random selects workers randomly. Meanwhile, uniform first divides the space into equal grids, where the number of grid cells is equal to the number of workers that it will assign, then it randomly selects one worker within each cell. For each experiment, we randomly select a location as the image location and varies the number of workers from 64 workers to 256 workers.

Figure 6.10(a) gives the average recruited worker's distance to an image. Stella and Stella+ recruit workers with an average distance of 46 and 32 miles, respectively. We omit the the average worker's distance of the two basic approaches as random and uniform has a very large distance of 958 and 1,341 miles, respectively. Stella+ is consistently able to recruit 30% more domestic workers than Stella which is resulted

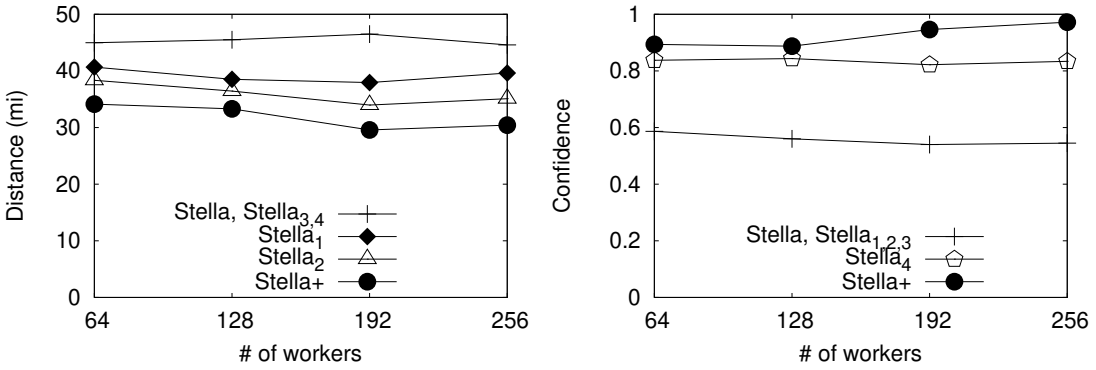
from its first two optimizations. We also see that there is not much difference in the average recruited worker's distance when varying the number of workers as both Stella approaches still recruit workers in the same search space.

Figure 6.10(b) gives the distance accuracy of every approach. Both Stella approaches are consistently able to outperform both basic approaches by having an average 90% more accuracy than random and uniform. Both Stella and Stella+ also have a similar distance accuracy between each other. The reason is that distance accuracy is calculated only based on the last iteration results and since both approaches are able to narrow down search space to the accurate one, there is no difference in their distance accuracy.

Figure 6.10(c) gives the overall confidence of every approach. We calculate the confidence of the two basic approaches by projecting their results into the lowest pyramid level and taking the ratio of workers' answers that fall into the cell that contains the majority of answers. As we can see from the figure, both Stella and Stella+ outperform the two basic approaches by achieving 50% and 90% more confidence, respectively. Stella+ outperforms Stella by having 40% more confidence as Optimizations 1 and 2 allow Stella+ to recruit more domestic workers while Optimization 3 gives higher weight to their answers. The confidence also directly impact the overall accuracy of each approach as can be seen in Figure 6.10(d), where Stella, Stella+, and the two basic approaches have 96%, 79%, and 50% overall accuracy, respectively.

Internal Performance. We compare six variants of Stella in this experiment: the basic Stella, the optimized Stella+, Stella with Optimization 1 only (Stella_1), with Optimization 2 only (Stella_2), and so on. For ease of readability, we combine several variants of Stella into one series if they have a similar performance. In this experiment, we also study the effect of varying the number of iterations of each approach in addition to varying the number of workers.

Figure 6.11(a) gives the average worker's distance to an image by varying the number of workers. Both Stella_1 and Stella_2 are able to assign more domestic workers than basic Stella. This confirms our idea of Optimization 1 that by recruiting workers distributed in a skewed way that matches the answers distribution, we will be able to get more domestic workers. Optimization 2 is able to assign more domestic workers than Stella as it is able to skip some of the iterations and use the skipped budget for further iterations to recruit 25% more domestic workers. Combining these two optimization results in a



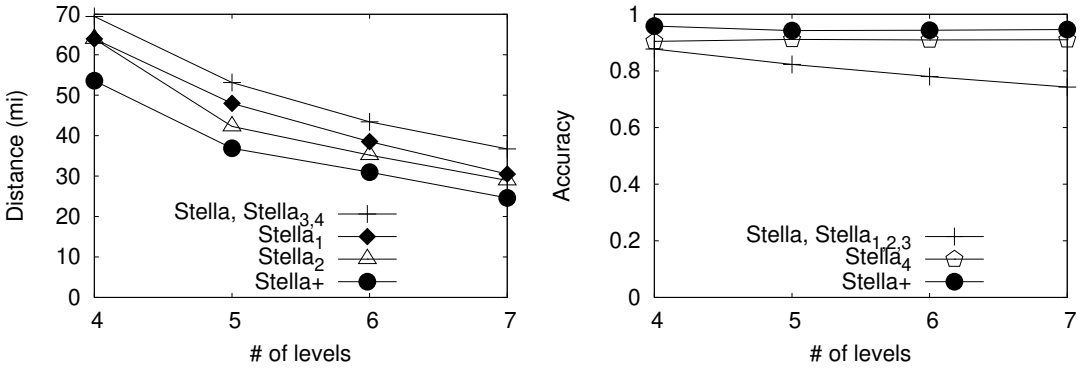
(a) Avg. Worker's Distance (b) Confidence

Figure 6.11: Internal Performance Varying # Workers

better performance as shown by Stella+ which outperforms Stella by recruiting 35% more domestic workers.

Figure 6.11(b) gives the overall confidence for each variant of Stella. Stella_4 is able to achieve 20% more confidence than Stella as it is able to select more than one cell as the new search space. However, Stella_3 does not have any impact to the overall confidence. The reason is that there is less agreement between workers at the lower pyramid level which is caused by from the accuracy model that we extract. For example, even with $d_w = 0$, the answer will have an error distance of $a_w = 0.431$ in lat/lon degree which, in our case, is roughly 23.4 miles while each cell has a diameter of 55 miles. However, by combining Optimizations 3 and 4 together, we achieve a better confidence as shown by Stella+. The reason is that Optimization 3 now has an impact in calculating the confidence as workers' answers can agree on more than one cell due to Optimization 4.

Figure 6.12(a) gives the average worker's distance that each variant of Stella assigns by varying the number of iterations. All variants are able to recruit more domestic workers as they traverse deeper into the pyramid which confirms our adaptive crowdsourcing concept. Both Stella_1 and Stella_2 are able to assign more domestic workers than Stella due to their optimizations which are geared towards recruiting more domestic workers. Meanwhile, Stella+ outperforms other variants of Stella as its third optimization weights those domestic workers' answers more. Figure 6.12(b) gives the overall accuracy of all variants. The overall accuracy of Stella, Stella_1, Stella_2, and Stella_3 decrease with deeper pyramid levels as there is less agreement between workers



(a) Avg. Worker's Distance (b) Overall Accuracy

Figure 6.12: Internal Performance Varying # Levels

due to the the accuracy model that we extract. However, Stella_4 is able to maintain its accuracy as it expands its search space in the lower pyramid levels. However, by combining Optimizations 3 and 4 together, Stella+ consistently outperforms Stella_4.

6.7 Related Work

This chapter highlights the related work to Stella in three directions, namely, crowdsourcing frameworks, spatial crowdsourcing, and object geotagging techniques:

Crowdsourcing Frameworks. Many crowdsourcing efforts are focused on providing efficient techniques to solve different types of machine-hard tasks. Examples include integrating crowdsourcing into the query plan of a Database [102, 103], using the crowd to sort and join data [103], to compute skyline over noisy dataset [104], for real-time image search [96], and many more. However, up to our knowledge, crowdsourcing has not been used for geotagging as it is first deemed unfit to be solved by the crowd due to the low accuracy of the result. In this paper, we provide the first framework that leverages crowdsourcing for geotagging. Task assignment, i.e., studying on how to assign a task to a set of workers, is a very important problem and has been widely studied [4,8,105]. [8] assigns a task to a worker that has completed similar tasks with high performance. In geotagging, the only information to predict such performance metric is to check the worker's performance in solving other tasks that are co-located with the new task. However, we do not know the location of the new task and in fact, this is

what we are trying to find. [4] assigns tasks that results in the highest improvement in quality by representing the possible answer of a task in a matrix. This approach will only work when there is a limited amount of answers for a task, e.g., a label of "equal" or "not equal" in an entity resolution, which is not the case in geotagging as every location can be treated as an answer. Our task assignment technique is closely related to [105] where it tries to maximize the overall utility, i.e., the location coverage in our case, given a fixed budget. The main difference is that [105] spends all of its budget at once and thus, the best result it can get is the first iteration of Stella.

Spatial Crowdsourcing. Several works have been conducted in studying spatial crowdsourcing where each crowdsourced task contains a spatial information about the task and/or the workers. Spatial crowdsourcing frameworks [49–52] require workers to physically go to the location of the task, e.g., take a picture of an object. However, these applications use one main assumption that the task's location to be known in advance. Similar to these techniques, our work tries to find the workers who are located near the task. However, without knowing the location of the image in advance, existing techniques are not fit to solve the geotagging problem.

Object Geotagging Techniques. Many works have been conducted to geotag an object by using natural language processing (NLP) [94], incorporating user profile [92], or using machine learning techniques [23]. However, they suffer from two main limitations: (1) Each object type, e.g., text and image, requires its own tailored solution. For example, NLP techniques cannot geotag images while computer vision technique cannot geotag tweets. (2) The answer quality mainly rely on having prior knowledge, e.g., an accurate training dataset. With Stella, we do not need to create a specific solution for each object type and we do not rely on having a large accurate training datasets. In fact, some of these approaches can use Stella to create an accurate training dataset for their techniques.

Chapter 7

Related Work

The recent interest in the capability of crowdsourcing in solving tasks that are still difficult for computers or machines has urged many researchers to propose new research attempts that leverage crowdsourcing. This chapter classifies existing research effort by considering four aspects that are related to our work. (1) The existing crowdsourcing-based systems that are built by integrating crowdsourcing into the query plan of a traditional Database Management System (DBMS) with the goal of enabling DBMS to solve machine-hard queries, e.g., sentiment analysis, in Chapter 7.1. (2) The existing research efforts that introduce new crowdsourcing queries or operations, including their optimization, in Chapter 7.2. (3) The existing research efforts in providing a new mechanism for both task-based and worker-based task assignment approaches in Chapter 7.3. (4) The existing research efforts in providing a new mechanism for result evaluation approaches in Chapter 7.4. (5) The existing approaches which consider and include the workers' expertise within their solutions where the goal is to increase the overall performance of their results in Chapter 7.5.

7.1 Crowdsourcing-based Systems

Recently, there have been multiple commercial crowdsourcing platforms that have been gaining a lot of popularity, including, Amazon Mechanical Turk [5], Figure Eight [6], ChinaCrowds [7], TaskRabbit [106], and Upwork [34]. Such platforms have been providing an interface for requester to post a task that will be solved by the crowd. With the

availability of those platforms, multiple crowdsourcing-based systems [102,103,107–109] have been built on top of those platforms to solve operations that are still difficult for computers or machines to solve, e.g., to extract the sentiment from a corpus of text. In particular, these systems decide to incorporate crowdsourcing into the query plan of traditional Database Management System (DBMS). By doing so, it makes DBMS able to solve such machine-hard queries. In particular, for every query that cannot be solved by the DBMS itself, e.g., entity resolution, the DBMS will outsource the query into one or more crowdsourcing platforms to ask the workers to provide the answers for the query. Internally, these systems provide (a) declarative programming interfaces which allow task requesters to use an SQL-like language for posing queries that involve crowdsourced operators, i.e., `COLLECT`, `FILL`, `SELECT`, and `JOIN` and (b) query optimization that is geared towards optimizing the three optimizations in crowdsourcing, namely, cost, latency, and quality.

CrowdDB [102] and Deco [108] provide the systems that solve all the four crowdsourced operators with the goal of minimizing the cost to be paid to the crowd when solving those operations. Qurk [109] solves the `SELECT` and `JOIN` operators with the goal of minimizing the cost as well. CrowdOP [107] solves the `FILL`, `SELECT`, and `JOIN` operators with the goal of minimizing the cost as well as the latency. Lastly, CDB [103] solves all four crowdsourced operators by optimizing the cost, latency, and quality.

Unfortunately, at the current stage, these crowdsourcing-based DBMS systems are unable to facilitate any expert-sourcing queries due to two main reasons. Firstly, they are not aware of the task and the worker's expertise, thus, the cannot assign task to workers according to the workers' expertise. Making the systems aware of the task required expertise can be done by either having a module that automatically detects the required expertise of a given task or by adding the required expertise of a task into the `WHERE` clause of the SQL statement. Secondly, even after they have added the capability of extracting the required expertise of a task, they are currently deployed on top of general purpose crowdsourcing platforms which randomly assign task and workers. Deploying these systems on top of Luna will give them the capability to handle expert-sourcing queries as Luna will assign each task to expert workers while these systems can focus on providing their two main features, i.e., declarative interface and query optimization.

7.2 Crowdsourcing Queries

In recent years, there have been a growing research interest to provide crowdsourcing-based techniques for queries that are still difficult to be solved by machines. This includes using crowd to sort and join data [3, 32, 103], to annotate data [110, 111], to compute skyline over noisy data [104, 112], to compute top-k queries [36, 113–115], for real-time image search [96], to compute sentiment analysis [4], and many more. A survey of queries that can be solved with crowdsourcing can be found in [28]. These efforts give the many use cases of queries that can be solved with crowdsourcing alongside with their own tailored approach on how to solve them. There are two major contributions that Luna brings into this angle. Firstly, Luna supports a plethora of queries that are originally cannot be supported since they require a certain worker's expertise to solve them, e.g., translation task and geotagging task. Secondly, by deploying those queries on top of Luna, we can achieve a better result than what is achieved originally, e.g., our image-labeling case study. This can be done simply by providing the preferred plug-ins that define the logic of the query into Luna.

7.3 Task Assignment in Crowdsourcing

One of the main research areas that have been widely studied in crowdsourcing is in studying the task assignment problem. Since inherently workers have diverse qualities on solving different tasks, it is important for researchers to study on the best assignment that they can make as a better assignment strategy may lead to a higher quality result. There are two task assignment scenarios that have been studied: (1) worker-based, given a task that is submitted by the requester, which subset of workers should be assigned to solve the task; (2) task-based, given a worker that decides to do a task, which task should be assigned to the worker.

In the worker-based task assignment, there are two factors that have been considered by the previous works in selecting a subset of workers for a given task, namely selecting workers with matching skills to the task [116, 117] and selecting workers based on the known *worker cost* of each worker [118, 119], i.e., the monetary cost that each worker requires to answer a task. In selecting workers with matching skills to the task, existing works have studied on how to build a profile for each worker based on the performance

of the worker when answering a given task. In selecting workers based on the worker cost of each worker, these works study on selecting a subset of workers that maximize the task's quality without exceeding the overall budget since workers with higher quality usually cost more than workers with lower quality.

In the task-based task assignment, multiple previous works decide to assign an incoming worker with tasks that can be benefited in terms of quality by being assigned to the worker based on a given score of each available task [4, 8, 31, 120]. The score can be calculated based on the number of answers that each task currently has, the overall quality of the answers that each task currently has, or based on the quality of the worker. Other techniques use machine learning techniques to assign tasks to a worker based on their trained models [121–123].

Currently, for researchers to create and test a new task assignment approach, they will need to rebuild the whole crowdsourcing system stack from scratch. This is because existing crowdsourcing platforms, e.g., AMT, can only do random task assignment. However, with Luna, it gives the platform for researchers to develop their new task assignment approach by just developing their techniques inside the two plugins of Luna, namely the `TaskRank` and `WorkerRank` functions. In particular, the worker-based task assignment is basically the `TaskRank` ranking function while the task-based task assignment is basically the `WorkerRank` ranking function.

7.4 Result Evaluation in Crowdsourcing

As a single worker may be biased for some tasks, many existing works in crowdsourcing have decided to assign each task to more than one workers, and infer the task result by aggregating the workers' answers for the task. Specifically, such aggregation technique can be seen as a function that takes two inputs: (1) the workers' answers of a task and (2) the quality of each worker who answers the task. Existing works have employed three aggregation strategies, namely, Majority Voting [30, 118], Weighted Majority Voting [2, 121, 124], and Bayesian Voting [4, 29, 31].

Majority Voting basically selects the result that receives the highest number of votes (i.e., the majority of the results). For example, if there are three answers for a labeling task as "dog", "dog", and "cat", then Majority Voting will return "dog" as it

is the majority. Meanwhile, Weighted Majority Voting will consider the weight of each workers' answer (usually based on the worker's quality) and return the answer with the highest weight. Using the same labeling task above, if each of the workers' quality is 0.2, 0.3, and 0.9, respectively, then the Weighted Majority Voting will return "cat" as the label of the task since it has the highest weight, i.e., 0.9 (compared to a total weight of $0.2 + 0.3 = 0.5$ for "dog"). Meanwhile, Bayesian Voting will not only consider each worker's quality but also leverages Bayes' Theorem to compute the probability distribution of each answer being the true answer.

Similar to the contribution that Luna makes to the task assignment research in crowdsourcing above, Luna provides a plugin that can be easily used by researchers to formulate their new aggregation technique without the need of recreating the crowdsourcing system stack via the `Aggregate` aggregation function.

7.5 Expertise in Crowdsourcing

There have been multiple research efforts to incorporate worker's unique features when (a) assigning a task to worker(s) and (b) evaluating the workers' results. This is mainly done to achieve better quality result in comparison to assigning the task to random workers. Most of the these works [1,2,9–11,51,53,125] are mainly focused on solving queries that require the *location* feature of a worker and/or a task, also widely known as spatial crowdsourcing. These works are ranging from providing a tailored task assignment and result evaluation algorithms for spatial crowdsourcing applications [11,51,53,125], ride-sharing frameworks [9,10], labeling frameworks [1,2], and many more. Meanwhile, there have been multiple research efforts [8,126–128] that focus on matching a more general domains, e.g., politics, sports, and economics, between a worker and a task.

Without the existence of any crowdsourcing platforms to support them, each of these works needs to build a new full-fledged system from scratch in order to test its approach while, in fact, they share many common components with each other. For example, every spatial crowdsourcing approach uses spatial expertise for to match their tasks with workers, and some of them are using the nearest neighbor approach when assigning a task to worker(s). With Luna, researchers can focus on creating their own tailored solution to solve the problem, in the form of simple plug-ins, while Luna hides

the complexities of the whole expert-sourcing process. Furthermore, with Luna, many expert-sourcing solutions can be instantiated by just changing one simple plug-in from other existing solutions. For example, solving a image-geotagging problem can now be done by just simply changing the Expertise Index that is used in an image-labeling application into a spatial one.

Chapter 8

Conclusion

This thesis describes Luna; the first extensible expert-sourcing platform. In contrast to existing general purpose crowdsourcing platforms, Luna ensures that a task is assigned to expert workers of that task, and thus Luna is able to solve tasks that are initially deemed unfit to be solved by general purpose crowdsourcing platform and results in more accurate results. Luna realizes its extensibility by allowing the same core algorithm to be used for any expert-sourcing applications without requiring application developers to develop the whole system stack for each application. To deploy a new application on top Luna, developers only need to provide simple plug-ins that will be integrated with the core of Luna to provide the expert-sourcing platform for the new application. We showed the extensibility of Luna by deploying five expert-sourcing applications on top Luna as our case studies, namely domain-specific image-labeling, ride-sharing, translation, image-labeling, and image-geotagging. Experimental evaluation of Luna, in three different expert-sourcing applications deployment with real crowdsourcing scenario and by using real dataset, showed that Luna is able to achieve not only more accurate but also higher quality results than general purpose crowdsourcing platforms in powering those applications.

Chapter 2 discusses the system overview of Luna. Luna is composed of three index structures, namely, Task Index, Worker Index, and Expertise Index, as well as four modules, namely, Submit Task, Request Task, Result Evaluation, and Application Creation. In this chapter, we detail the structure of the three indexes as well as the functionality of each module.

Chapter 3 details our case study of an expert-sourcing problem that is deployed out of Luna, namely domain-specific data-labeling. In particular, we study the configuration of Luna that is made to provide an expert-sourcing platform for the problem. In this chapter, we closely study two variants of the data labeling problem, namely domain-specific image-labeling and image-labeling, as our case studies. Experimental evaluation based on real crowdsourcing deployment shows that Luna is able to outperform existing crowdsourcing platforms in solving the problem by achieving not only higher quality results but also more accurate results.

Chapter 4 details our case study of another expert-sourcing problem that is deployed out of Luna, namely spatial crowdsourcing problem. In this chapter, we first provide the configuration of Luna as the spatial crowdsourcing marketplace for any kinds of spatial crowdsourcing applications to use. Then, we study two spatial crowdsourcing applications as our case studies, namely ride-sharing and image-geotagging. Experimental evaluation based on real crowdsourcing deployment and real dataset shows that Luna is able to solve spatial crowdsourcing problem more efficiently than any existing crowdsourcing platforms.

Chapter 5 details our last case study of another expert-sourcing problem that is deployed out of Luna, namely translation. In this chapter, we provide two different configurations of Luna that can be used to solve the translation problem. Then, we study the advantages and the disadvantages of one configuration to the other.

Chapter 6 details Stella; a full fledged crowdsourcing framework that is fully tailored for crowdsourcing-based image-geotagging application. Stella shows that to be able to solve image-geotagging accurately, we need to recruit workers who are located close the location of the image, i.e., domestic workers. Stella is able to find such domestic workers for each image by using a novel crowdsourcing approach that gradually understands the image location. In addition, Stella has four different optimization to further improve the accuracy of its result.

Chapter 7 gives the related work to our work. In this chapter, we go through five main aspects of related work, namely, crowdsourcing systems, crowdsourcing queries, task assignment in crowdsourcing, result evaluation in crowdsourcing, and expertise in crowdsourcing. It reviews the state-of-the-art approaches in the literature and shows where Luna stands along each of these five aspects.

References

[1] Huiqi Hu, Yudian Zheng, Zhifeng Bao, Guoliang Li, Jianhua Feng, and Reynold Cheng. Crowdsourced poi labelling: Location-aware result inference and task assignment. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 61–72. IEEE, 2016.

[2] Christopher Jonathan and Mohamed F Mokbel. Stella: geotagging images via crowdsourcing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 169–178. ACM, 2018.

[3] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.

[4] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1031–1046. ACM, 2015.

[5] Amazon mechanical turk. `https://www.mturk.com/`.

[6] Figure eigth. `https://www.figure-eight.com/`.

[7] Chinacrowds. `http://www.chinacrowds.com/`.

[8] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1015–1030. ACM, 2015.

[9] AKM Khan, Oscar Correa, Egemen Tanin, Lars Kulik, and Kotagiri Ramamohanarao. Ride-sharing is about agreeing on a destination. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 6. ACM, 2017.

[10] Mohammad Asghari and Cyrus Shahabi. An on-line truthful and individually rational pricing mechanism for ride-sharing. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 7. ACM, 2017.

[11] Zhao Chen, Rui Fu, Ziyuan Zhao, Zheng Liu, Leihao Xia, Lei Chen, Peng Cheng, Caleb Chen Cao, Yongxin Tong, and Chen Jason Zhang. gmission: A general spatial crowdsourcing platform. *Proceedings of the VLDB Endowment*, 7(13):1629–1632, 2014.

[12] Peng Cheng, Xiang Lian, Zhao Chen, Rui Fu, Lei Chen, Jinsong Han, and Jizhong Zhao. Reliable diversity-based spatial crowdsourcing by moving workers. *Proceedings of the VLDB Endowment*, 8(10):1022–1033, 2015.

[13] Quora. `https://www.quora.com/`.

[14] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[15] Yahoo answers. `https://answers.yahoo.com/dir/index`.

[16] Wikipedia portals. `https://en.wikipedia.org/wiki/Portal:Contents/Portals`.

[17] Shenghua Bao, Guirong Xue, Xiaoyuan Wu, Yong Yu, Ben Fei, and Zhong Su. Optimizing web search using social annotations. In *Proceedings of the 16th international conference on World Wide Web*, pages 501–510. ACM, 2007.

[18] Yonghong Yu and Xingguo Chen. A survey of point-of-interest recommendation in location-based social networks. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[19] Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. Lars: A location-aware recommender system. In *2012 IEEE 28th international conference on data engineering*, pages 450–461. IEEE, 2012.

[20] Omar Montasser and Daniel Kifer. Predicting demographics of high-resolution geographies with geotagged tweets. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[21] Netflix. `https://www.netflix.com/`.

[22] Tinder. `https://tinder.com/`.

[23] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[24] Vinod Hegde, Josiane Xavier Parreira, and Manfred Hauswirth. Semantic tagging of places based on user interest profiles from online social networks. In *European Conference on Information Retrieval*, pages 218–229. Springer, 2013.

[25] Jaeho Shin, Christopher Ré, and Michael Cafarella. Mindtagger: a demonstration of data labeling in knowledge base construction. *Proceedings of the VLDB Endowment*, 8(12):1920–1923, 2015.

[26] Jing Zhang, Victor S Sheng, Tao Li, and Xindong Wu. Improving crowdsourced label quality using noise correction. *IEEE transactions on neural networks and learning systems*, 29(5):1675–1688, 2018.

[27] Yudian Zheng, Guoliang Li, and Reynold Cheng. Docs: a domain-aware crowdsourcing system using knowledge bases. *Proceedings of the VLDB Endowment*, 10(4):361–372, 2016.

[28] Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J Franklin. Crowdsourced data management: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2296–2319, 2016.

[29] Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.

[30] Ludmila I Kuncheva, Christopher J Whitaker, Catherine A Shipp, and Robert PW Duin. Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis & Applications*, 6(1):22–31, 2003.

[31] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: a crowdsourcing data analytics system. *Proceedings of the VLDB Endowment*, 5(10):1040–1051, 2012.

[32] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.

[33] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[34] Upwork. `https://www.upwork.com/`.

[35] Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1133–1148. ACM, 2017.

[36] Susan B Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, pages 225–236. ACM, 2013.

[37] Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. Counting with the crowd. *Proceedings of the VLDB Endowment*, 6(2):109–120, 2012.

[38] Christoph Lofi, Kinda El Maarry, and Wolf-Tilo Balke. Skyline queries in crowd-enabled databases. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 465–476. ACM, 2013.

[39] Louai Alarabi, Bin Cao, Liwei Zhao, Mohamed F Mokbel, and Anas Basalamah. A demonstration of sharek: an efficient matching framework for ride sharing systems. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 95. ACM, 2016.

[40] Blerim Cici, Athina Markopoulou, and Nikolaos Laoutaris. Designing an online ride-sharing system. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 60. ACM, 2015.

[41] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[42] Leyla Kazemi and Cyrus Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 20th international conference on advances in geographic information systems*, pages 189–198. ACM, 2012.

[43] Lei Chen and Cyrus Shahabi. Spatial crowdsourcing: Challenges and opportunities. *IEEE Data Eng. Bull.*, 39(4):14–25, 2016.

[44] Yongjian Zhao and Qi Han. Spatial crowdsourcing: current state and future directions. *IEEE Communications Magazine*, 54(7):102–107, 2016.

[45] Uber. `https://www.uber.com/`.

[46] Lyft. `https://www.lyft.com/`.

[47] Didi. `https://www.didiglobal.com/`.

[48] Nyc taxi trips. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`.

[49] Yongxin Tong, Lei Chen, and Cyrus Shahabi. Spatial crowdsourcing: challenges, techniques, and applications. *Proceedings of the VLDB Endowment*, 10(12):1988–1991, 2017.

[50] Wei Li, Haiquan Chen, Wei-Shinn Ku, and Xiao Qin. Scalable spatiotemporal crowdsourcing for smart cities based on particle filtering. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 63. ACM, 2017.

[51] Yongxin Tong, Jieying She, Bolin Ding, Libin Wang, and Lei Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *2016 IEEE 32Nd international conference on data engineering (ICDE)*, pages 49–60. IEEE, 2016.

[52] Christopher Jonathan and Mohamed F Mokbel. Towards a unified spatial crowdsourcing platform. In *International Symposium on Spatial and Temporal Databases*, pages 379–383. Springer, 2017.

[53] Peng Cheng, Xun Jian, and Lei Chen. An experimental evaluation of task assignment in spatial crowdsourcing. *Proceedings of the VLDB Endowment*, 11(11):1428–1440, 2018.

[54] Hien To, Cyrus Shahabi, and Li Xiong. Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server. In *2018 IEEE 34th International Conference on Data Engineering*, pages 833–844. IEEE, 2018.

[55] Mohammad Asghari, Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, and Yaguang Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 3. ACM, 2016.

[56] Hien To, Cyrus Shahabi, and Leyla Kazemi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems*, 1(1):2, 2015.

[57] Peng Cheng, Xiang Lian, Lei Chen, Jinsong Han, and Jizhong Zhao. Task assignment on multi-skill oriented spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2201–2215, 2016.

[58] Dingxiong Deng, Cyrus Shahabi, and Ugur Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *Proceedings of the*

21st acm sigspatial international conference on advances in geographic information systems, pages 324–333. ACM, 2013.

[59] Yu Li, Man Lung Yiu, and Wenjian Xu. Oriented online route recommendation for spatial crowdsourcing task workers. In *International Symposium on Spatial and Temporal Databases*, pages 137–156. Springer, 2015.

[60] Google translate. `https://translate.google.com/`.

[61] Number of google translate users. `https://www.cnet.com/news/google-translate-boasts-64-la`

[62] itranslate voice. `http://itranslatevoice.com/`.

[63] Sayhi. `https://www.sayhitranslate.com/`.

[64] Canadian hansard. `http://parl.canadiana.ca/`.

[65] Philip Resnik and Noah A Smith. The web as a parallel corpus. *Computational Linguistics*, 29(3):349–380, 2003.

[66] Dragos Stefan Munteanu and Daniel Marcu. Improving machine translation performance by exploiting non-parallel corpora. *Computational Linguistics*, 31(4):477–504, 2005.

[67] Jason R Smith, Chris Quirk, and Kristina Toutanova. Extracting parallel sentences from comparable corpora using document level alignment. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 403–411. Association for Computational Linguistics, 2010.

[68] Aria Haghighi, Percy Liang, Taylor Berg-Kirkpatrick, and Dan Klein. Learning bilingual lexicons from monolingual corpora. In *Proceedings of ACL-08: Hlt*, pages 771–779, 2008.

[69] Charles Schafer and David Yarowsky. Inducing translation lexicons via diverse similarity measures and bridge languages. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

[70] Katharina Probst, Lori Levin, Erik Peterson, Alon Lavie, and Jaime Carbonell. Mt for minority languages using elicitation-based learning of syntactic transfer rules. *Machine Translation*, 17(4):245–270, 2002.

[71] Sonja Nießen and Hermann Ney. Statistical machine translation with scarce resources using morpho-syntactic information. *Computational linguistics*, 30(2):181–204, 2004.

[72] Ulrich Germann. Building a statistical machine translation system from scratch: how much bang for the buck can we expect? In *Proceedings of the workshop on Data-driven methods in machine translation-Volume 14*, pages 1–8. Association for Computational Linguistics, 2001.

[73] Omar F Zaidan and Chris Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1220–1229. Association for Computational Linguistics, 2011.

[74] Vamshi Ambati, Stephan Vogel, and Jaime G Carbonell. Active learning and crowd-sourcing for machine translation. In *LREC*, volume 1, page 2, 2010.

[75] Vamshi Ambati, Stephan Vogel, and Jaime Carbonell. Collaborative workflow for crowdsourcing translation. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1191–1194. ACM, 2012.

[76] Chris Callison-Burch. Fast, cheap, and creative: evaluating translation quality using amazon's mechanical turk. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 286–295. Association for Computational Linguistics, 2009.

[77] Chang Hu, Benjamin B Bederson, and Philip Resnik. Translation by iterative collaboration between monolingual users. In *Proceedings of Graphics Interface 2010*, pages 39–46. Canadian Information Processing Society, 2010.

[78] Michael Bloodgood and Chris Callison-Burch. Using mechanical turk to build machine translation evaluation sets. In *Proceedings of the NAACL HLT 2010*

workshop on creating speech and language data with Amazon's Mechanical Turk, pages 208–211. Association for Computational Linguistics, 2010.

[79] Vamshi Ambati and Stephan Vogel. Can crowds build parallel corpora for machine translation systems? In *Proceedings of the NAACL HLT 2010 workshop on creating speech and language data with Amazon's mechanical turk*, pages 62–65. Association for Computational Linguistics, 2010.

[80] Ann Irvine and Alexandre Klementiev. Using mechanical turk to annotate lexicons for less commonly used languages. In *Proceedings of the NAACL HLT 2010 workshop on creating speech and language data with Amazon's Mechanical Turk*, pages 108–113. Association for Computational Linguistics, 2010.

[81] Philip Resnik, Olivia Buzek, Chang Hu, Yakov Kronrod, Alex Quinn, and Benjamin B Bederson. Improving translation via targeted paraphrasing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 127–137. Association for Computational Linguistics, 2010.

[82] Location isn't just a feature anymore, it's a platform. https://www.wired.com/2010/03/location-isnt-just-a-feature-anymore-its-a-platform/, 2010.

[83] Barak Pat and Yaron Kanza. Where's waldo?: Geosocial search over myriad geotagged posts. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 37. ACM, 2017.

[84] Hanan Samet, Jagan Sankaranarayanan, Michael D Lieberman, Marco D Adelfio, Brendan C Fruin, Jack M Lotkowski, Daniele Panozzo, Jon Sperling, and Benjamin E Teitler. Reading news with maps by exploiting spatial synonyms. *Communications of the ACM*, 57(10):64–77, 2014.

[85] John Krumm and Eric Horvitz. Eyewitness: Identifying local events via space-time signals in twitter feeds. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 20. ACM, 2015.

[86] Majid Alivand and Hartwig Hochmair. Extracting scenic routes from vgi data sources. In *Proceedings of the Second ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information*, pages 23–30. ACM, 2013.

[87] Donglian Sun, Sanmei Li, Wei Zheng, Arie Croitoru, Anthony Stefanidis, and Mitchell Goldberg. Mapping floods due to hurricane sandy using npp viirs and atms data and geotagged flickr imagery. *International Journal of Digital Earth*, 9(5):427–441, 2016.

[88] Charles W Schmidt. Trending now: using social media to predict and track disease outbreaks, 2012.

[89] Amr Magdy, Louai Alarabi, Saif Al-Harthi, Mashaal Musleh, Thanaa M Ghanem, Sohaib Ghani, Saleh Basalamah, and Mohamed F Mokbel. Demonstration of taghreed: A system for querying, analyzing, and visualizing geotagged microblogs. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1416–1419. IEEE, 2015.

[90] Jeremy W Crampton, Mark Graham, Ate Poorthuis, Taylor Shelton, Monica Stephens, Matthew W Wilson, and Matthew Zook. Beyond the geotag: situating 'big data'and leveraging the potential of the geoweb. *Cartography and geographic information science*, 40(2):130–139, 2013.

[91] Yusuke Nakaji and Keiji Yanai. Visualization of real-world events with geotagged tweet photos. In *2012 IEEE international conference on multimedia and expo workshops*, pages 272–277. IEEE, 2012.

[92] Mark Graham, Scott A Hale, and Devin Gaffney. Where in the world are you? geolocation and language identification in twitter. *The Professional Geographer*, 66(4):568–578, 2014.

[93] Claudia Hauff. A study on the accuracy of flickr's geotag data. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 1037–1040. ACM, 2013.

[94] Isis Truck and Mohammed-Amine Abchir. Natural language processing and fuzzy tools for business processes in a geolocation context. *Advances in Artificial Intelligence*, 2017, 2017.

[95] Google images. `https://images.google.com/`.

[96] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 77–90. ACM, 2010.

[97] Jakob Rogstadius, Vassilis Kostakos, Aniket Kittur, Boris Smus, Jim Laredo, and Maja Vukovic. An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[98] Steven Tanimoto and Theo Pavlidis. A hierarchical data structure for picture processing. *Computer graphics and image processing*, 4(2):104–119, 1975.

[99] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[100] Mike Thelwall. The precision of the arithmetic mean, geometric mean and percentiles for citation data: An experimental simulation modelling approach. *Journal of informetrics*, 10(1):110–123, 2016.

[101] Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. Lars: A location-aware recommender system. In *2012 IEEE 28th international conference on data engineering*, pages 450–461. IEEE, 2012.

[102] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.

[103] Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. Cdb: optimizing queries with crowd-based selections and joins. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1463–1478. ACM, 2017.

[104] Christoph Lofi, Kinda El Maarry, and Wolf-Tilo Balke. Skyline queries in crowd-enabled databases. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 465–476. ACM, 2013.

[105] Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R Jennings. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. *Artificial Intelligence*, 214:89–111, 2014.

[106] Taskrabbit. `https://www.taskrabbit.com/`.

[107] Ju Fan, Meihui Zhang, Stanley Kok, Meiyu Lu, and Beng Chin Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. *IEEE Transactions on Knowledge and Data Engineering*, 27(8):2078–2092, 2015.

[108] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1203–1212. ACM, 2012.

[109] Adam Marcus, Eugene Wu, David R Karger, Samuel Madden, and Robert C Miller. Crowdsourced databases: Query processing with people. In *Cidr*. Cidr, 2011.

[110] Jingru Yang, Ju Fan, Zhewei Wei, Guoliang Li, Tongyu Liu, and Xiaoyong Du. Cost-effective data annotation using game-based crowdsourcing. *Proceedings of the VLDB Endowment*, 12(1):57–70, 2018.

[111] Chengliang Chai, Ju Fan, and Guoliang Li. Incentive-based entity collection using crowdsourcing. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 341–352. IEEE, 2018.

[112] Jongwuk Lee, Dongwon Lee, and Sang-Wook Kim. Crowdsky: Skyline computation with crowdsourcing. In *EDBT*, pages 125–136, 2016.

[113] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. Crowdsourced top-k algorithms: An experimental evaluation. *Proceedings of the VLDB Endowment*, 9(8):612–623, 2016.

[114] Yan Li, Ngai Meng Kou, Hao Wang, Zhiguo Gong, et al. A confidence-aware top-k query processing toolkit on crowdsourcing. *Proceedings of the VLDB Endowment*, 10(12):1909–1912, 2017.

[115] Kaiyu Li, Xiaohang Zhang, and Guoliang Li. A rating-ranking method for crowdsourced top-k computation. In *Proceedings of the 2018 International Conference on Management of Data*, pages 975–990. ACM, 2018.

[116] Zhou Zhao, Furu Wei, Ming Zhou, Weikeng Chen, and Wilfred Ng. Crowd-selection query processing in crowdsourcing databases: A task-driven approach. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015*, pages 397–408, 2015.

[117] Zhou Zhao, Da Yan, Wilfred Ng, and Shi Gao. A transfer learning based framework of crowd-selection on twitter. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1514–1517. ACM, 2013.

[118] Caleb Chen Cao, Jieying She, Yongxin Tong, and Lei Chen. Whom to ask?: jury selection for decision making tasks on micro-blog services. *Proceedings of the VLDB Endowment*, 5(11):1495–1506, 2012.

[119] Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo. On optimality of jury selection in crowdsourcing. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015*. OpenProceedings. org., 2015.

[120] Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang-Chiew Tan. Asking the right questions in crowd data sourcing. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1261–1264. IEEE, 2012.

[121] Chien-Ju Ho, Shahin Jabbari, and Jennifer Wortman Vaughan. Adaptive task assignment for crowdsourced classification. In *International Conference on Machine Learning*, pages 534–542, 2013.

[122] Chien-Ju Ho and Jennifer Wortman Vaughan. Online task assignment in crowdsourcing markets. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[123] David R Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, pages 1953–1961, 2011.

[124] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[125] Peng Cheng, Xiang Lian, Lei Chen, and Cyrus Shahabi. Prediction-based task assignment in spatial crowdsourcing. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 997–1008. IEEE, 2017.

[126] Djellel Eddine Difallah, Gianluca Demartini, and Philippe Cudré-Mauroux. Pick-a-crowd: tell me what you like, and i'll tell you what to do. In *Proceedings of the 22nd international conference on World Wide Web*, pages 367–374. ACM, 2013.

[127] Fenglong Ma, Yaliang Li, Qi Li, Minghui Qiu, Jing Gao, Shi Zhi, Lu Su, Bo Zhao, Heng Ji, and Jiawei Han. Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 745–754. ACM, 2015.

[128] Zheng Liu and Lei Chen. Worker recommendation for crowdsourced q&a services: A triple-factor aware approach. *Proceedings of the VLDB Endowment*, 11(3):380–392, 2017.