UNIVERSITY OF CALIFORNIA,
IRVINE

Discovering Real-World Context

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Arjun Satish

Dissertation Committee:
Professor Ramesh Jain, Chair
Professor Nalini Venkatasubramanian
Professor Deva Ramanan
Professor Bill Tomlinson
Dr. Amarnath Gupta

2013

DEDICATION

(Optional dedication page)
To ...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

CURRICULUM VITAE

Arjun Satish

EDUCATION

Doctor of Philosophy in Computer Science **2012**
University name *City, State*

Bachelor of Science in Computational Sciences **2007**
Another university name *City, State*

RESEARCH EXPERIENCE

Graduate Research Assistant **2007–2013**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2007–2013**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Ground-breaking article **2012**
Journal name

REFEREED CONFERENCE PUBLICATIONS

Awesome paper **Jun 2011**
Conference name

Another awesome paper **Aug 2012**
Conference name

SOFTWARE

CueNet `https://github.com/wicknicks/cuenet/`
Polyglot implementation of the CueNet ecosystem to tag faces in photos.

ABSTRACT OF THE DISSERTATION

Discovering Real-World Context

By

Arjun Satish

Doctor of Philosophy in Computer Science

University of California, Irvine, 2013

Professor Ramesh Jain, Chair

The search spaces of real world AI problems are extremely large. The strategy adopted to prune large search spaces is to accurately model the environment, and reason which parts of the search space can be pruned without hurting the performance of the decision making algorithm. Although relatively easier in virtual environments, modeling dynamic real world environments is a very challenging problem.

Consider the example of tagging faces in a person's photo album. The search space contains a few billion potential candidates. Any algorithm which attempts to directly tag one of billion people in a given photo will perform poorly. But, given a complete model of the world, the search space needs to be limited only to the entities who were present close to the camera's field of view at the time of photo capture. Now, the algorithm needs to decide over few tens of candidates as opposed to billions.

In this dissertation, we present *Context Networks*, a representation of real world environments used to prune search spaces for real world AI problems, and a novel *Progressive Discovery* algorithm to construct such networks from heterogenous data sources. We facilitate our following discussions through an example system to tag faces in personal photos. We discuss the architecture of a complete system to model data sources, construct context networks for a given AI problem (which in the case of face tagging would be a personal photo and the

exhaustive search space) and provides a pruned version of the search space most relevant to the problem. We also present experiments to quantitatively demonstrate the efficacy of our algorithm.

Chapter 1

Introduction

Annotating the heterogeneous content of photos is a challenging problem. The primary complexity of photo annotation problems lie in their large search spaces and the diversity of feature-based representations of semantically similar images. Contextual information about a photo provides knowledge to prune this large search space, and reducing the complexity of feature-based classification techniques. In this work, we specifically address the problem of annotating faces in photos. The main intuition behind this work is that photos capture the state of the real-world, and if we can reconstruct the various real-world objects and their relations at the time of photo-capture, this real-world context can prune the candidate search space. Ideally, the reduced search space will be orders of magnitude smaller, and will contain all the correct tags.

Recently, the following changes in the technology climate motivate us to view real world information as context. First, with mobile phones becoming the primary mode of photo taking, the nature of context has evolved from providing cues about tags, to describing the world around a photo taking moment when a person was clicking the camera. Second, with the ever increasing amount of personal, social and public information, it is becoming harder

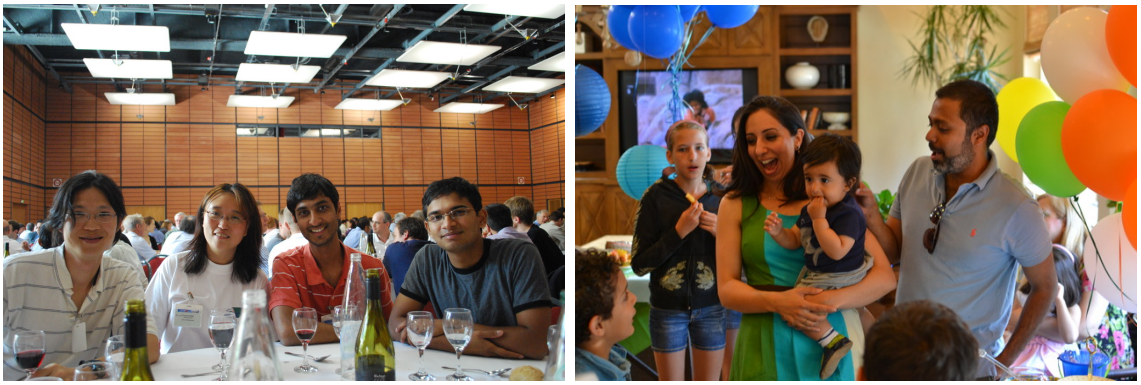


Figure 1.1: Photos can captured at a variety of different events.

to specify which subset of these would constitute the most interesting context for a given picture. Thus, it is not clear what source will provide context, and how do we combine them to form models of the real world which will allow photo annotation algorithms to reason what tags to assign various regions in a given photo. Finally, a large number of photos to be tagged today are *personal* in nature. Tagging them is very different from tagging photos in datasets like LFW [58] or Pascal-VOC [42]. Photos are created alongside other media (tweets, calendar entries, webpages), and a tagging system must take advantage of their inter-relationships.

The most relevant context for a photo is contained in the relationships between the various real-world events and entities at the *time of photo capture*. Given the various data sources and sensors, each of which contain partial information about real-world events and entities, it is non-trivial to meaningfully combine this information to produce the context for a photo. In this dissertation, we construct computational representations of such dynamic real-world events, entities and their relationships from the partial information in various heterogeneous data sources. We refer to such a representation as the **Context Network** of the photo. The network describes real-world events occurring with the photo-capture event, the entities participating in them, and their semantic inter-relationships. Given such a network for a photo, we can reason which parts of the search space can be pruned, and which of the remaining candidates are most likely present in the photo.

1.1 Importance of Context Discovery

Annotation of multimedia artifacts such as images, audio and/or video clips [45, 80, 84, 99, 104, 108, 106] is a very active area of research. A technique to discover relevant context will play a large role in making them more effective, and allow them to operate in large-scale application domains.

Lately, computer science is being utilized in addressing social [89], environmental [7] and economic real-world problems [30]. Social network analysis, data-driven diagnosis in medical sciences, philanthropic engineering, monitoring public interests through real time communication networks, and situation-based advertising are some of the emerging applications of computer science. The common requirement for all of them is to construct real-world context networks. A technology to construct such unified representations from various data sources available today will play a key role in the scope and architecture of such systems.

For the purposes of this dissertation, we propose context discovery techniques in the light of photo annotation problems, but the technology and ideas are not tightly coupled with any singly media, and can be ported to assist in solving any problem which requires models of real world information.

1.2 Context in Multimedia

Context has been used to address many multimedia problems [54, 69, 75, 78, 96]. For example, time and location information or social network information from Facebook to address the face recognition problem in personal photos. We refer to such a direct dependency between the search space and a data source as **static linking**. Although these systems are meritorious in their own right, they suffer from the following drawbacks: they are tightly

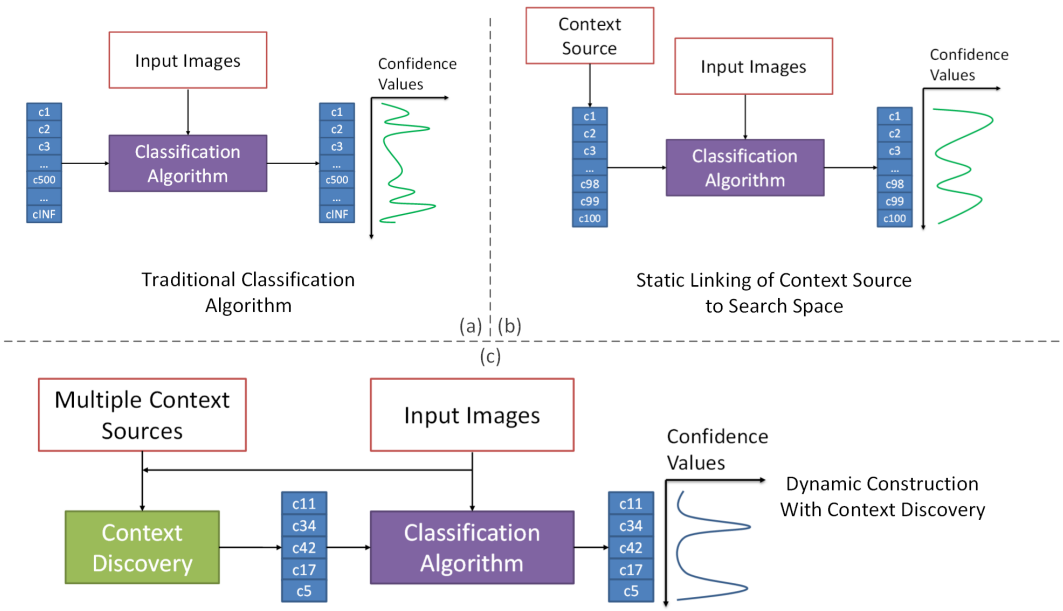


Figure 1.2: The different approaches in search space construction for a multimedia annotation problem.

coupled with a few data sources, the unavailability of any of which would reduce the efficiency of the system. They do not employ multiple sources synergistically, and therefore undervalue the **relations** between them. By realizing that these sources are interconnected in their own way, we are able to treat the entire source topology as a network, and traverse it to discover the relevant context.

Figure 1.2 shows three approaches to building photo annotation systems. Figure 1.2(a) is one of the earliest system setups where the search spaces were manually constructed, and the focus was mainly on constructing smarter features to correctly classify image regions [15, 97]. Figure 1.2(b) shows systems such as [96] which use a single source, to populate their search space based on some attributes of the input problem (in this case, the identity of the user to query Facebook to discover relevant candidates). This restricts the search space but it assumes that all relevant tags will be supplied from the context source, which may turn out to be an expensive restriction. In contrast, our approach **dynamically links** context sources to the photo. We assume the availabilty of a large number of sources and sensors, but extract context from only a subset of them for a photo taken by a known user.

Figure 1.2(c) shows how properties of input photo is used to reduce the search space.

1.3 Approach

This primary contribution of this dissertation is a ***progressive discovery*** algorithm to ingest information from various real world data sources to construct context networks containing relevant information for pruning the search space of annotation problem. Examples of data sources include social media web services to provide information about events and entities such as Facebook, Twitter; services which can be queried to find information about places such as Yelp; Sensors on personal mobile phones, for example GPS which inform applications of the location of a person is present at any given point in time.

In contrast to static linking, progressive discovery **dynamically links** context sources to the photo. It uses all available knowledge about a given problem (the input photo and related properties) to associate a subset of data from various sources as context. This allows us to decouple the technique of gathering context from the properties of the photo data. Because of this decoupling, there is no direct dependency on any specific set of sources, freeing the system developers to plug-and-play with them.

1.4 Examples

Here are two examples of dynamic linking. Figure 1.3 shows a photo of a person giving a presentation. Progressive discovery is done in three steps. First, the discovery algorithm proceeds to find the EXIF parameters and the camera device's owner with the photo. In our discussions we will use the term `photo-capture` event to signify the event in which a person clicks his/her camera and records a photograph. The event is assumed to contain spatio-

temporal attributes which specify when and where the photo was taken. In the next step, these spatio-temporal attributes are used to discover *if this entity (owner) is participating in any event at this time*? Only those data sources are searched which can provide results to such a query (for example, facebook's social network will not be searched, as it does provide temporal information about people). We obtain a response from a conference database saying that the owner was attending the ICMR conference at Dallas, Texas. Third, given this new conference event, the algorithm discovers what conference subevents (keynotes, talks or break sessions) were occurring at this time. Finally, it finds that Mor Naaman and John Smith were speaker and host respectively, for the keynote talk going on that time. Given the two candidates, the face tagging algorithm proceeds to identify Mor Naaman as the person in the photo, as shown in figure 1.4.

Figure 1.3: Who is in this photo?

Figure 1.4: Mor Naaman at ICMR.

Now, lets look at in photo in figure 1.5. Context discovery initiates the same way as in the above example, but after searching for events related to the owner in data sources, finds nothing. It proceeds to rank all known contacts according to location, and given that this photo was taken to the owner's workplace ranks colleagues higher than friends. The top 20 (an arbitrary constant) ranked candidates are passed to the face tagging algorithm which finds Ramesh Jain in the photo. But not the person to his right in the photo. Since the `photo-capture-event` has an additional participant, Ramesh, his personal information (calendar or social network) can be queried to find events in which he was participating. The calendar returns the entry "Kasturi". The algorithm uses this term to find all Ramesh's

contacts to find all people with first or last name "Kasturi", and finds his long time friend and colleague "Rangachar Kasturi". The face tagging algorithm is invoked with one candidate. In this case, it is simply checking if Prof. Kasturi is present in the photo or not.

Figure 1.5: Who is in this photo?

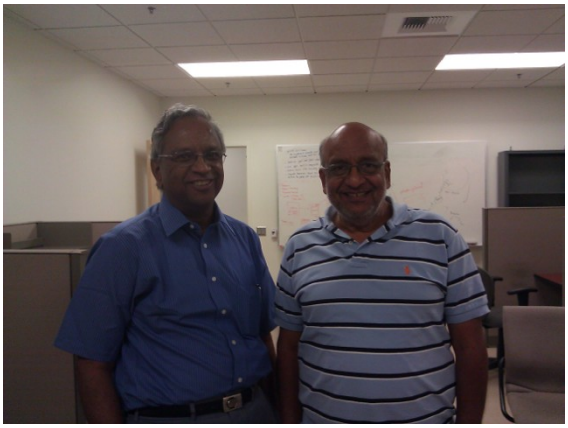


Figure 1.6: Kasturi and Jain.

In this first run, the algorithm links only to a conference database, whereas in the second case, it used spatial information, personal calendar and contact information. For the first image, we did not link social networking or contact information to the image, and similarly, conference databases were not linked to the second image. This variation in linking to sources is an example of dynamic linking. In the later chapters, we will present techniques to represent and link context in a systematic manner.

1.5 Overview

This dissertation is organized into the following chapters. Chapter 2 provides an overview of context, how context has been used to address problems in various scientific disciplines and how we use context in our specific personal photo tagging application. Chapter 3 describes the related work in computer science, specifically the main ideas in photo annotation and tag propagation techniques, and presenting the techniques prevalent in context modeling and processing communities: how context is used to solve problems in different domains.

Chapter 4 describes our context discovery framework, how it models various data sources, and how our progressive discovery algorithm constructs models for real world problems. We facilitate this discussion with an example real world application to tag faces of people in personal photos. Chapter 5 analyzes the algorithmic complexity of different parts of the system, and provides experiments to verify the competence and performance of the system. We also present experiments to confirm the efficacy of our approach in the light of the real world application. Chapter 6 presents a technique to rank candidates using past context networks to predict candidates in new photos. Finally, chapter 7 attempts to describe the future possibilities of using context discovery in various applications.

Chapter 2

What is Context?

The existing definitions of the word context are usually tied to the applications they are supporting. In the photo tagging problem, photos can be taken in a variety of situations, and appropriate context can be obtained from a multitude of data sources. Existing definitions of context do not provide enough clues as to what context is relevant when many data sources are present. Should we consider all available objects and relations across time? Or, does context consist of co-occurring events only? Or, the entities in a social network? Or, a combination of both? In this chapter, we look deeper into these definitions, and extract a relation-centric perspective of contextual information. This perspective will equip us to associate all relevant objects, irrespective of their type, as context, and at the same time, provide a reasoning framework to evaluate which source is beneficial and which is not.

Our justification for the use of context in a personal face tagging application begins with the assumption: *For a given user, the correctness of face tags for a photograph containing people she has never met is undefined.* This observation prepares us to understand what context is, and how contextual reasoning assists in tagging photos. The description of any problem domain requires a set of abstract data types, and a model of how these types are related to

each other. We define contextual types as those which are semantically different from these data types, but can be directly or indirectly related to them via an extended model which encapsulates the original one. Contextual reasoning assists in the following two ways. **First**, contextual data restricts the number of people who might appear in the photographs. We can also argue that all the personal data of a user (her profile on Facebook, LinkedIn, email exchanges, phone call logs) provides a reasonable estimate of all these people who might appear in her photos. **Second**, by reasoning on abstractions in the contextual domain, we can infer conclusions on the original problem. We exploit this property to develop our algorithm in the later sections. Though context based search space pruning can be applied to a variety of recognition problems, we focus on tagging people in personal photos for concreteness, where, the image and person tag form the abstractions in the problem domain, and events and event based relations constitute the contextual domain.

2.1 Previous Definitions

One of the earliest studies on context was reported by Bill Schilit et al. in [91]. The focus of this study was how to build software in dynamic environments. The dynamics of the environments were largely due to people requiring different computational services at the different times, the modality of request (through a mobile device or through a workstation), and the environment of the device (are there cameras and projectors nearby if the task requires video conferencing?). This software-centric view of context highlights the importance of two things. One, context is always described with respect to an object. In this case it is the software which runs on processors distributed in a real world environment. Second, context is used to determine how this object interacts with the entities near it? For example, Schilit uses the example that a workstation should automatically load his favorite text editor when he approaches it; and an rooster music sample must be played whenever fresh

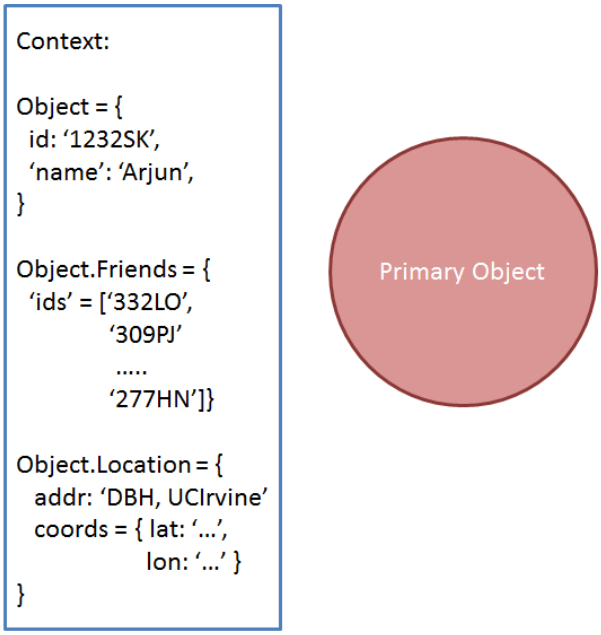


Figure 2.1: Information related to the situation of an Object.

coffee is prepared. Both very different and precise interactions even though they might share common background (environment or participating entities). We would not expect a text editor to be shown when coffee is prepared, and the rooster music to be played when an employee walks to a workstation.

In his seminal paper, Anind Dey [34] describes context *as any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*, as shown in figure 2.1. He proceeds to explain this definition with the example of an "indoor mobile tour", arguing that there are there are two additional pieces of information which can be used: *weather* and *presence of other people*. if the user is present with his friends, they might visit sites that are of interest to everybody. There the presence of other people is important context. Because the tour is indoor, weather does not affect the application. It is true that the weather has no direct affect on the application but what about the following scenarios:

- Could we use the weather information to serve different drinks in the cafeteria to boost the experience of the visitors? On a cold day, placing the hot chocolate kiosk next to the entrance and the ice cream kiosk closer on a warmer day might boost some sales.

- If the tour is similar to Alcatraz, where a ferry ride takes people to the island, and back from it, a storm brewing in the ocean could lead to disrputed ferry services. Should the application warn its users who are liesurely touring at this time? Or should they continue the tour at the same pace, miss the last ferry and spend the night at Alcatraz?

They proceed to define Context-Aware computing as follows: *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's tasks.* But, we need to ask ourselves why a system which uses this "additional information" should be considered a context-aware system? There are numerous systems which would simply consider these "additional information" as regular inputs. What is different between a system which takes in these inputs and processes them as regular data, and one which processes them as context?

Karen Henricksen et al. [53] make the following interesting observation: Context information exhibits a range of temporal characteristics. Some context information can be static, for example the attributes of people using a system (for example, the date of birth of a person). But a large amount of information is dynamic. For example, the current geo position of a person or her social network, as shown in figure 2.2. There is no straightforward way to obtain this dynamic information other than through sensors. But, such a approach tightly couples the application logic to the types of sensors used, and requires the system to convert the input data to usable representations. For example, the application requires explicit modules to convert GPS coordinates to readable addresses. The problem with such an approach is that there are many ad-hoc modules built to tackle the sensors, and therefore causing the context-awareness to be tied to a specific application.

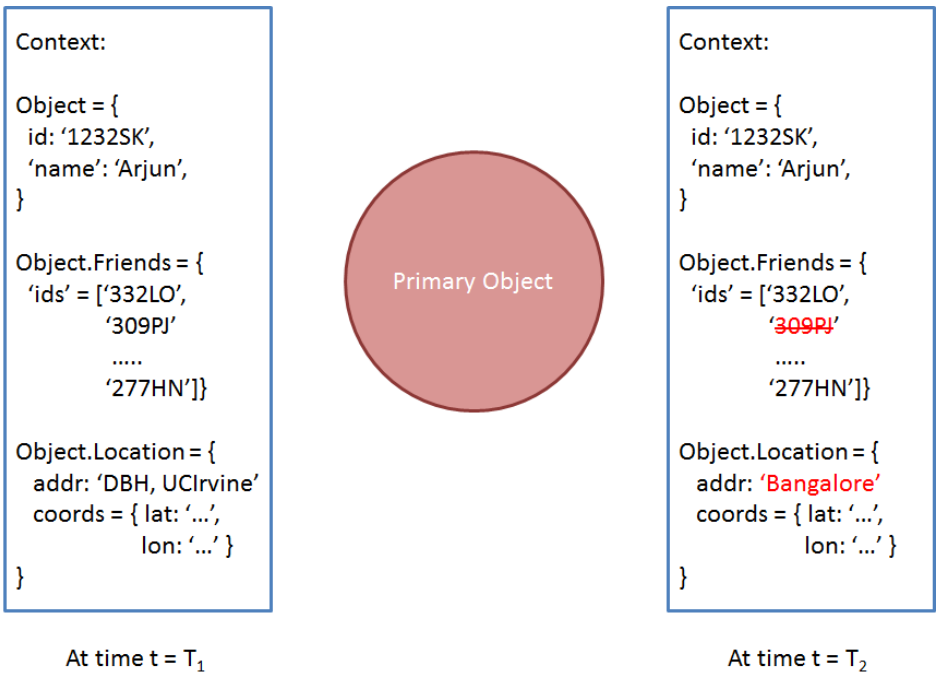


Figure 2.2: Henricksen's observation about temporality of Context.

More recently, Vaninha Vieira et al. [98] uses a rule centric view of context to design their context sensitive system, Cemantika. Vaninha defines a contextual element as any piece of data or information which can be used to characterize an entity in an application domain, whereas the context of an interaction between an agent and an application is the set of instantiated contextual elements that are necessary to support the task at hand. Context awareness, for them, is to explicitly switch the task the system is executing under different conditions. For this they explictly model the *context sources* which includes heterogenous and external sources like sensors, user dialog interfaces and databases. Figure 2.3 shows various data sources providing context. Some data sources are preferred over others depending on certain conditions pre-defined in the system. This allows the various processes to operate independently of the type of sources. It should be noted that the use of ontologies is describing knowledge and context sources is becoming increasingly popular (more similar systems are described in chapter 3).

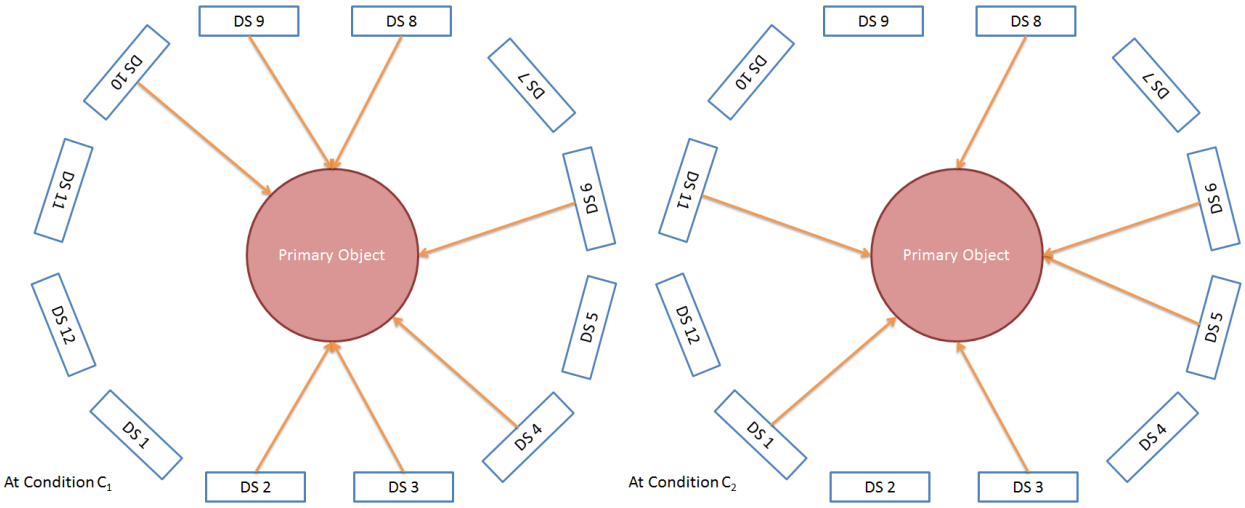


Figure 2.3: Modern context aware system obtain data from different sources.

2.2 Relation-Centric View of Context

The common ground behind these definitions is their object centric view of context. Context is largely a set of objects that "surrounds" a primary object (whose context is in question). This view is insufficient while addressing applications which are very broad in scope, the photo tagging application for example, where users can take photos in very diverse environment, and a large number of sensors and sources of context exist. Specifically, it is non-trivial to identify which subset of available data qualifies to be relevant context for the given photo, and which is not. The two examples from chapter 1, are shown in figures 2.5 and 2.6. In order to tag the photo on the left, we exclusively used conference calendar information. Whereas, to tag the photo on the right, we used personal information. Our motivation in this chapter is to extend the above definitions of context to allow context aware systems to better extract relevant context from these various sources.

We define context of a given object at a particular time as **"the real world information which can be related to the object directly or indirectly through a known set of relationships"**. Figure 2.4 is an extension of 2.3 by making explicit the relations between various objects in the application ecosystem. Here, objects contained within two data sources

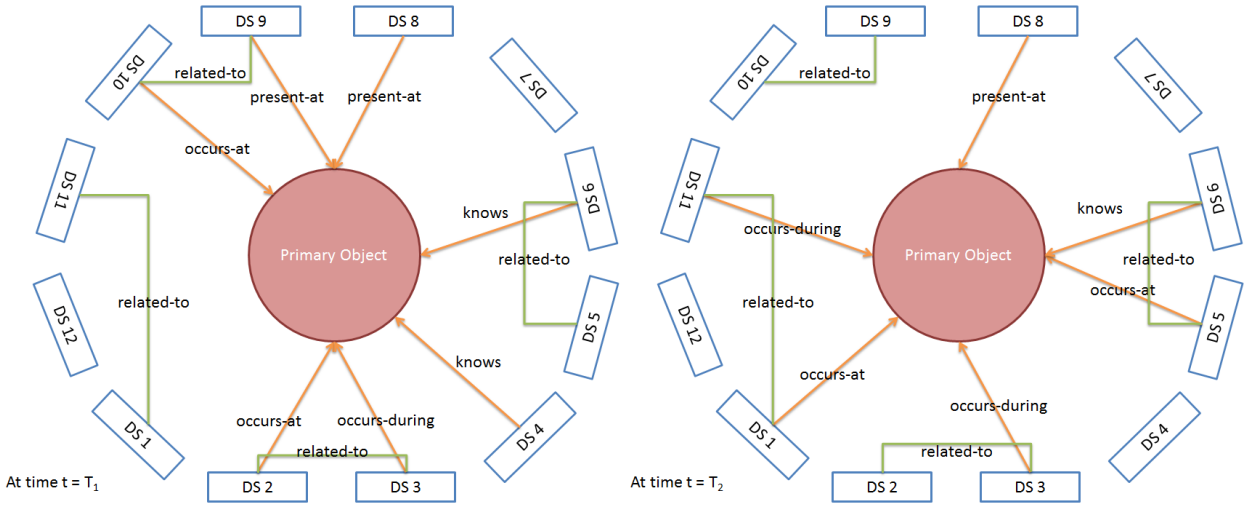


Figure 2.4: Utilizing relations to define context for the primary object.

can be related to each other. For example, an object in DS 9 can be related to another in DS 10 with the `related-to` edge. The primary object can be related to an object in DS 2 with an `occurs-at` relationship at time t_1. Whereas at time t_2, the relationship with DS 2 does not exist, but a new relationship `occurs-during` surfaces with DS 11.

Relationships can be of different types. They can be simple labels like `friend-of` signifying a social relationship. Or they can more actionable like `located-at`, which relates a person to a particular location, and therefore causes a particular audio stream to play through the handheld device. This relation is not just a label, it imposes constraints on the properties of objects which it relates. Here, the spatial attributes of the the person and the exhibit must match if they are being related through a `located-at` attribute. In this dissertation, we will see that such relations, which impose such property constraints are critical in algorithmically determining which information is relevant context. Let us also assume that we have available to us the four types of data sources: event sources, place databases (like yelp.com), weather information sources and social networking information.

Using this relation centric view, we now look at how the examples from chapter 1 can be formulated in a systematic process to discover context. A system to discover context must

Figure 2.5: Mor Naaman at ICMR.

Figure 2.6: Kasturi and Jain.

establish a set of relationships its context objects can be connected with. For the two photos, we choose **participant-of**, which indicates that an entity is a participant in an event, and **subevent-of** which indicates that an event is occurring within another super-event. Thus, any entity can be related to other events through a `participant-of` edge, and events can be related to others events as well as entities through `subevent-of` and the `participant-of` edge respectively.

Figures 2.7 through 2.10 show how the two relationships can be used to gather context for the photo in figure 2.5. Figure 2.7 shows the initial graph created using the `photo-capture-event` and the photographer, `entity:ent81`. Figure 2.8 shows the result of adding context by associating an event with `entity:ent81`. Given this new graph with three nodes, a context aware system can find more context by trying to find objects which can be related through the two edges. Since one of the nodes is a `conference` event, it proceeds to find events occurring within it, and adds the keynote event, which also happens to be the super-event of the `photo-capture-event`. The result is shown in figure 2.9. Finally, the keynote event is extended with relations to associate subevents or participants. In this case, the only new context available are the participants of the keynote event. These two entities are associated with the event as shown in figure 2.10.

In the above example, given a graph containing primary objects, we grew it by relating ob-

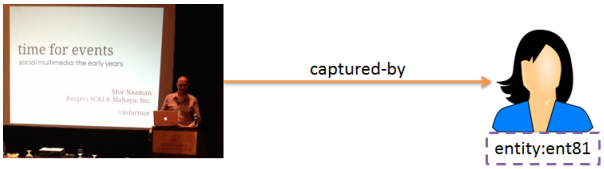


Figure 2.7: Primary objects.

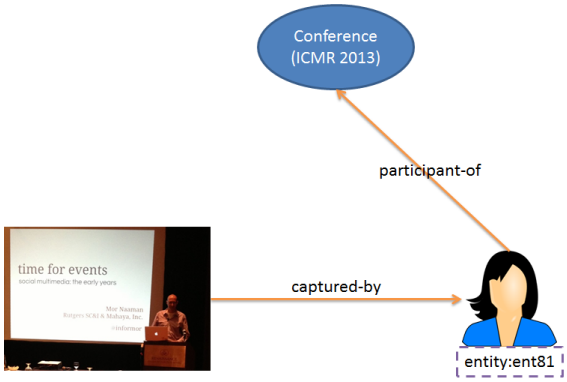


Figure 2.8: Associating `conference event` using the `participant-of` relation

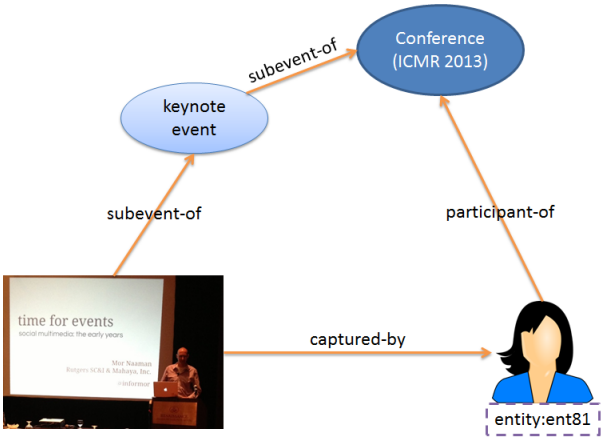


Figure 2.9: Associating the keynote event.

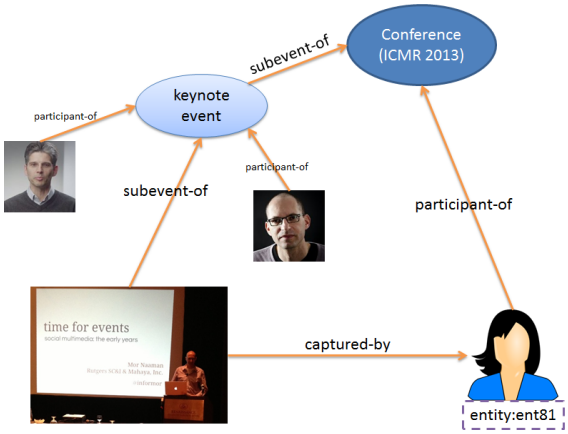


Figure 2.10: Associating the participants using the `participant-of` relation

jects from the real-world using a fixed set of relationships {`participant-of`, `subevent-of`}. No matter how many different sources or object types are included, we can use the relations and their semantics to reason which source is suitable, and extract context from it, if any, to construct and grow context networks.

Because of the type of relationships chosen, some information which was readily available (weather, place or social networking, for example) was not associated. But if we extend the relationship set to contain another relation `occurs-at`, then the place where the conference was held will be included in the context network. Similarly, the inclusion of a `knows` relation

can relate entities with each other. If the social networking source reports that `entity:ent81` was a friend of Mor Naaman, an additional edge would be introduced between these nodes in the context network. Thus, relations are key in determining which objects are context and which are not, and how they are related to the primary objects. Figure 2.11 shows one such context network for Mor Naaman's photo taken at ICMR 2013 with the additional `knows` edges.

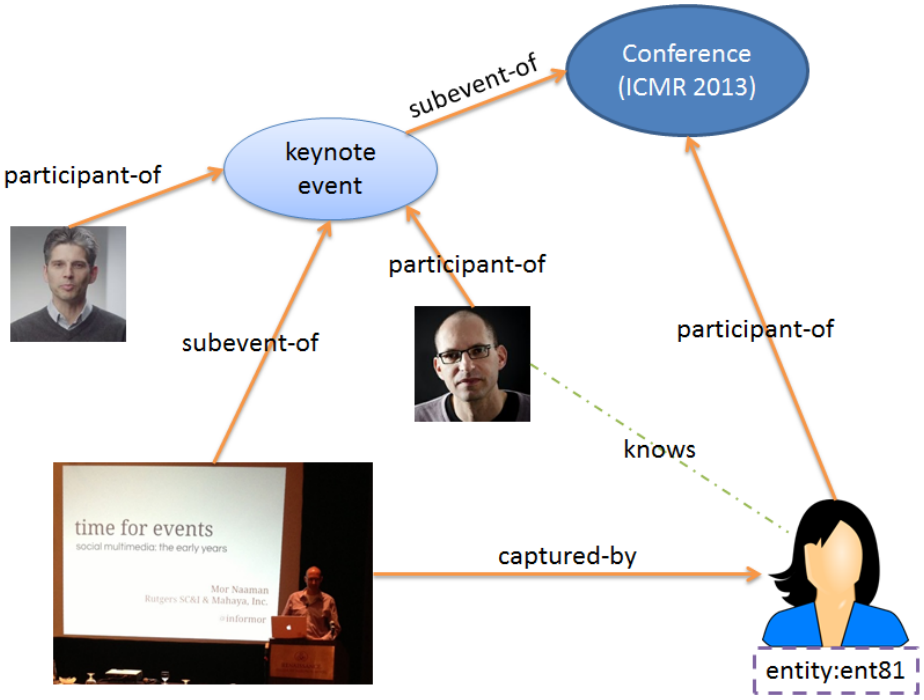


Figure 2.11: Context Network for Mor Naaman's photo with the additional `knows` edge.

It must be noted that asserting a relation-centric view of context is not discounting the importance of objects. Rather, it is a way of saying that relationships are atleast as important as objects are. Keeping this in mind, we will see what are the primitives required to model context for real-world problems in this section.

2.3 Modeling Context for Real-world Problems

There is no silver bullet to model context. Henricksen et al. [53] model context using context graphs, and [87] provides a technique to construct petri nets for given context graphs to implement context-aware behavior. Although, these techniques work for specific domain (for example, [87] presents a domain of class presentations, and [53] presents a case study in context-aware communications), creating context graphs to exist for each and every case which might occur in the real world is non-trivial. Also, with the rapidly changing relations in the real-world, it is not clear what are the good principles and practices to build scalable systems around context graphs.

If we are to represent context networks computationally, we first need to understand what are the basic building blocks of such a network. The following sections list these blocks, and present their properties which are essential in constructing such networks. These blocks are fairly generic, and can be utilized to model context for various application.

2.3.1 Object Types and Semantics

Context is always specified with respect to a real world object. This primary object must be uniquely identifiable in the computational system, and must be an instance of one of the known classes. This object must have some real world attributes. For example, if the object is an event, then the interval during which it occurs, and location of occurrence are two real world attributes. In the tour guide application, the visitor is one possible primary object whose context needs to determined. In the photo application, possible types of interest include the `photo-capture-event`, the `photographer`, various possible events which can occur in the world, people and places where photography can occur.

Different objects bring with them very specific semantics. While modeling objects as context,

these semantics must be preserved. For example, an event object exists only for a fixed time interval. People can only be present at one place at one point in time. The `photographer` is a person, and therefore inherits the property of being at only one place at a time. In the later chapter, we will see that these semantic properties play a very important role in the context discovery algorithm.

2.3.2 Relationships and Constraints

Context is any event or entity which can be *related* to the objects in a problem. Instead of finding objects which are of a specific type, and qualifying them as context, objects related to the problem variables through specific relationship types are to be considered context. These relationships can impose certain constraints on the objects which they relate. For example, the subevent relation requires that the super-event spatio-temporally contain the smaller event. Such constraints are very common in real-world relationships. An examples of a real-world relation can be seen in naturally occurring chemical reactions, where it is not sufficient to have just two reactants at the same time and place, but the reaction could demand very specific environmental factors (like temperature or pressure). The context modeling tool should permit the specification of these specific constraints.

More importantly, the relationships and constraints are not black boxes. Their semantics are transparent to all modules of a discovery engine, and these semantics are being used to reason which objects are part of a context network and which are not. Pushing our example of chemical reactions, given all the materials and conditions which can be present at a given region, it is almost impossible to treat the relationships between different materials under given conditions to simulate how the various reactions in the region.

In our application of photo tagging, social relations and presence of people in a particular event is to be used to rule them out of other events. Why? Because, a person can be present

only at one place at one point in time, and if the events are too far apart or contain a very distribution of people than what a person commonly co-participates with, the chances of that person being present in that photo is very low. Such a relation must be exclusively utilized in the context discovery algorithm.

2.3.3 Temporal Semantics

Modeling context requires the ability to express relationships which assert temporal constraints. For example, during a lunch break at a conference, there are no co-occurring session events; the workshops at a banquet is always the last event at a conference, which could be followed by one or more workshops. Temporal relations have been studied in literature [9, 101], and can be reused for this purpose. Temporal relationships defined in Allen's Interval Algebra are shown in figure 2.12. Figure shows these relationships. In order to relate events with entities, we use the `occurs-during` relation defined in [49]. For example, the event X is said to `occur-during` the interval I iff X's temporal extent is equal-to I.

Additional relations `occurs-sometime-during` and `occurs-during-n` are also defined in [49]. An event is said to `occurs-sometime-during` another interval I if its interval is completely within I, but not `equal-to` the extent of I. Look at the `during` relation in figure 2.12. The `occurs-during-n` is used to represent events which occur multiple times within a given interval. This relation is accompanied by an arithmetic function $\phi(n)$ which asserts constraints on the number of occurrences within the interval.

2.3.4 Spatial Semantics

Real-world events play an important role in context, and therefore, we need to pay special attention to the spatial relationships between entities and events. A large amount of

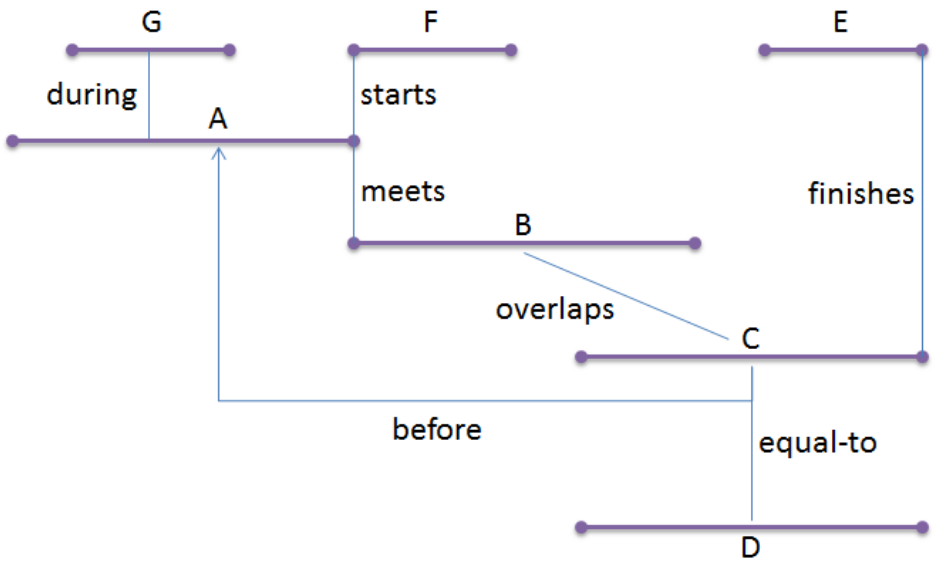


Figure 2.12: Intervals and their relations in Allen's Interval Algebra.

literature is available on spatial representation and reasoning framework originally known as RCC-8. Some of its common relationships are shown in figure 2.13. The important relations in the figure are `disconnected` (B and D), `externally-connected` (B and C), `tangential-proper-part` (A and B), `contained-in` (E and D) and `partial-overlaps` (C and D). In order to represent spatial relations between events and entities, we use the `occurs-at` relation defined in [49]. An event E is said to `occur-at` region S if the spatial extent of E completely lies within S.

Additional relations `occurs-somewhere-at` and `occurs-at-n` are also defined in [49]. An event is said to `occurs-somewhere-at`, if the actual region of occurrence is completely contained the given region, but not fully overlapping with it. The relation `occurs-at-n` is used when the event occurs at multiple places within the given region. This relation is accompanied with a function $\Psi(n)$, which asserts a constraint on the number of occurrences.

The relations between the different regions in figure 2.13 are as follows:

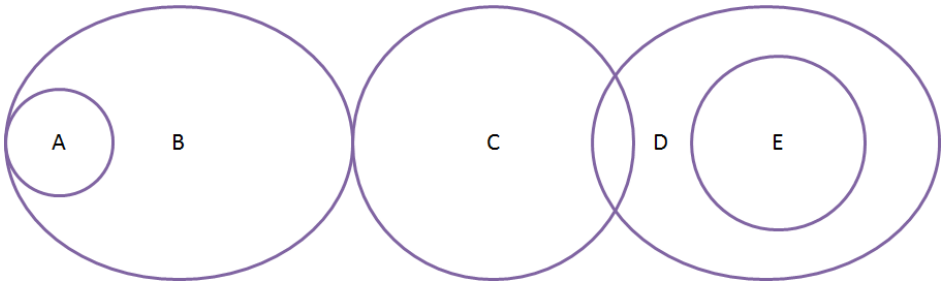


Figure 2.13: Spatial regions which can be represented in the RCC-8 framework.

Relation	**Symbol**	**Example**
Disconnected	*DC*	B *DC* D
Externally Connected	*EC*	B *EC* C
Tangential Proper Part	*TPP*	A *TPP* B
Completely Contained-In	*CI*	E *CI* D
Partially Overlaps	*PO*	C *PO* D

2.3.5 Real-World Knowledge

Modeling real world knowledge is critical in context based systems. This is in contrast with rule based systems, where a set of conditions are evaluated to trigger one or more actions. Examples of knowledge are: An academic conference has atleast one keynote talk; or Sodium reacts with atmospheric oxygen at room temperature; or water expands when it freezes. By explicitly modeling such networks of knowledge, we provide the opportunity to influence extraction of context from different sources. For example, if an object is associated with a keynote talk, data about the co-occurring conference is definitely going to be relevant context. Alternatively, if the system had associated a conference with the object, data about the keynote must be sought out. This 'guiding light' trait of knowledge makes it an important prerequisite in a context-aware system.

2.3.6 Source Agnosticism

Most context based techniques rely on sensors or data sources to gather context. Therefore, it must be imperative to separate the sources from context. Context must be modeled irrespective of the nature of sources or their query abilities [105]. This provides the following advantage: context is now entirely defined in terms of objects, their possible inter-relationships and real world knowledge decides how objects are related to each other, irrespective of what data actually can be obtained. Second, with the growing number of data sources, personal and public sensors, this allows system engineers to plug-and-play different sources with ease, without disturbing the model and discovery algorithms.

This design requirement also brings a problem. How does context-aware system know what data sources are at its disposal, and how does it interact with them? The short answer to this question is to utilize data integration technology [37, 51, 66]. Data integration techniques provide a uniform query interface to a multitude of autonomous data sources, irrespective of their native query capabilities or data storage formats. This might not be most ideal in the long run as these technologies were built to support RDBMS-like analytics operations, and we might benefit by adding more "context operations" into the integration layers, thereby allowing more fruitful grounds for query optimization. But for our current needs, it is advisable to re-use the ideas presented in these techniques to realize a fully working context-based system. The later chapters will describe in detail the data integration framework used to obtain data for the photo application.

There are many tools which partially support encoding models. For example, the ontology specification (OWL 2) supports specification of various objects, their inheritance structure (`is-a`), their inter-relationships (where the inter-relationships can have their own inheritance structure). But encoding time and spatial semantics is a relatively recent capability [55] in such frameworks, and not ready for prime-time for applications like real-world context

discovery. Thus, there is no silver bullet for modeling context, and in order to realize full systems, we need to separate the concerning capabilities between existing tools. In our work, relations expressed in OWL are simply labels with domain and range restrictions and sometimes cardinality constraints, their semantics being entirely delegated to the discovery engine.

2.4 Context for Personal Photos

In this section, we list the specific objects, relationships which will be used in constructing context networks, as shown in figure 2.14, for tagging personal photos. Once such a context network is constructed, the search space is derived from it. The types used in the contextual domain, but not limited to, are the following:

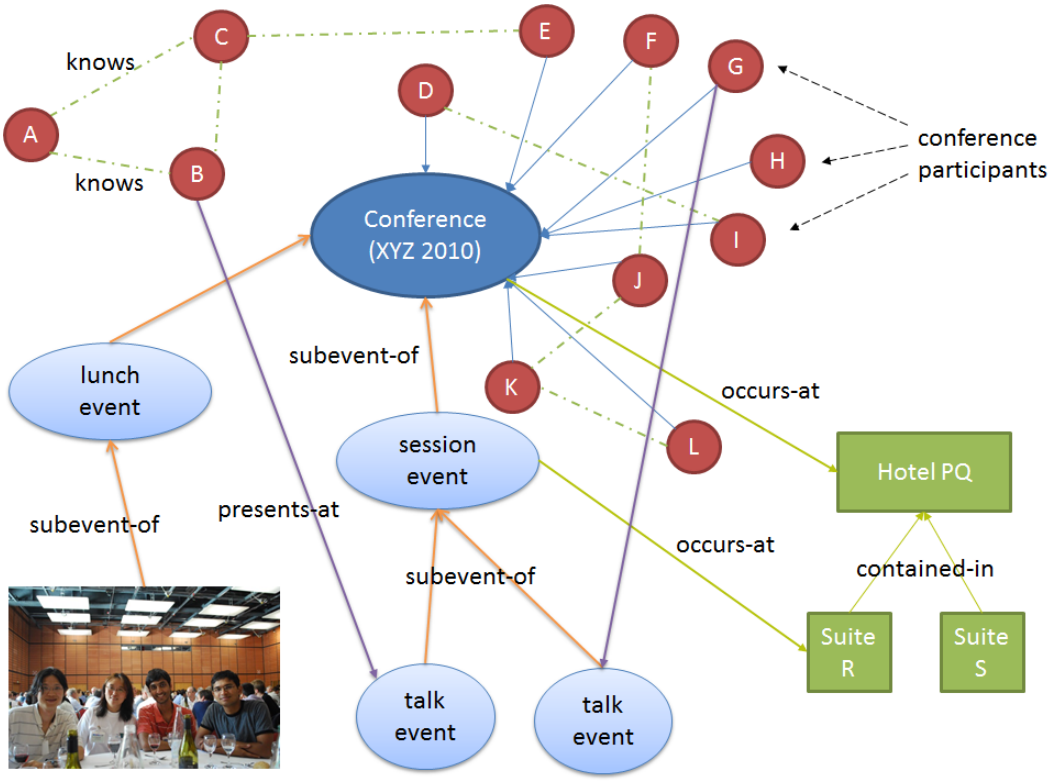


Figure 2.14: A Context Network representing real world events, entities and their respective relationships.

- **Event Objects**: events like conferences, parties, trips or weddings, and their structure (for example, what kind of sessions, talks and keynotes are occurring within a particular conference). We also model the photo as media facet of a `photo-capture-event`, which signifies the moment where a person took a photo with his/her camera. We model events as described in [49, 100].

- **Entity Objects**: entities model users, people in their social graph, people with whom he/she corresponds with using email and other messaging services. The different organizations which can participate in events mentioned above. For example, a company Acme Corp where the user and person X are colleagues working in the same team.

- **Geographical Objects**: Places play an important role. A place can referenced by a geo position, for example (33.643036, -117.841911), or an address for example "Donald Bren Hall, University of California, Irvine".

In order to relate the above objects, we use the following types of relationships, along with their constraints and semantics:

- **Subevent Relation**: If two events occur such that one spatio-temporally contains the other, we say that it is the subevent [49] of the one with larger spatio-temporal span. For example, a talk event is a subevent of the conference during which it happens.

- **Participation Relation**: If an entity is participating in an event, s/he is said to be a participant-of that event. Note that this relation constraints the spatio-temporal boundary where the entity could be present during the interval of the event.

- **Social Relation (knows)**: Social relations relate people who are acquainted with each other. This is used to model social networking information obtained from sources like Facebook or Email.

- **Spatio-Temporal Relations**: Events occur in specific time intervals, and at some location. We use the relations occurs-at and occurs-during to model these properties. More details are provided in the next chapter.

The above classes of contextual data can be obtained from a variety of data sources. Examples of data sources range from mobile phone call logs and email conversations to Facebook messages to a listing of public events at upcoming.com. We classify sources into the following types:

- **Metadata**: Photos contain very valuable metadata in the form of EXIF tags. Prior research in multimedia systems has used them extensively to gain remarkable results [21, 94]. We use time and location tags from EXIF in our work to construct `photo-capture-events`. Although we do not use them in our current work, the high importance of sensors embedded inside mobile phones deems their mention. Sensors like gyroscope, inclinometer, accelerometers and ambient light sensors can provide interesting cues about the environment [81, 93].

- **Personal Data**: include all sources which provide details about the particular user whose photo is to be tagged. Some examples of personal data sources include Google Calendar, Email and Facebook profile and social graph. Details include common personal attributes like name, date-of-birth, place of residence, place of work, the various they are attending (personal calendar), their personal preferences (type of food, music concerts, activity preferences).

- **Social Data**: include all sources which provide contextual information about a user's friends and colleagues. For example, LinkedIn, Facebook and DBLP are some of the commonly used websites with different types of social graphs. The information includes the personal information of the user's friends, their past or future activities where their participate together, their common friends (communities in social networks [12, 63]),

- **Public Data**: include all sources which provide information about public organizations (for example restaurants, points of interest or football stadiums) or about public events (for example conferences, fairs, concerts or sports games). Some sources include Yelp, Upcoming, DBLP and Factual.

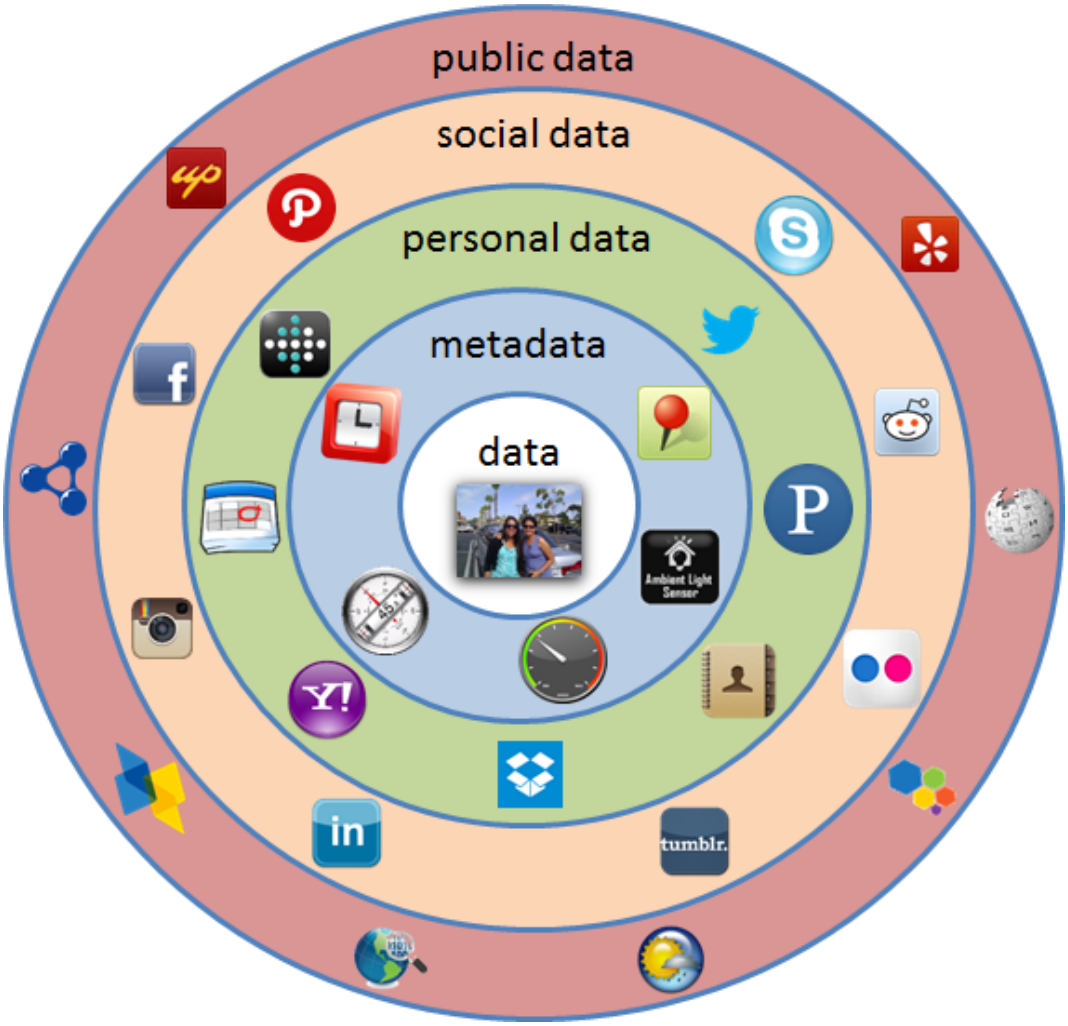


Figure 2.15: Sources containing personal, social and public context for a given photo.

Figure 2.15 shows some commonly available personal, social and public sources. Social and public data sources are enormous in size, containing information about billions of events and entities. Trying to use them directly will lead to scalability problems similar to those faced by face recognition and verification techniques. But, by using personal data, we can identify which parts of social and public sources are more relevant. For example, if a photo was taken

at Staples Center, Los Angeles, CA, an indoor stadium, we only need consider public events which such as concerts or sports games in the area. Thus, the role of personal information is twofold. **Firstly**, it provides contextual information regarding the photo. **Secondly**, it acts as a bridge to connect to social and public data sources to discover interesting people connected to the user who might be present in the event and therefore, the photo.

At this point we should revisit the **dynamic linking** property of a discovery algorithm. Given a stream of photos taken during a time interval, the source which contributed interesting context for a photo might not be equally useful for the one appearing next. This is because sources tend to focus on a specific set of event types or relationship types, and the two photos might be captured in different events or contains persons with whom the user maintains relations through different sources. For example, two photos taken at a conference might contain a user's friends in the first, but with advisers of these friends in the next. The friends might interact with the user through a social network, but their advisers might not. By using a source like DBLP, the relations between the adviser and friends can be discovered. We say that the temporal relevance of these context sources is ***low***. This requirement will play an important role in the design of our framework, as now, sources are not hardwired to photo, but instead need to be discovered gradually.

In short, the problem to be tackled is to construct context networks similar to the one shown in figure 2.14 using metadata, personal, social and public sources as shown in figure 2.15. This problem will be addressed using our context discovery algorithm, presented in chapter 4. The next chapter, will present a short survey of various techniques relevant to our problem, and highlight the important ideas which helped develop the algorithms presented in this dissertation.

Chapter 3

Related Work

In this chapter, we will look at the state-of-the-art understanding of different topics presented in a variety of areas, which context relies on. These include techniques to annotate photos using image features, techniques to represent knowledge in ontologies, techniques to query data from a variety of sources.

3.1 Photos and Annotation

Kindberg [61] conducted a study on multiple subjects to analyze photo capture behavior. Specifically, they were trying to understand the different motivations for people to take photos. They found two main motivations – *affective* and *functional*. A significant number of photos are captured to enrich a shared experience, or to share an experience with absent members like friends or family. Much lower, but a significant amount of photos were taken to share photos who were present at the event. Their study shows that photos are mostly used in social context, and lesser in personal context. Ames [10] and Frohlich [44] independently describe a survey conducted to study motivations for people to tag their photos. They

noticed two broad motivations: Organization of photos and Communication with photos. Almost orthogonal to the applications observed by Kindberg, Ames and Frolich are brave new world applications for photography described in [46, 39], where life logs were collected in the form of photos, emails, document scans and stored in SQL Server database, and photos were retrieved using SQL queries. The photo content was tagged by the user in this case. The annotated photos are used exclusively for personal *feedback* to improve quality of life. The medical advantages of collecting photos to log personal health are becoming very well known [16].

3.1.1 Spatio-Temporal Annotation

Ever since its release in 1995, EXIF metadata, contained in JPEG photos, has been exploited to organize pictures. Almost all photo management applications use timestamps to order photos in an album, a concept which has also been studied in academia [48, 50]. Apple's iPhoto is the most common example of using both time and space to organize personal photos. Naaman et al. have exploited GPS attributes to extract place and person information [74, 75]. Rattenbury [86] devised techniques to find tags which describe events or places by analyzing their spatiotemporal usage patterns. Sinha [94] and Boutell [22] have used EXIF metadata to predict concepts such as (Indoor, Outdoor, Portrait, Landscape) to further organize these photos. Boll [20] is a very interesting work which aggregates textual and experiential content from web 2.0 communities about places using a set of predefined rules. These works motivate the need for adding annotations to improve the overall experience of viewing a collection of photos.

3.1.2 Computer Vision

The Computer Vision community has contributed extensive work in the area of detecting scenes [103], humans [32] or geo localization [52]. Here we will specifically look at the part of the computer vision research which is relevant to face tagging.

Face Recognition The face recognition problem is one of the primary problems taken on by the computer vision community, the others being action, pose, gesture recognition and human detection. One of the earliest works on recognizing people in faces was attempted by Turk and Pentland in [97]. The essence of the technique put forward here was to extract useful features which can be represented mathematically and compared using some distance measures (such as euclidean, mahalanobis or cosine distance) to features in an image database. This contribution sparked a large interest in the community to find more meaningful and powerful features, which led to Fisherfaces [15] and more recently, local binary pattern [6] features. SIFT features have also been used to identify faces [19, 47, 71]. One of the big difficulties of such feature-based representation is the scalability of the technique. It is now well known that with the increasing number of candidates, these distance based techniques provide in lower performance [102]. A quick alternative is to increase the dimensions in the features to allow for more diversity, but it has been proven that as the number of dimensions increase, the maximum distance between the two points reduces, and if the number of dimensions is as less as 32, the distance is almost negligible [18]. Effectively, for dimensions more than 32, the distance between all points can be considered very small, and the concept of 'nearest-neighbour' loses its meaning. Thus, an pure feature based technique cannot be used to build large scale real world photo tagging frameworks.

Face Verification More recently, face verification techniques have attracted a lot of attention in the computer vision community. One of the earliest efforts towards robust face verification was undertaken by Huang et al. in [57]. They constructed and annotated a

dataset of profile photos of celebrities taken in unrestricted environments. Previous datasets such as FRGC [82], BioID [60] and the color FERET database [83] were criticized to have photos taken at very constrained environments thereby reducing the complexity of the face tagging problem. But the techniques which worked on such photos could not duplicate their performance in real-world photos. Huang's database, commonly known as the *Faces In the Wild* has allowed many researchers to implement various face verification technologies. A recent and important development on top of Huang's work is that of Neeraj Kumar et al. [64], where the confidence of presence or absence of facial attributes such as range, skin color, hair style, gender, eye color features are used to train various classifiers to ultimately test if two given faces are of the same person or not? The technique was able to achieve up to 85.29% accuracy on the LFW dataset. More recently, Berg [17] increased this accuracy to 93% by automatically finding distinguishing features.

The interesting perspective face verification brings forward is in its contract with co-existing components in a system. Whereas, face recognition works as a standalone component, face verification doesn't make this assumption, and allows external components to filter candidates. This systemic behavior of face verification will prove to be very useful in future systems.

Probabilistic Techniques Both face verification and recognition techniques seen so far assume that none of the input photos contain any annotation. But what if this assumption could be relaxed? A large number of photos on Facebook, Flickr or Google+ are annotated. If we further assume that these partial annotations are mostly true, the label propagation technique by Cao et al. [25] can be applied to annotate the rest of the dataset. This technique was tested on propagating concept based tags (such as beach, fun, dinner, yard) on personal photo datasets. Also, Barthelmess et al. extract semantic tags from noisy datasets containing discussions, speeches about a set of photos in question[13].

Miscellaneous Techniques Collaborative games also have been evaluated as a possible

way to tag photos[35]. Systems like Picasa, iPhoto and [48] organize photos based on time, GPS coordinates and sometimes faces in the photo. These attributes of the photo do not capture event semantics [90]. Events are a natural way of categorizing photo media. Events also allow large number of photos captured during a single event be organized hierarchically using subevents.

3.2 Context

The use of context in the sciences has been continuously increasing. It finds applications in various fields, starting from its use in holistic thinking to better understand biological and ecological phenomenon in [27], to associating the right external data to form coherent stories about economic phenomenon [67], to associating plausible connections between history and geography in [36]. The advances proposed in these works can be loosely characterized as "utilizing external information" or "out-of-the-box thinking". The reason they are included in this section is because of their common trait of relating multiple pieces of external data, to create coherent stories which allows the scientists to gain valuable insights into the problem they are solving.

3.2.1 Uses in Computing

In Computer Sciences, the main interest towards context has been largely from the Pervasive Computing community and the Human-Computer Interaction community. Their interpretation of the word context is mainly inspired from the definition set by Anind Dey [34], i.e. *Context is any information which describes the situation of an entity.* The role of context in mobile computing has been studied in [28]. It shows the growing importance of contextual thinking in reasoning about networking problems in the mobile computing era.

More recently, information retrieval communities are showing interest in context based representation of data and context-based techniques. One of the most important works in information retrieval is the PageRank algorithm [79] developed by Google co-founders Larry Page and Sergey Brin in 1996. In this work, Page and Brin define context as the links between different web pages, and the anchor text of these links, and argued that this context is more descriptive of a page than the contents of the page itself. PageRank was derived by combining this insight with Jon Kleinberg's HITS [62] algorithm.

3.2.2 Definitions

There are many definitions of the word context in academic works. Notable among them are those of Schilit [91], Dey [34], Viera [98], Zimmermann [109] and the work of Patrick Brezillon, a summary of which can be found in [73]. One of the problems with the term Context is the overloaded use of it [53]. To this effect, Brezillon compiled a list of 150 definitions, and did to this list what lots of scientists do with large collections of text – text analysis using natural language processing techniques, and derived the essential components which should make up a holistic definition of context. Their conclusion as described in [14] was that *context acts like a set of constraints that influence the behavior of a system (a user or a computer) embedded in a given task*. In spite of this promising "big-data" analysis, many questions are unanswered. For example, is context internal or external? Is it a phenomenon or an organized network? To the best of my knowledge, there is no consensus on what is a good definition of this word, and what are the general principles that one can be find a context-aware system.

3.2.3 Role in Photo Annotation

In [40] Dumitrescu and Santini argue that "images are a node in a complex network of signification that goes beyond their content and includes other form of textuality that go around them, as well as the cultural practices of the community that creates or receives images". Such philosophies became more commonplace after the web went contextual with PageRank. Image retrieval, specifically, obtained a remarkable opportunity to index and rank images on the web by using the textual content which accompanies it [29, 43]. Later, with Flickr the amount of user contributed tags has resulted in gigantic datasets which allow researchers to train sophisticated machine learning models to produce one or more relevant tags [23, 68, 70]. Some of early advocates of using external context to annotate photos are [33, 59]. Context information and image features are used in conjunction by [78, 25, 22, 26] identify tags. The semantic web community is using linked data technologies to annotate and query photographs [72, 76].

3.2.4 Modeling Context

Context is represented using three components: knowledge, external context and proceduralized context by [24]. These ideas are represented using a contextual graph (CxG) representation of knowledge and reasoning. Henricksen et al. also use a graphical notation to represent their context. The former approach uses the graph to model the flow of reasoning between different objects within an environment, whereas the latter approach uses graph to only represent the various objects and their inter-relationships. Their modeling framework allows representation of static and dynamic associations, which is very critical in modeling real world relationships. For our work, we rely on primitives like the one mentioned in this framework as well as event relations mentioned in [49]. [87] presents a technique to transform contextual graphs similar to concrete situation handling implementation using Petri Nets.

[56] presents a detail survey of various other context-aware systems.

Probabilistic and statistical frameworks are used to incorporate context in problem solving too. One example of using a probabilistic framework is [25], where high confidence tags from a few contextual images are propagated to other images which were taken during the same event. [96] uses potential functions in conditional random fields to model social relations and photo co-occurrence strengths. More recently, Zhang [107] used cues from photos like clothing, human attributes like face complexion or gender, people co-occurrence and scene annotations to cluster similar faces in photos together.

3.2.5 Industrial Momentum

The tech industry is immersed in the so-called "big-data" revolution. In order to make sense of such large collections of data, context is becoming increasingly popular. One of the biggest works in this area is the blog and upcoming book "Age of Context" by Robert Scoble and Shel Israel[1]. The reasons for this interest is cited on the increasing number of smartphones and sensors, personal and public. The increasing number of wearable devices like Google Glass, Oakley AirWave Goggles, Plantronics, Smith I/O Recon Heads-up Display, FitBit, Basis Watches, Nike FuelBand and Jawbone Up's personal health/activity monitoring gadget. The continuously increasing social data from companies like Facebook, Twitter, Flickr, Instagram and Pinterest, and information about locations through services like Foursquare, Google/Bing Maps, Waze and Factual. Lesser known startups like Tempo and Cueup (formerly Greplin) which are entirely focused on using personal contextual information to simplify otherwise very hard problems.

[1] http://scobleizer.com/2013/02/12/a-new-way-to-fund-a-books-development-sponsors-heres-ours/

3.3 Knowledge Representation

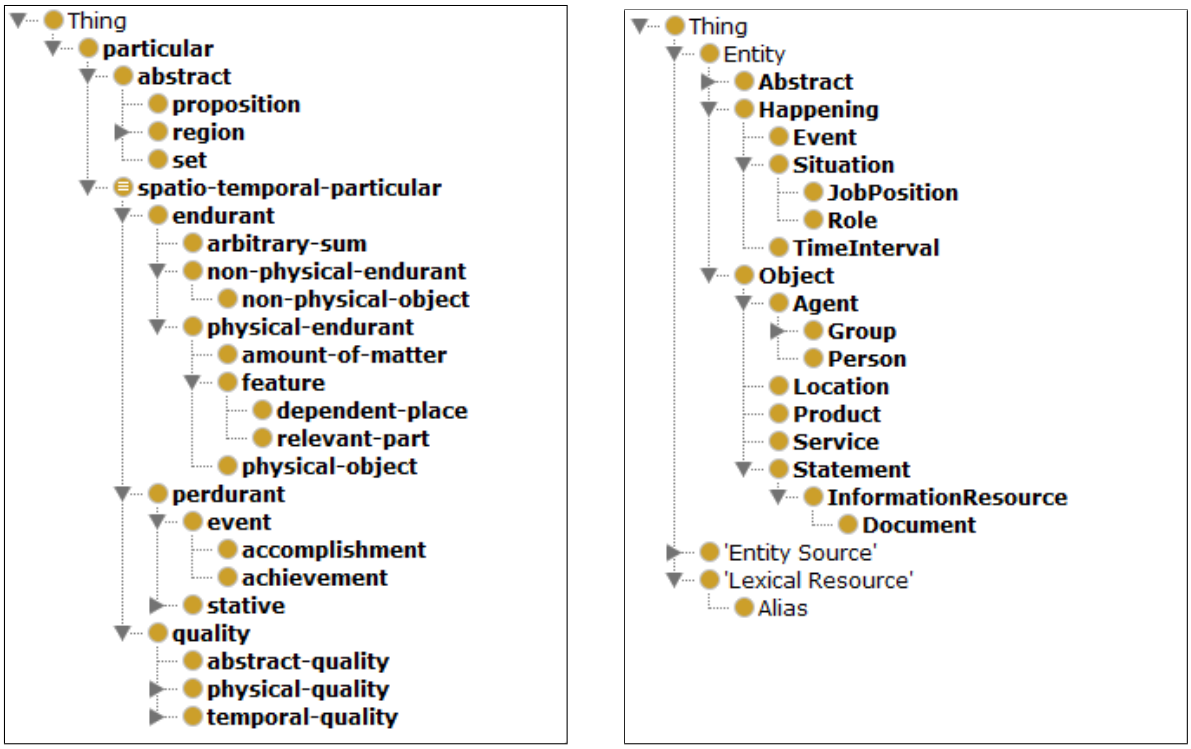


Figure 3.1: Dolce and Proton Class Hierarchies

In our work, we use ontologies to model key pieces of knowledge. This includes the type of objects and relationships they are expected to have. OWL is the current standard language in authoring ontologies. The current standard being OWL 2.0. The majority of work, related to this dissertation, is in utilizing ontologies have been in the data integration domain [95, 77, 11]. We utilize some of the features of the declarative mapping language [38] to express the structure of relations stored in various data sources. Specifically, we use the s-expression like syntax to declare the relations and the mapping axioms which relate items in the expression to objects in our ontology. SPARQL [85] is the current standard for querying RDF documents. The event and entity discover queries generated by the discovery algorithm in the next chapter are generated using SPARQL templates.

We use the terminologies followed in upper level ontologies. Figure 3.1 shows the class hierarchies for the Dolce Upper Ontology (on the left) and the Proton Top Level Ontology

(on the right). Similar to these taxonomies, we will use the term 'entities' to collectively refer to all real-world events and objects. Events, are perdurants, and have temporal or spatial parts. Events in our context discovery will include Academic Conferences, Concerts, Photo-Capture Events or Meetings. Objects, such as Persons, on the other hand, are uniquely identifiable wholes. They do not need temporal or spatial attributes for their description. It must be noted that we extend the DOLCE-Lite upper ontology to construct our knowledge base for CueNet.

Chapter 4

Context Discovery Framework

In this chapter, we shall look at the the CueNet framework and its components: a *data integration module* to model and query the various data sources and sensors, a *discovery algorithm* to construct queries agnostic to what the sources are themselves, a *knowledge representation module* to store relationships about the various real world objects, and finally mechanisms to interact with a *face verification algorithm*, which computes the confidence score of a person being present in a photo or not.

4.1 Pruning Search Spaces with CueNet

Automatic media annotation algorithms essentially assign one or more labels from a search space to a given input image. Figure 4.1 shows the various approaches of constructing such a search space for such an algorithm. The traditional approach is shown in 4.1(a). These spaces were limited to a set of labels chosen by an expert, with no way of pruning the search space in case it got very large. The focus was instead on extracting the best features from images, to obtain high overall classification accuracy[97].

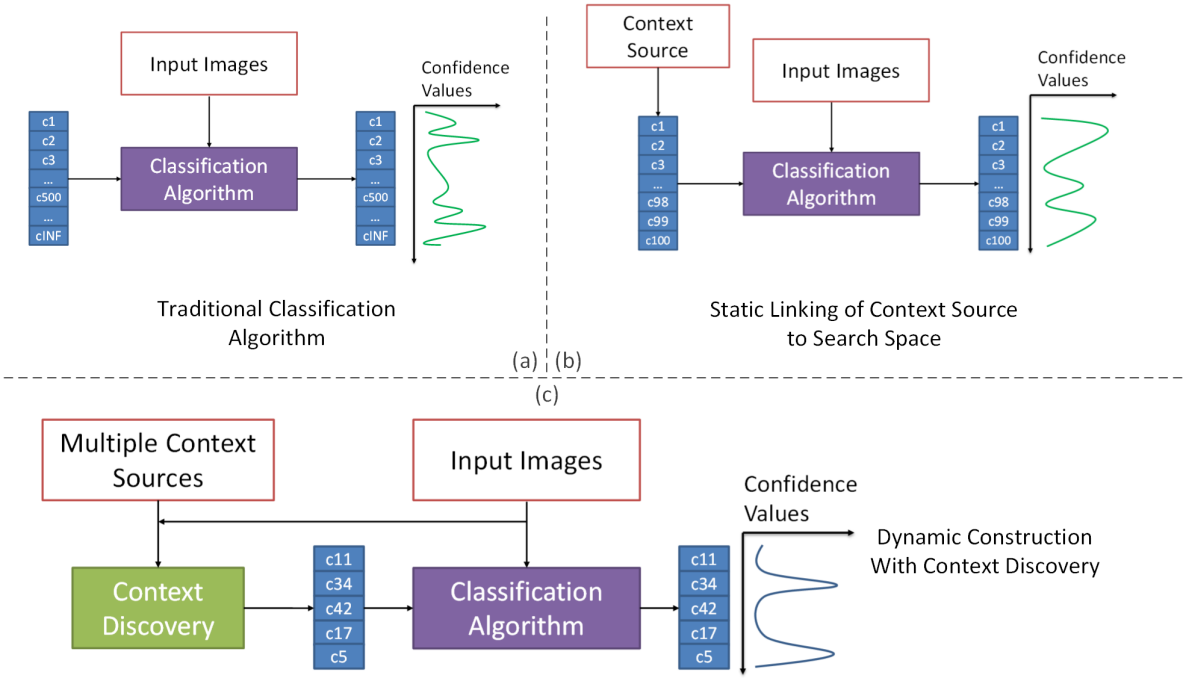


Figure 4.1: The different approaches in search space construction for a multimedia annotation problem. A traditional classifier setup is shown in (a) where the search space candidates are manually specified. Context is used to generate large static search spaces in (b). The desired framework is shown in (c), which aims to produce small search spaces with many correct annotations.

With the popularity of global social networks and proliferation of mobile phones, information about people, their social connections and day-to-day activities becoming available at a very large scale. The web provides an open platform for documenting many real world events like conferences, weather events and sports games. With such context sources, the search space construction is being delegated to one or a few sources [54, 69, 75, 78, 96] (figure 4.1(b)). These approaches rely on a single *type* of context. For example, time and location information or social network information from Facebook to solve the face recognition problem. We refer to such a direct dependency between the search space and a data source as **static linking**. Although these systems are meritorious in their own right, they suffer from the following drawbacks: they do not employ multiple sources, and therefore the **relations** between them. By realizing that these sources are interconnected in their own way, we are able to treat the entire source topology as a network. Our intuition in this work is to navigate this network

to progressively discover the search space for a given media annotation problem. Figure 4.1(c) shows how context discovery can provide substantially smaller search spaces for a set of images, which contain a large number of correct tags. A small search space with large number of true positives provides the ideal ground for a classification algorithm to exhibit superior performance.

The CueNet framework, provides access to multiple data sources containing event, social, and geographical information through a unified query interface to extract information from them. CueNet encapsulates our **Context Discovery Algorithm**, which utilizes the query interface to discover the most relevant search space for a media annotation problem. To ensure a hands-on discussion, we show the use of context discovery in a real world application: face tagging in personal photos. As a case study, we will attempt to tag photos taken at conference events by different users. These photos could contain friends, colleagues, speakers giving very interesting talks, or newly found acquaintances (who are not yet connected to the user through any social network). This makes the conference photos particularly interesting because no single source can provide all the necessary information. It emphasizes the need to utilize multiple sources in a meaningful way.

Here is an **example** to illustrate CueNet's discovery process. Let's suppose that Joe takes a photo with a camera that records time and GPS in the photo's EXIF header. Additionally, Joe has two friends. One with whom he interacts on Google+, and the other using Facebook. The framework checks if either of them have any interesting event information pertaining to this time and location. We find that the friend on Google+ left a calendar entry describing an event (a title, time interval and name of the place). The entry also marks Joe as a participant. In order to determine the category of the place, the framework uses Yelp.com with the name and GPS location to find whether it is a restaurant, sports stadium or an apartment complex. If the location of the event was a sports stadium, it navigates to upcoming.com to check what event was occurring here at this time. If a football game or a music concert was taking

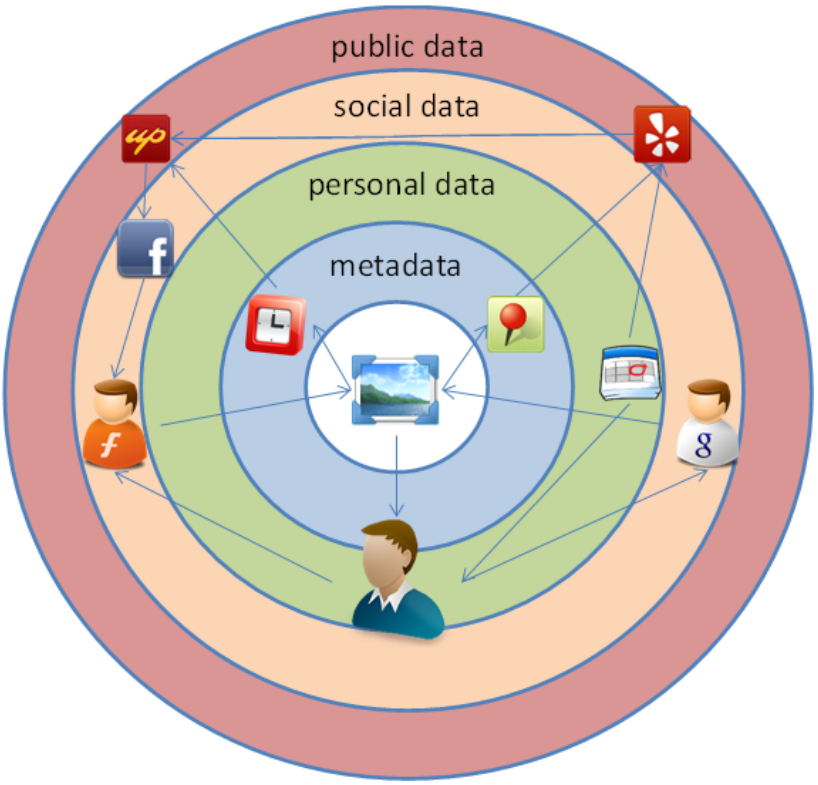


Figure 4.2: Navigation of a discovery algorithm between various data sources.

place at the stadium, we look at Facebook to see if the friend "Likes" the sports team or music band. By traversing the different data sources in this fashion, the number of people, who could potentially appear in Joe's photograph, was incrementally built up, rather than simply reverting to everyone on his social network or people who could be in the area where the photograph was taken. We refer to such navigation between different data sources to identify relevant contextual information as **progressive discovery**. The salient feature of CueNet is to be able to progressively discover events, and their associated properties, from the different data sources and relate them to the photo capture event. We argue that given this structure and relations between the various events, CueNet can make assertions about the presence of a person in the photograph. Once candidates have been identified by CueNet, they are passed to the face tagging algorithm (as in [65]), which can perform very well as their search space is limited to two candidates.

Figure 4.3 shows the different components of the CueNet framework. The Ontological **Event**

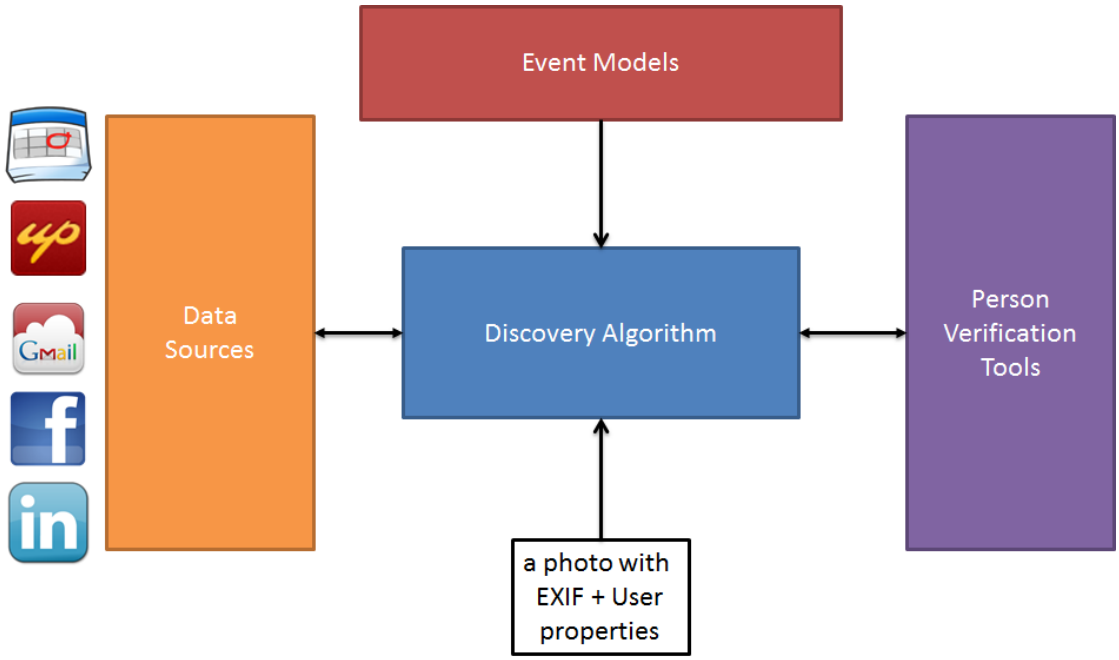


Figure 4.3: The Conceptual Architecture of CueNet.

Models specify various event and entity classes, and the different relations between them. These declared types are used to define the **Data Sources** which provides access to different types of contextual data. The **Person Verification Tools** consist of a database of people, their profile information and photos containing these people. When this module is presented with a candidate and the input photograph, it compares the features extracted from the candidate's photos and the input photo to find the confidence threshold. In this section, we describe each module, and how the context discovery algorithm utilizes them to accomplish its task.

4.2 General Approach

Figure 4.3 shows a high level architecture of CueNet. The major functional blocks consist of a data integration system (left), which provides a uniform query interface to a multitude of autonomous data sources, which may reside within an enterprise or on the World-Wide

Web [51]; a specification of model describing real-world knowledge in terms of objects and their relations, along with any axioms and constraints to be imposed on instance graphs (top); since we are assisting face tagging application, the final block (right) consists of a set of hooks to invoke appropriate face tagging algorithms by providing a set of candidate for the input photo. In this work, we shall assume verification semantics in such a tagging algorithm, where given an input photo and a candidate person, the algorithm returns true or false (with a confidence score). Face recognition models would have to be retrained when the candidate set changes. Also, as described in chapter 3, the state-of-the-art techniques for face verification perform much more reliably than their recognition counterparts. At the heart of CueNet, lies the context discovery algorithm. Given a photo the algorithm constructs a context network with all the known information. Using the knowlege base, the algorithm constructs queries to be executed on the interface provided by the data integration layer. Objects which are returned are merged into existing context network. New entities in the network are passed to the face tagging algorithm to check for their presence in the photo. If they are present, the context network is altered to reflect this fact. The execute-merge cycle is iteratively performed until all the faces are tagged, or the data integration module is unable to furnish any new data.

4.3 Execution Trace

In this section, we will trace the execution on two different photos, to see how the different modules interact to produce context networks, and how they are used to tag faces. The first example will be a relatively simpler one, requiring only 2 data sources, whereas the second photo will require multiple sources to sucessfully tag all photos.

4.3.1 Simple Case

Consider the photo shown in figure 4.4. For the purposes of this trace, we assume that we have access to the sources shown in figure 4.5 through the data integration module. Given, an input photo, the knowledge base is queried to find what other objects can be associated with an photo object. The KB stores the information that every photo consists of an EXIF header, which stores timestamp and location coordinates and a fact which states that every photo is owned by a user object, where **owner-of** is a relationship described in the KB. This knowledge is used to construct the context network shown in figure 4.6.

Figure 4.4: Input Photo.

Figure 4.5: Available Data Sources.

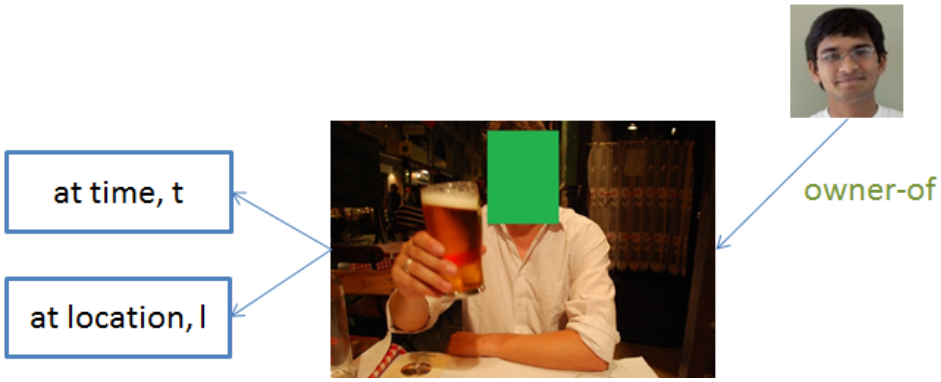


Figure 4.6: Context Network built after User Information and EXIF sources are queried and merged.

Now, the algorithm traverses the graph to list all the possible queries it can execute on the data integration layer. Given the knowledge that entities participate in events, and events

can contain participants, it generates the following queries:

- Does any data source contain participant information related to the photo capture event?
- At time t, and location l, which events contain the owner (entity:ArjunSatish) as a participant?

The data integration system looks at the different sources and says that none of them store information about photo capture events, and skips executing the first query. But many sources describe events, and store their participants too (Google Calendar, Facebook, Conference Calendar). These query is converted to their native formats (API calls or relational database queries) and results, are sent back to the data integration module. We see that there was a calendar entry returned be the Google Calendar source, as shown in 4.7. This information is now merged with the existing context graph to produce a context graph similar to that shown in 4.8.

Figure 4.7: Calendar Event.

Now, we have new entities related to the photo. The face verification algorithm is invoked with the new set of candidates. It must be noted that this verification problem is much easier than trying to verify out of many thousands of candidates. Once the correct entity is identified, the photo is annotated as shown in figure 4.9.

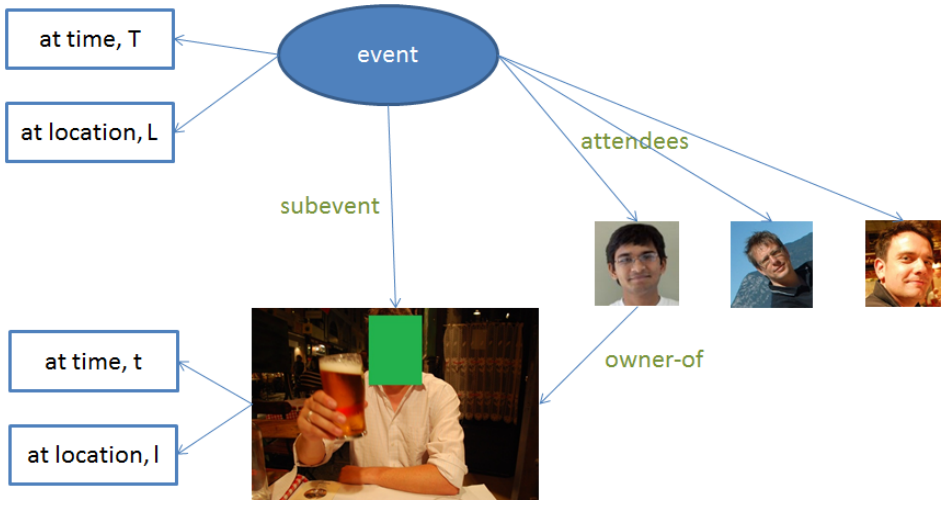


Figure 4.8: Context Network after integrating calendar information.

One last look at the source diagram in figure 4.10 shows which data sources revealed interesting information related to this photo. In this case, EXIF provided some relevant context on when and where the photo was taken. The owner's personal calendar provided information on what event was occurring during the time of photo capture, and who else was involved in it.

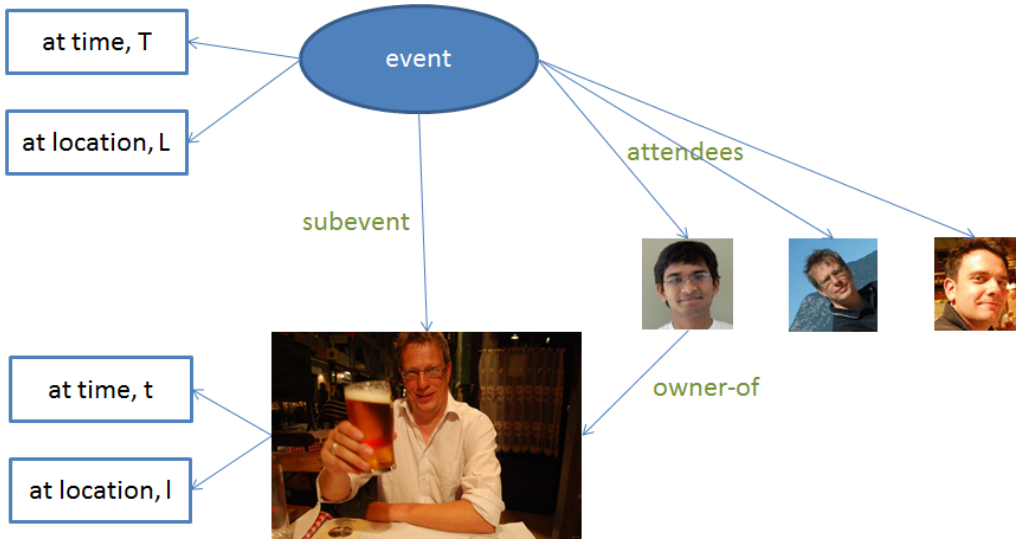


Figure 4.9: Face tagged with the correct person.

Figure 4.10: Highlighted Sources Provided Relevant Context.

4.3.2 Complex Case

Now, we will consider a more complex case which requires more than just metadata and personal sources for successful tagging. The photo under consideration is shown in 4.11. We will use the same set of data sources, shown again in 4.12.

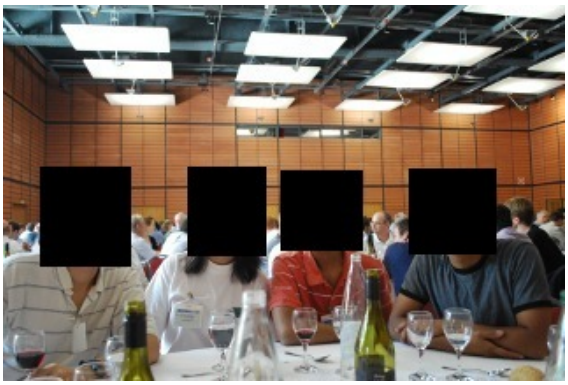


Figure 4.11: Input Photo.

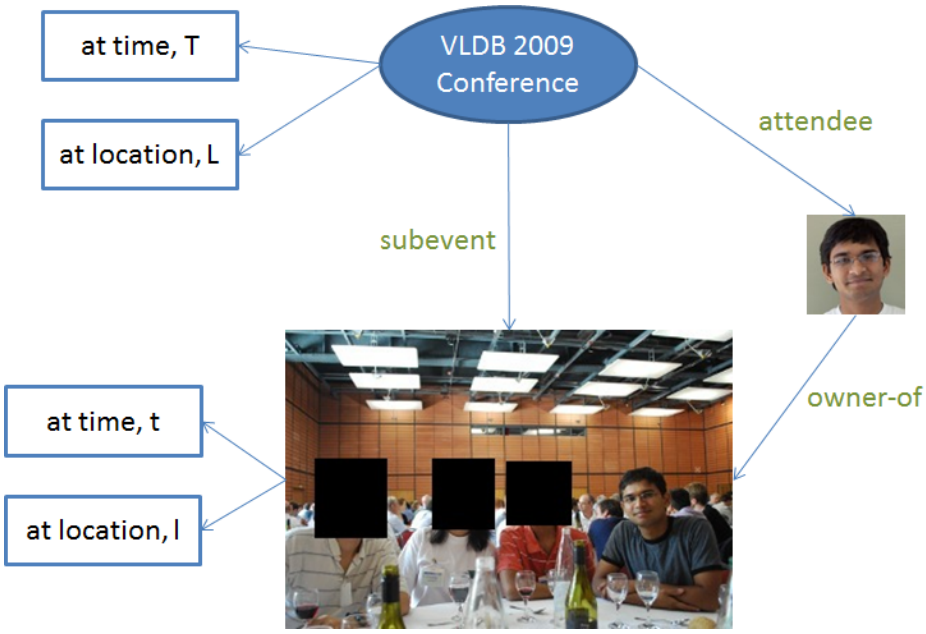
Figure 4.12: Available Data Sources.

Figure 4.13: Context Network built after User Information and EXIF sources are queried and merged.

Using metadata sources and personal information, we arrive at the context network shown in figure 4.13. The procedure until here is exactly same as that for the previous scenario. Now, given the known state of the world, if we invoke the face verification tools, we discover that the owner is actually present in the photo (figure 4.14). In this case, the candidate set contains just one entity, and therefore reduces the complexity of the tagging algorithm.

The next query generated by the system is to discover what the (entity:ArjunSatish) was doing at this time? But, this time we find that the conference calendar holds the answer.

Figure 4.14: Context Network built after User Information and EXIF sources are queried and merged.



At this point, the conference event is known to our knowledge base to have a definite structure, in terms of keynote, session and talk events with lunch/coffee breaks interleaved, and having many attendees. So it immediately queries the conference source to find and merge all of these objects. It discovers that the photo was taken during a break event, and that the conference (VLDB 2009) has many hundreds of participants, as shown in figure 4.16.

Figure 4.16 shows the various attendees discovered by the algorithm from the conference source. But finding 3 candidates from hundreds is an equally challening task. Before invoking the face tagging algorithm, we want to see if we can discover any more relations between the objects in the context network. So the discovery algorithm consults the knowledge base

Figure 4.15: Context Network after querying conference sources.

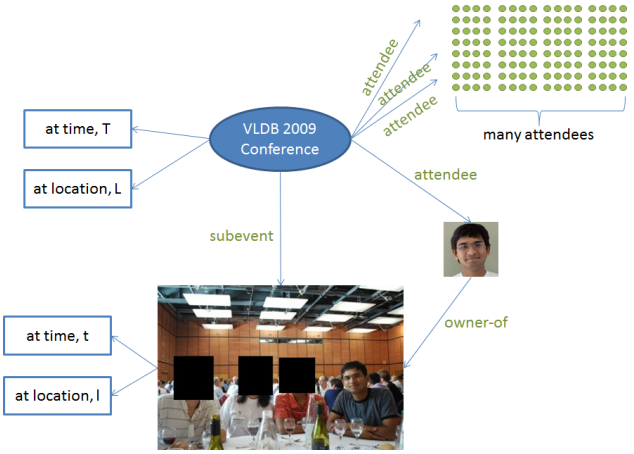


Figure 4.16: Context Network after discovering conference attendees.

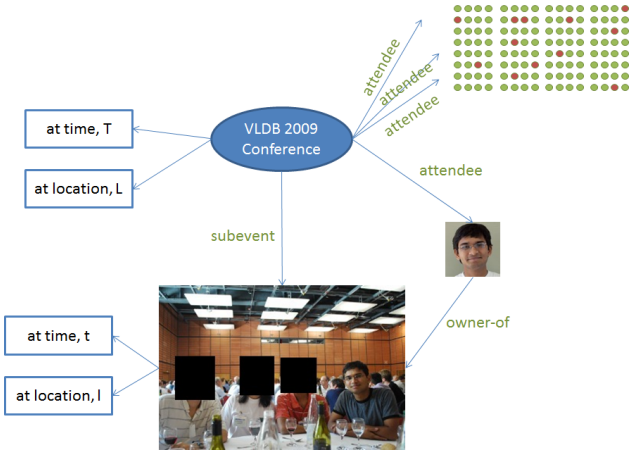


Figure 4.17: Context Network after discovering relations between attendees and owner.

to find that entities can be related through a **friend-of** relation. So it queries all known sources to find friend relations, and finds that Facebook, Gmail and Twitter are sources which store data containing this relation. Querying it reveals that a few of the entities who were present at the conference were related to the user, and therefore have a bigger chance of appearing in the photo. The face verifier is invoked only with these candidates, for potential true positives. By doing this we tag two more faces in the photo. The context network is shown in the figure 4.18.

Since we have more candidates tagged in the photo, we can repeat the above procedure to discover more relation between the entites related to the photo and those who are present in the conference. This time results are returned from Gmail, and none from Facebook and Twitter (because these people had sent emails to each other during the conference, but did not connect through Facebook or Twitter). The changes in the context network are shown in 4.20 and 4.21. Figure 4.22 highlights all sources which returned relevant context for this trace.

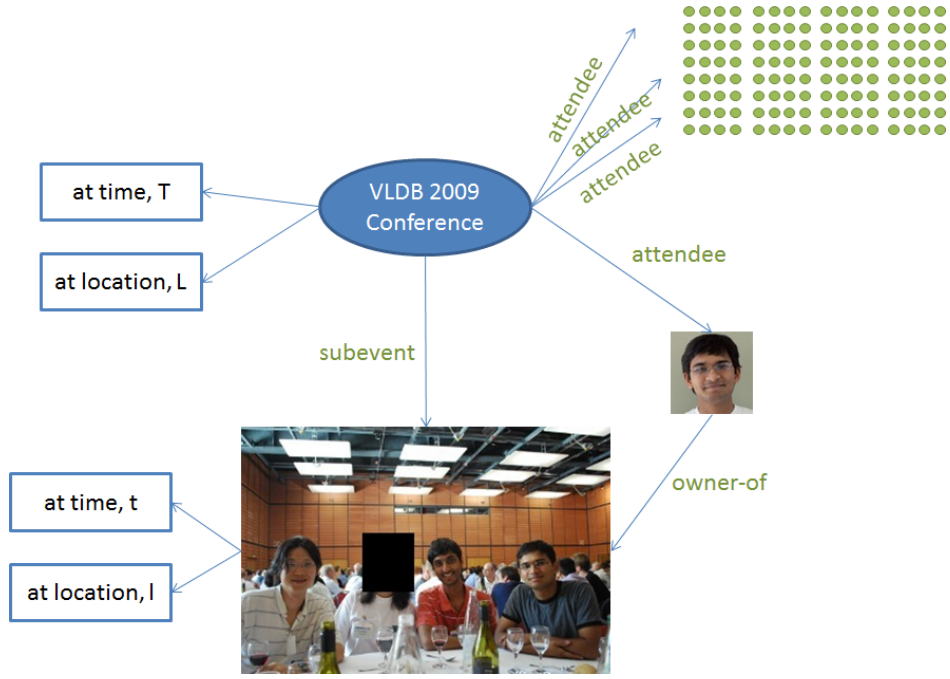


Figure 4.18: Context Network after discovering social relations.

Figure 4.19: Sources used so far.

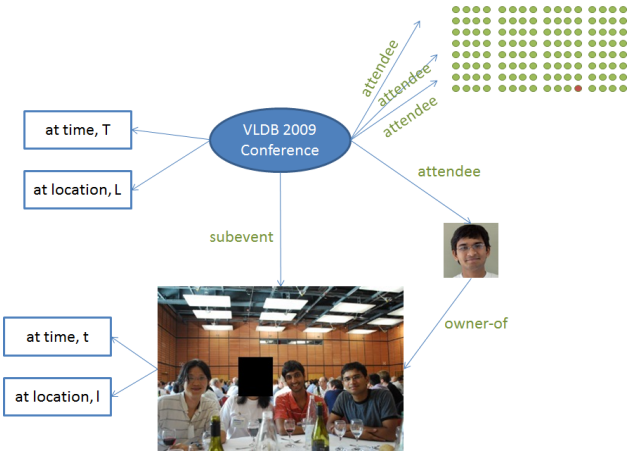


Figure 4.20: Context Network after discovering further social relations.

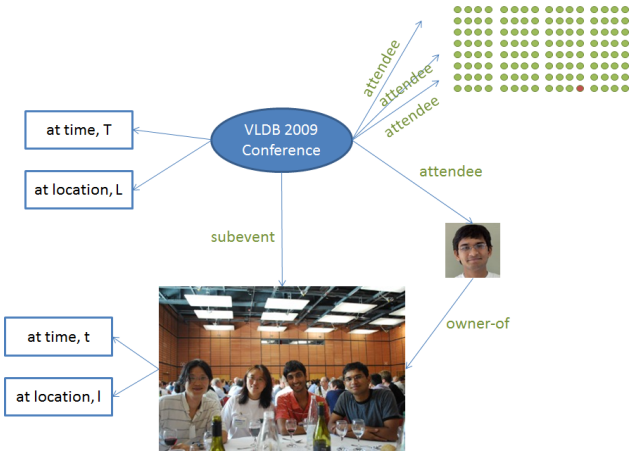


Figure 4.21: Context Network after tagging all faces.

Figure 4.22: Sources used to tag all faces.

4.4 Event Model

Our ontologies extend the E* model[49] to specify relationships between events and entities. Specifically, we utilize the relationships "**subevent-of**", which specifies event containment. An event $e1$ is a subevent-of of another event $e2$, if $e1$ occurs completely within the spatiotemporal bounds of $e2$. Additionally, we utilize the relations **occurs-during** and **occurs-at**, which specify the space and time properties of an event. Also, another important relation between entities and events is the "**participant**" property, which allows us to describe which entity is participating in which event. It must be noted that participants of a subevent are also participants of the parent event. A participation relationship between an event and person instance asserts the presence of the person within the spatiotemporal region of the event. We argue that the reverse is also true, i.e., if a participant P is present in $\mathcal{L}_P$ during the time $\mathcal{T}_P$ and an event E occurs within the spatiotemporal region $< \mathcal{L}_E, \mathcal{T}_E >$, we say P is a participant of E if the event's spatiotemporal span contained that of the participant.

$$\texttt{participant}(\text{E, P}) \iff (\mathcal{L}_P \sqsubset_L \mathcal{L}_E) \wedge (\mathcal{T}_P \sqsubset_T \mathcal{T}_E) \tag{4.1}$$

The symbols $\sqsubset_L$ and $\sqsubset_T$ indicate spatial and temporal containment respectively. Please refer to [49] for more details. In later sections, we refer to the location and time of the event, $\mathcal{L}_E$ and $\mathcal{T}_E$ as E.**occurs-at** and E.**occurs-during** respectively.

4.5 Data Sources

The ontology makes available a vocabulary of classes and properties. Using this vocabulary, we can now declaratively specify the schema of each source. With these schema descriptions, CueNet can infer what data source can provide what type of data instances. For example, the framework can distinguish between a source which describes conferences and another

which is a social network. We use a LISP like syntax to allow developers of the system to specify these declarations. The example below describes a source containing conference information.

```
(:source conferences
   (:attrs url name time location title)
   (:rel conf type-of conference)
   (:rel time type-of time-interval)
   (:rel loc type-of location)
   (:rel attendee type-of person)
   (:rel attendee participant-in conf)
   (:rel conf occurs-at loc)
   (:rel conf occurs-during time)
   (:axioms
      (:map time time)
      (:map loc location)
      (:map conf.title ltitle)
      (:map conf.url url)
      (:map attendee.name name)))
```

A source declaration comprises of a single nested s-expression. We will refer to the first symbol in each expression as a keyword, and the following symbols as operands. This above declaration uses five keywords (`source`, `attrs`, `rel`, `axioms`, `map`). The `source` keyword is the root operator, and declares a unique name of the data source. The source mapper can be queried for finding accessors using this name. The `attrs` keyword is used to list the attributes of this source. Currently we assume a tuple based representation, and each operand in the attrs expression maps to an element in the tuple. The `rel` keyword allows construction of a relationship graph where the nodes are instances of ontology concepts.

And edges are the relationships described by this particular source. In the above example, we construct individuals *conf*, *time*, *loc* and *attendee* who are instances of the *conference*, **time-interval**, **location** and **person** class respectively. We further say that attendee is a ***participant of*** the conference, which ***occurs-at*** location loc and ***occurs-during*** the interval time. Finally, the `mapping axioms` are used to map nodes in the relationship graph to attributes of the data source. For example, the first axiom (specified using the map keyword) maps the time node to the time attribute. The third map expression creates a literal called title, and associates it to the conference node, whose value comes from the ltitle attribute of the conference data source.

Formally, we represent the given ontology as O. The various classes and properties in O are represented by C^O and P^O respectively. Since our upper ontology consists of DOLCE and E*, we assume the inclusion of the classes `Endurant`, `Perdurant`, `Event` and `Person` in C^O. Each source S consists of three parts, a relation graph $G^S(V^S, E^S)$ where the nodes $V^S \in C^O$, specify the various "things" described by the source. The edges $E^S \in P^O$ specify the relations among the nodes. Any graph retrieved from such a source is an instance of the relation graph, G^S. Further, the tuple A_T^S consists of the attributes of the data source. Finally, the mapping $M^S : \{G^S \rightarrow A_T^S\}$ specifies how to map different nodes in the relation graph to the different attributes of the native data source.

4.6 Conditions for Discovery

CueNet is entirely based on reasoning in the event and entity (i.e., person) domain, and the relationships between them. These relationships include participation (event-entity relation), social relations (entity-entity relation) and subevent relation (event-event). For the sake of simplicity, we restrict our discussions to events whose spatiotemporal spans either completely overlap or do not intersect at all. We do not consider events which partially overlap. In

order to develop the necessary conditions for context discovery, we consider the following two axioms:

Entity Existence Axiom: Entities can be present in one place at a time only. The entity cannot exist outside a spatiotemporal boundary containing it.

Participation Semantics Axiom: If an entity is participating in two events at the same time, then one is the subevent of the other.

Given, the ontology O, we can construct event instance graph $G^I(V^I, E^I)$, whose nodes are instances of classes in C^O and edges are instances of the properties in P^O. The context discovery algorithm relies on the notion that given an instance graph, *queries* to the different sources can be automatically constructed. A query is a set of predicates, with one or more unknown variables. For the instance graph $G^I(V^I, E^I)$, we construct a query $Q(D, U)$ where D is a set of predicates, and U is a set of unknown variables.

Query Construction Condition: Given an instance graph $G^I(V^I, E^I)$ and ontology $O(C^O, P^O)$, a query $Q(D, U)$ can be constructed, such that D is a set of predicates which represent a subset of relationships specified in G^I. In other words, D is a subgraph induced by G^I. U is a class, which has a relationship $r \in P^O$, with a node $n \in D$. Essentially, the ontology must prescribe a relation between some node n through the relationship r. In our case, the relation r will be either a **participant** or **subevent** relation. If the relationship with the instances does not violate any object property assertions specified in the ontology, we can create the query $Q(D, U)$.

Identity Condition: Given an instance graph $G^I(V^I, E^I)$, and a result graph $G^R(V^R, E^R)$ obtained from querying a source, we can merge two events only if they are identical. Two nodes $v_i^I \in V^I$ and $v_r^R \in V^R$ are identical if they meet the following two conditions **(i)** Both v_i^I and v_r^R are of the same class type, and **(ii)** Both v_i^I and v_r^R have exactly overlapping

spatiotemporal spans, indicated by the $=_L$ and $=_T$. Mathematically, we write:

$$\begin{aligned} v_i^I = v_r^R \iff & (v_i^I.\textbf{type-of} = v_r^R.\textbf{type-of}) \wedge \\ & (v_i^I.\textbf{occurs-at} =_L v_r^R.\textbf{occurs-at}) \wedge \\ & (v_i^I.\textbf{occurs-during} =_T v_r^R.\textbf{occurs-during}) \end{aligned} \tag{4.2}$$

Subevent Condition: Given an instance graph $G^I(V^I, E^I)$, and a result graph $G^R(V^R, E^R)$ obtained from querying a source, we can construct a subevent edge between two nodes $v_i^I \in V^I$ and $v_r^R \in V^R$, if one is spatiotemporally contained within the other, and has at least one common `Endurant`.

$$\begin{aligned} & v_i^I \sqsubset_L v_r^R, \\ & v_i^I \sqsubset_T v_r^R \end{aligned} \tag{4.3}$$

$$v_i^I.\textbf{Endurants} \cap v_r^R.\textbf{Endurants} \neq \{\phi\} \tag{4.4}$$

Here $v_i^I.\textbf{Endurants}$ is defined as a set $\{w | w \in V_i^I \wedge w.\text{type-of=Endurant}\}$. If equation (4.4) does not hold, we say that v_i^I and v_r^R co-occur.

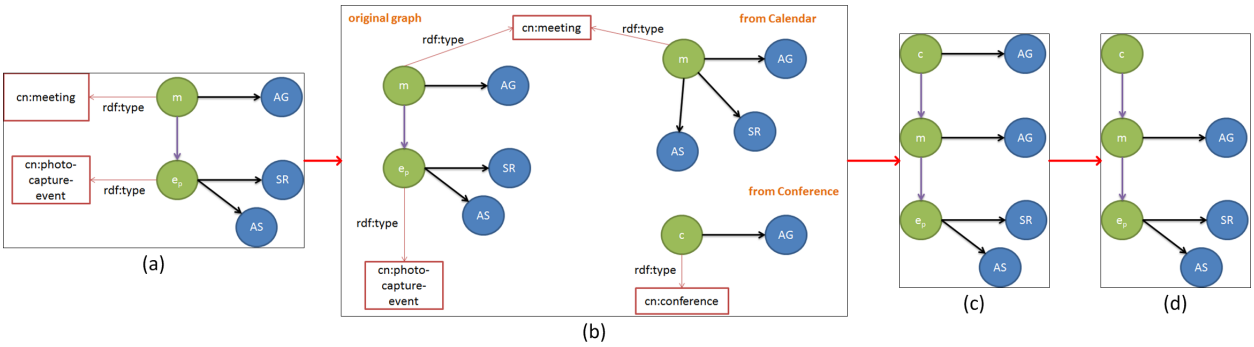


Figure 4.23: The various stages in an iteration of algorithm 1.

Merging Event Graphs: Given the above conditions, we can now describe an important building block for the context discovery algorithm: the steps needed to merge two event

graphs. An example for this is shown in figure 4.23(b-d). Given the event graph consisting of the photo capture event on the left of (b) and a meeting event m and conference event c, containing their respective participants. In this example, the meeting event graph, m is semantically equivalent to the original graph. But the conference event, c is telling that the person AG is also participating in a conference at the time the photo was taken. The result of merging is shown in (d). An event graph merge consists of two steps. The first is a `subevent hierarchy join`, and the second is a `prune-up` step.

Given an original graph, O_m, and a new graph N_m, the join function works as follows: All nodes in N_m are checked against all nodes in O_m to find identical counterparts. For entities, the identity is verified through an identifier, and for events, equation (4.2) is used. Because of the entity existence and participation semantics axioms, all events which contain a common participant are connected to their respective super event using the subevent relation (equations (4.3) and (4.4) must be satisfied by the events). Also, if two events have no common participant, then they can be still be related with the subevent edge, if the event model says it is possible. For example, if in a conference event model, keynotes, lunches and banquets are declared as known subevents of an event. Then every keynote event, or banquet event to be merged into an event graph is made a subevent of the conference event, if the equation (4.3) holds between the respective events.

It must be noted that node AG occurs twice in graph (c). In order to correct this, we use the participation semantics axiom. We traverse the final event graph from the leaves to the root events, and remove every person node if it appears in a subevent. This is the `prune-up` step. Using these formalisms, we now look at the working of the context discovery algorithm.

4.6.1 Context Discovery Algorithm

Algorithm 1 below outlines the tail recursive discovery algorithm. The input to the algorithm is a photo (with EXIF tags) and an associated owner (the user). It must be noted that by seeding the graph with owner information, we bias the discovery towards his/her personal information. An event instance graph is created where each photo is modeled as a photo capture event. Each event and entity is a node in the instance graph. Each event is associated with time and space attributes. All relationships are edges in this graph. All EXIF tags are literals, related to the photo with data property edges. Figure 4.23 graphically shows the main stages in a single iteration of the algorithm.

The event graph is traversed to produce a queue of entity and event nodes, which we shall refer to as DQ (discovery queue). The algorithm consists of two primary functions: **discover** and **merge**. The discover function is tail recursive, invoking itself until a termination condition is reached (when at most k tags are obtained for all faces or no new data is obtained from all data sources for all generated queries). The behavior of the query function depends on the type of the node. If the node is an event instance, the function consults the ontology to find any known sub-events, and queries data sources to find all these subevents, its properties and participants of the input event node. On the other hand, if it is an entity instance, the function issues a query to find all the events it is participating in.

Results from data source wrappers are returned in the form of event graphs. These event graphs are merged into the original event graph by taking the following steps. First, it identifies **duplicate** events using the conditions mentioned above. Second, it identifies subevent hierarchies using the graph merge conditions described above, and performs a **subevent hierarchy join**. Third, the function **prune-up** removes entities from an event when its subevent also lists it as a participant node. Fourth, **push-down** is the face verification step if the number of entities in the parents of the photo-capture events is small (less than T).

Algorithm 1: The Context Discovery Algorithm

Data: A photograph H, with a set of detected faces F. Voting threshold, T. The owner O of the photo.

Result: For each face f ∈ F, a set of atmost k person tags.

```
1  begin
2
3      function discover(): {
4          while (DQ is not empty): {
5              node = DQ.deque()
6              results = query (node)
7              E ← merge (E, results)
8              if (termination_check()):
9                  return prepare_results();
10         }
11         reconstruct DQ ← E
12         discover()
13     }
14
15     function merge(O, N): {
16         remove_duplicates()
17         M ← subevent_hierarchy_join(O, N)
18         prune_up(M)
19         if (less than T new candidates were discovered):
20             push_down(M)
21         else:
22             vote_and_verify(M)
23         return M;
24     }
25
26     E ← construct event graph with H and O
27     construct discoverable nodes queue, DQ ← E
28     return discover()
29 end
```

Push down will try to verify if any of the newly discovered entities are present in the photo and if they are (if the tagging confidence is higher than the given threshold), the entities are removed from the super event, and linked to the photo capture event as its participant. On the other hand, if this number is larger than T, the algorithm initiates the **vote-and-verify** method, which ranks all the candidates based on social relationships with people already identified in the photo. For example, if a candidate is related to two persons present in the

photo through some social networks, then its score is 2. Ranking is done by simply sorting the candidate list by descending order of score. The face verification runs only on the top ranked T candidates. If there are still untagged faces after the termination of the algorithm, we vote over all the remaining people, and return the ranked list for each untagged face.

Figure 4.23 shows the various stages in the algorithm graphically. (a) shows an example event graph describing a photo taken at a meeting. The meeting consists of three participants AG, SR and AS. The photo contains SR and AS. (b) shows two events returned from the data sources. One is a meeting event which is semantically identical to the input. The other is a conference event with AG. (c) shows the result of merging these graphs. (d) The `prune-up` function removes the duplicate reference to AG. A live visualization of these steps for different photos can be found at http://cuenet.site44.com.

4.7 Merging Context Networks

In this section, we look more closely at the merge function. Algorithm 2 presents the pseudocode for merging a secondary context network, S, into a primary context network P. A terminology of primary and secondary is to signify that all data instances from a secondary network will be merged into a primary network. While merging networks, we also assume that events have atmost one super-event. Thus, no diamond structures are found in either network. In, algorithm 2 shows how two networks with a single root event are merged. A root event is one which has no super-events. The symbol $\forall_{se}$ stands for "for all subevents".

Once the two root nodes, Pr and Sr have been identified, we descend the subevent trees of the two context networks, and do one of the following operations. For each subevent of Pr, we check if any subevent is equal to Sr, then merge the information from Sr to this subevent, and continue recursively merging the children of Sr into the children of the subevent. If a

Algorithm 2: The Merge Algorithm

Data: Context Network P and S, where S will be merged into P.
Result: All event instances in S will be merged into P.

```
1  begin
2
3      function merge(P, S): {
4          [Pr] = root event of P
5          [Sr] = root event of S
6          recursive_merge(P, Pr, S, Sr)
7      }
8
9      function recursive_merge(P, Pr, S, Sr): {
10         addAsSubevent = true
11         containedSubevents ← {φ}
12         ∀se (ps of P) {
13             if (equals(ps, Sr)) {
14                 addAsSubevent = false
15                 mergeInformation(ps, Sr)
16                 ∀se (s of Sr): recursive_merge(P, ps, S, se)
17             }
18             if (contains(ps, Sr)) {
19                 addAsSubevent = false
20                 recursive_merge(P, ps, S, Sr)
21             }
22             if (contains(Sr, ps)) {
23                 addAsSubevent = false
24                 containedSubevents.add(ps)
25             }
26         }
27
28         if (addAsSubevent) {
29             addSubeventEdge(Pr, Sr)
30             if (containedSubevents.size() > 0) {
31                 removeSubevents(Pr, containedSubevents)
32                 createSubevents(Sr, containedSubevents)
33             }
34             ∀se (s of Sr): recursiveMerge(subtree, Sr, other, s);
35         }
36     }
37
38 end
```

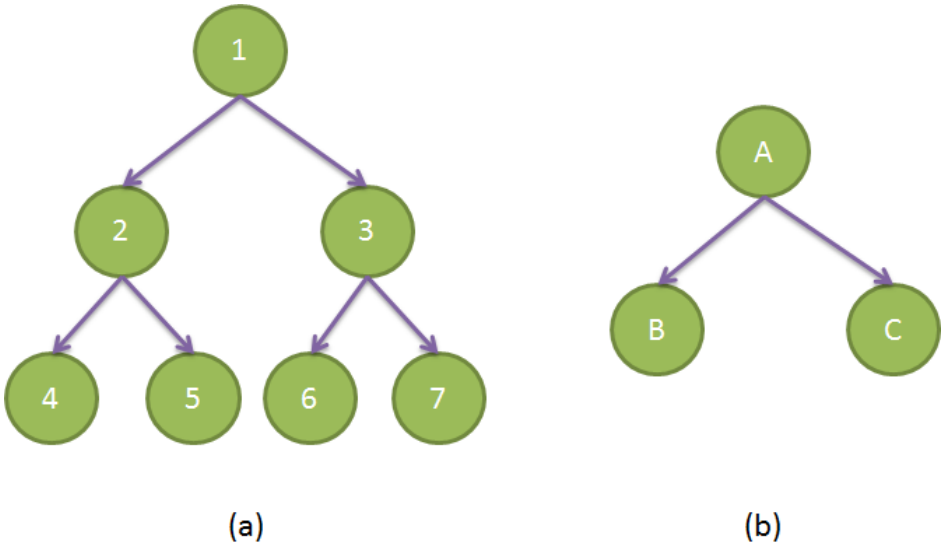


Figure 4.24: (a) Primary and (b) Secondary Example Networks.

subevent of Pr contains the new instance, Sr, we simply continue recursion. But, if Sr can contain the subevent node, then we add of the siblings sub-events which can become subevents of Sr to a list `containedSubevents`, and add Sr as a subevent of Pr, remove all the children from Pr in `containedSubevents`, and add them as subevents of Sr. Recursion is continued on the children of Sr from the newly connected Sr node in the primary network. Any new literal properties, spatio-temporal attributes and participant information in events which exist in the primary networks are copied using the `mergeInformation` function.

Let's consider an example to understand the merge function. Assume we have to merge network 4.24(b) into 4.24(a). To simplify the example, we will assume that all the events are occurring at the same location. The time intervals occupied by the events are provided in the table 4.1 below.

We first check if the root of the primary would contain the secondary. Since, event `1`'s time interval completely contains event `A`, this is true, and we initiate the merge function where `Sr=A, Pr=1`. The subevents `2, 3` of event `1` are both contained within A, the lines 23-24 from the algorithm populate the `containedSubevents` list. This list is used to remove the

Event	Time Interval
1	0 - 120
2	10 - 45
3	70 - 100
4	10 - 15
5	28 - 37
6	70 - 80
7	90 - 100

Event	Time Interval
A	10 - 105
B	10 - 45
C	80 - 90

Table 4.1: Time Intervals for Events in Primary and Secondary Networks shown in 4.24

edges between `1` and `2, 3` and create subevent edges between `1` and `A` and A and `2, 3` in the lines 28-36. The new primary network is shown in figure 4.25(a). Now, the algorithm recursively proceeds to add events `B, C` as subevents of A in the primary network `Sr=A, Pr=B`. Since the temporal extents of `2` and `B` are identical, the event `B` is merged into `2` as shown in 4.25(b). All participant information which exists in `B`, and not in `2` is copied. This is done in lines 13-17 of the algorithm. Finally, the event `C` is compared with event `3`, and found to be contained within it. The recursion proceeds to the subevents of `3` in lines 18-21. Here, the algorithm realizes that none of the events `6, 7` could contain `3`, and hence it is made a subevent of `3` itself. This is done at the line 29. Figure 4.25(c) shows the final primary network with all events from the secondary merged into it.

4.8 Implementation

In this section, we will look at the current implementation of the CueNet framework. Figure 4.26 shows a detailed conceptual architecture of the framework. It consists of three verticals: the data integration modules on the left side, the discovery algorithm and its supporting components in the middle, and the candidate management and face verification tools on the right side. Let's look at them one by one. The data integration module is configured using a source script. This script contains the schema declarations of the various sources, and

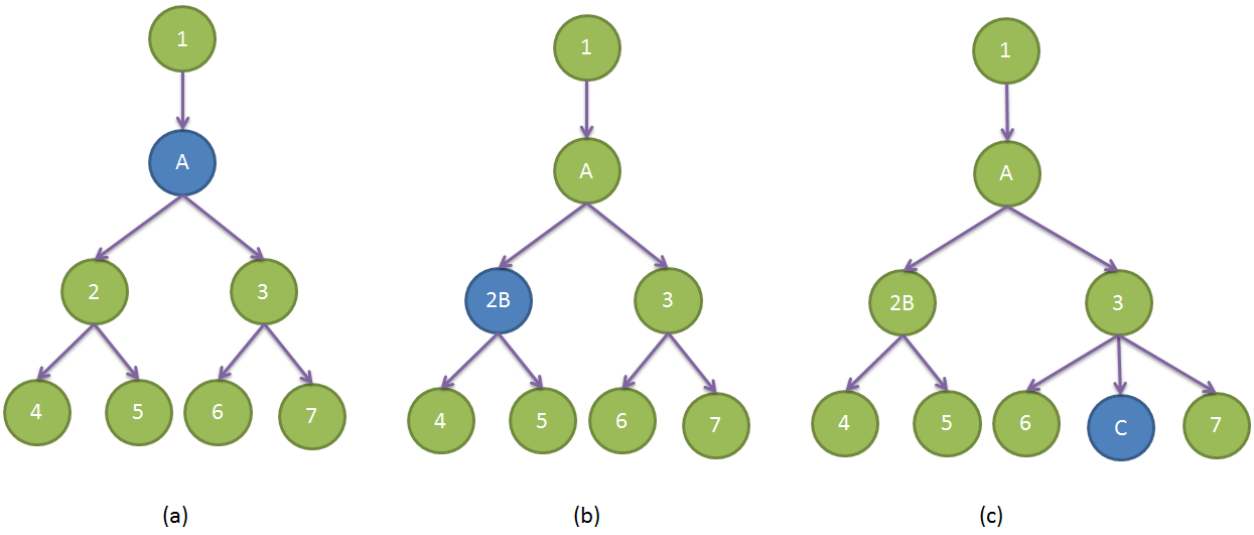


Figure 4.25: The three steps taken to merge the different nodes of the secondary into the primary network.

specifications of how the objects in these sources are related to the objects in the ontology. The complete script listing is provided in Appendix A. We use JavaCC to construct the compiler for this script. The schema and the relations are stored as graphs which can be queried by other modules. Any discover query to be executed on the multitude of sources is checked upon each graph to see if can respond to the type of query. For example, exclude social networks when queries are requesting event information. Each source requires mediator code to align its content with that of the tuples requested in the query engine. This approach is more reminiscent of GAV designs in data integration vocabulary. We chose this because of simpler query processing designs, and the fact that although we required to plug-in and out sources easily, these sources themselves changed very rarely. At the bottom of the data integration stack, we find some common utilities to make HTTP requests and DB query requests, which are used by the source mediators. The discover queries are created in SPARQL. The query engine uses Jena's SPARQL query processing framework, and extracts triple patterns and predicates using the framework's visitor pattern capabilities.

Now, lets look at the various components making up the discovery algorithm. Our knowledge base is specified using OWL ontologies. This is parsed using the open source Jena library, and

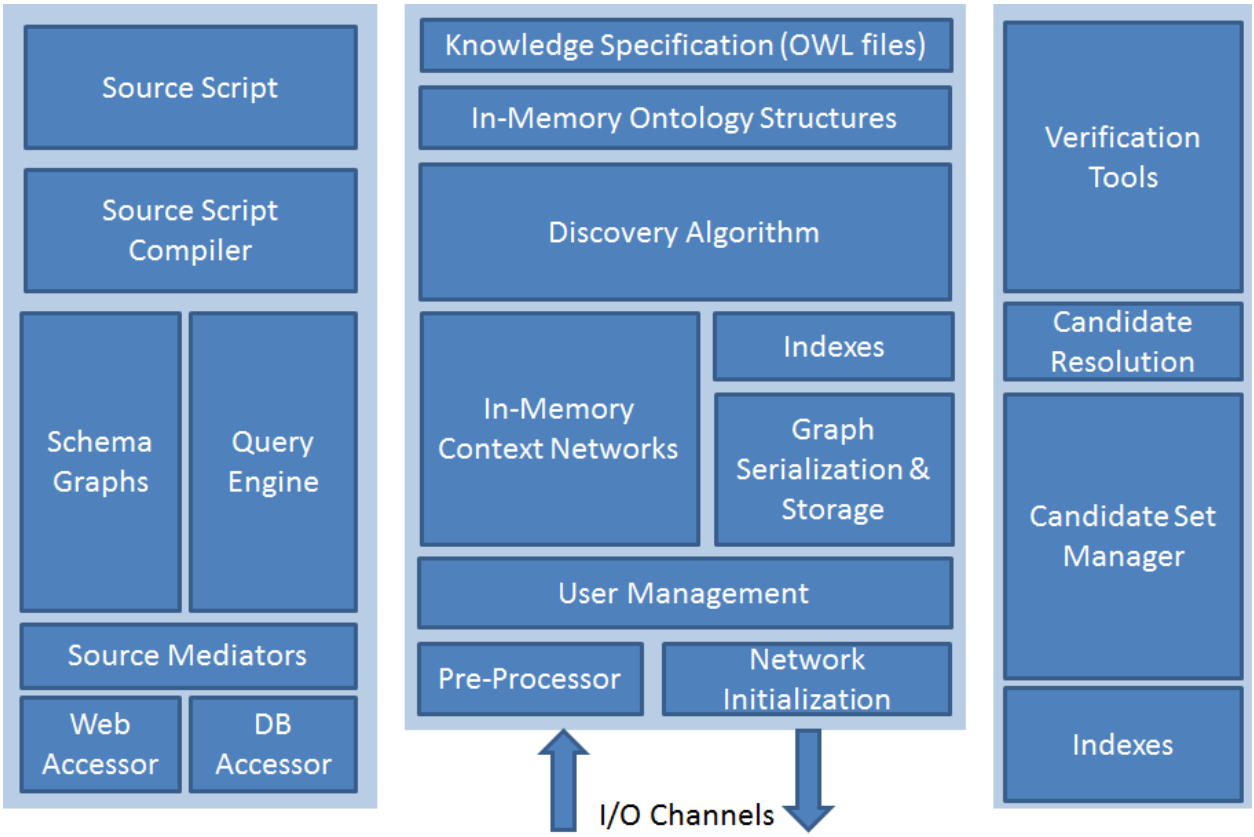


Figure 4.26: Main components of the current CueNet implementation.

the objects and relations specified are loaded into in memory DAGs. When a tag discovery request is made by a user, the discovery algorithm is initialized with a photo-capture-event, EXIF information is extracted through perl-based exiftool, and the seed context network is constructed and provided to the discovery algorithm. The algorithm initiates the discover and merge operations using the query engine, and the candidate manager. As new relations are discovered, the related entities are then checked for their presence in the photo using verification tools. We used face.com until its termination in mid-2012, after which we use the system based on [64] until early 2013, after which we resorted to manual verification upon the top ranked candidates.

When a tag request is initiated, a message is passed to the candidate manager, which scans the user's database to create a list of all entities. This is done by invoking scan queries on

all data sources through the query engine. Each data sources has a different representation for the same entity. The job of the candidate manager is to aggregate entities by using cues based on their common names or emails, pass them to an entity resolver (if the data is stored on a web-page), and if needed ask a user to correct any mistakes it made in this process. Ultimately, at the end of its processing, we obtain a list of candidates, each which holds identifiers which are local to each data source. When a context network is created by the source mediator, it looks up the candidate reference through the candidate manager before emitting the network back to the discover algorithm. This step makes the nature of data sources oblivious to the discovery algorithm.

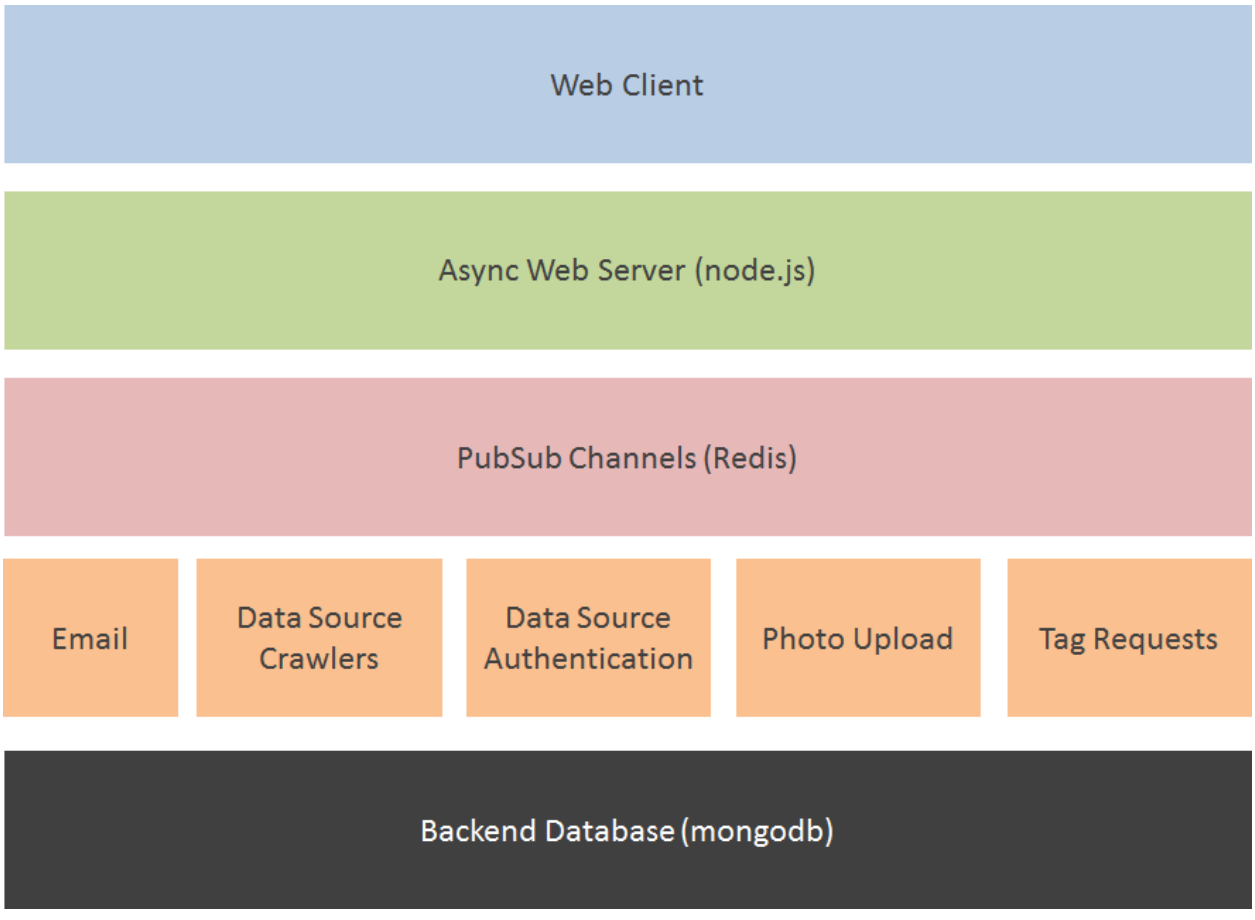


Figure 4.27: Web stack for data aggregation.

The web interface to the system is shown in figure 4.27. This stack is essential in aggregating data from various sources, and is instrumental in dealing with their intricacies related to

authentication, API requests and data formatting. Sources like Facebook and Google have rigid authentication mechanisms which need to be bypassed before any personal data can be accessed. Obtaining user permission needs to be done via a web client for Google APIs, whereas Facebook requires a valid session key from a user. These small details prompted us to implement the data aggregations using Javascript APIs which are served using an asynchronous web server based on node.js. The asynchronous nature of the server allows to develop web clients which execute a large number of HTTP callbacks without worrying about multi-threading issues – a design decision which is becoming increasingly popular for web architecture. Once the data has been aggregated at the client, it is passed back to the server which pushes it to specific process which can deal with processing it. For example, email information is sent to an independent Python process which initiates the IMAP protocol on the user's message server to aggregate emails. The Facebook events, social network, photo tag and Google calendar information is sent to a script which loads them into appropriate mongodb collections. Unstructured data like conference webpages are sent to a process which initiates the SNER over them to aggregate entities. These entities are stored in appropriate collections. The server communicates through these processes using REDIS, an in memory cache/database which supports PubSub channels. Multiple channels are created, one for each *type* of data source. Any process can listen to it, obtain the data, process it of its own accord.

Lastly, we will describe the data structures used to hold context networks. A context network is a directed graph containing event, and entity nodes, and their relationships. Any standard directed graph implementation can be used to hold this, but given the operations we perform on these network, some additional properties can be exploited to achieve faster merges. One operation we need often is to eliminate merging networks which provide no new information. For this purpose we modify the adjacency list structure to look like the one in figure 4.28. A traditional adjacency list holds a hash-table for nodes, and each node is associated with a linked list or another hash-table for the nodes which it connects to. Since IDs don't hold

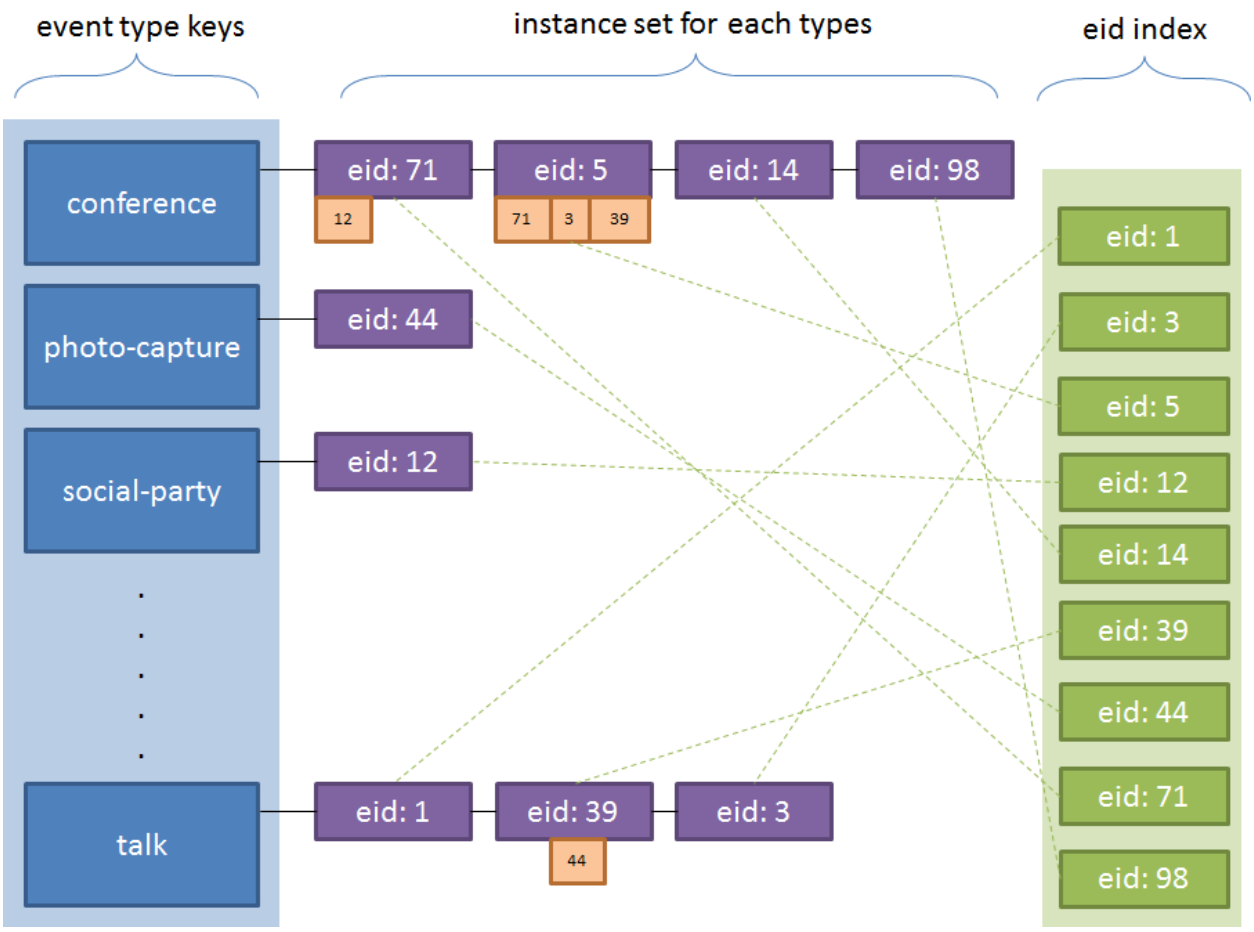


Figure 4.28: The data structure for maintaining Context Network.

any special meaning between context networks, we cannot rely only on them to distinguish between events in separate context networks. Event identity is established based on type, spatial and temporal information. For this purpose, our primary hash-table contains the **types** of events present in the network. In the example figure 4.28, we see the events of types `conference`, `photo-capture`, `social-party` and `talk` are contained in the network. For each event type, we associate a list of instances to it. This is implemented currently using a HashSet. Each instance in the context network is identified by a unique instance Id. This architecture allows us to quickly compare new events to existing ones and eliminate duplicates.

Edge relations are maintained by a list of (instance-id, edge-type) pairs. Thus traversing a

network implies starting from a root node, looking up the instance Ids of the outgoing edges by scanning the HashSet, looking the outgoing nodes in the adjacency list, and repeating the process. In order to avoid multiple lookups, we maintain an additional hash-table over instance IDs. Given an instance ID, this table allows fast lookups to avoid time spent in scanning very large sets which are created when there are many instances of the same type of events.

This architecture suffices because we discover context for a single photo at a time. In the case where an application needs to discover context for many events, we use a separate network for each atomic event under consideration, and assume that the context for one event is independent of the other. Thus, a merge operation reduces to finding the existing context network it can be merged with, and performing the merge only with one such network. This assumption will not be true if the events are spatio-temporally very close to each other. Here, information about one event will affect events in the other. This is beyond the scope of this dissertation. We also use a R-Tree built using the Java Spatial Index library to index the time intervals of the events of the various context networks (which do not have any super-events themselves), to reduce the search during the merge step. This increases the overall efficiency of discovery algorithm.

Chapter 5

Experiments and Analysis

In this chapter, we will present experiments to show the efficacy of the context discovery approach. Experiments will be performed on real-world data from real users using a variety of social and personal media applications, public data sources. The motivation behind these experiments is to tag faces in thousands of personal photos taken by various people at a different events. We will also perform experiments on generated data to simulate large scale use cases. Here, the motivation is to analyze the performance characteristics of the merge algorithm.

5.1 Experiments on Real-World Data

In this section, we analyze how CueNet helps tags a person's photos taken at different types of events. For a given users, we will construct a dataset consisting of photos taken at a particular event. For each of these users, we will create a candidate set by aggregating people over personal, public and social data sources. In order to evaluate CueNet, we will attempt to reduce this candidate set, and analyze how many of the faces can be correctly

tagged by this reduced set. We will also compare this performance over metrics like location based ranking, where candidates are ranked according to their last known location, and (if time permits) tie strength. Our final conclusion is that, in order to rank candidates for tagging faces in photos, CueNet provides an event-agnostic platform, where as other techniques perform inconsistently across different types of events.

5.1.1 Setup

We use photos taken during three different **types of events**: Social Parties, Academic Conferences and Trips. This diversity allows for different distributions of face tags across different aspects of a user's life. Social Parties generally tend to have close friends who are spatially co-located. Conferences tend to have people from different parts of the world, but those who are affiliated with the area of the conferences. Trips, cannot always rely on location as a useful metric and can involve people from either social, personal or professional circles from a user's life.

For each event type, we collect multiple datasets from 6 different people. A **dataset** consists of multiple photos during the event, the user's personal information, which contains information from sources like Google Calendar, personal email and profile information from social networks like Facebook and Twitter. We also collect a person's social networking information which consists of tweets written by the user or their friend during the time the event was occurring, the social network itself (friends on Facebook and Twitter, along with their profile information). Conference proceedings are downloaded from DBLP and the conference website. Facebook events are also obtained and stored in our database. Besides, location databases like Yahoo Placefinder were used to geocode addresses and reverse geocode EXIF GPS coordinates. We assume that all photos have a valid EXIF tag, especially the timestamp and GPS coordinates. This assumption is not a very hard one, as almost all photos

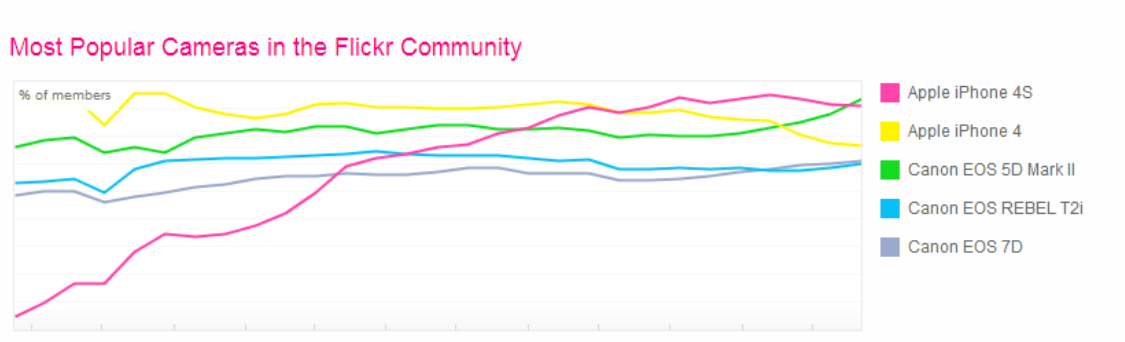


Figure 5.1: Popularity of iPhone at Flickr.com (October 2011).

captured in the last two years are through iPhone or Android smartphones, which add reasonably accurate GPS tags and accurate timestamps (where the phone clock is synced with the cell tower). The domination of iPhone in the photo market is shown in figure 5.1. The ground truth was annotated by the user with our annotation interface. For each photo, an annotation consisted of the ID of the person in the candidate set in it. Face verification was achieved initially with Face.com, but after their website was shutdown, we used the web service, Automatic Face Systems, maintained by Neeraj Kumar. After this service was terminated, we resorted to manual face verification. We use the manual verification for the experiments described below, unless otherwise mentioned.

In order to construct a **candidate set** for each dataset, we construct a so-called `CandidateSet`. A `CandidateSet` data structure performs two functions. It is a persistent set of candidates each of which have a unique identifier. Second, each data source, which has its own identifier for a candidate can look up the candidate set with this identifier to obtain the candidate set identifier. Here is an example of a simple candidate set:

```
[
{csid: 1010-poqs, name: 'Arjun Satish', email: 'arjun@uci.edu',
                                        email: 'arjun.satish@gmail.com',
                                        facebook-id: '656563332'}
{csid: 2010-pasd, name: 'Ramesh Jain', email: 'jain@ics.uci.ed',
```

```
                                        name: 'Dad',
                                        email: 'jain49@gmail.com',
                                        facebook-id: '10004290384'}
{csid: 1255-juen, name: 'Peter Boncz', confid: 'cf-12-candidate-322'}
{csid: 7585-kdye, name: 'Amarnath Gupta', email: 'gupta@sdsc.edu',
                                           twitter: 'aguptasd'}
{csid: 1111-bmel, name: 'Arjun', twitter: '@wicknicks'}
]
```

The above snippet is a part of the candidate set created during the tagging of my personal photos. A `CandidateSet` is a multimap where the keys are the unique IDs (shown as csid above). The values are a list of key-value attribute pairs. The key could be a global key, such as name or email address, which can be added by any data source. In the above example, different sources could contribute different names for the same entity. Same values for the same key are not duplicated. The list could contain keys which are local to a data source, for example the facebook identifier key: `facebook-id`. This primarily helps in querying the candidate set to obtain a reference to the candidate during the discovery phase.

The construction of the `CandidateSet` happens when the system is brought up. For each user, a unique `CandidateSet` is created, or loaded if it exists on disk. In order to create it, each data source is probed individually to provide a list of unique persons. The data source mediator checks if the user already exists in the `CandidateSet` through its local primary key, and if he does, adds additional key value pairs to the attribute list. If a candidate corresponding to its primary key does not exist, a new candidate is created in the set, and the local primary key is added as an attribute pair. This technique will fail to merge different identities of people if they use multiple email addresses or do not store email information on social networks. Thus, we create a merge file, which lists a set of merges between candidates. In the above example, we will use this functionality to combine

the entries with IDs `1111-bmel` and `1010-poqs` into a unified candidate. This gurantees that context discovered for one online identity is propagated to other identities of the same person.

We use 1889 photos taken at 17 different events in our face tagging experiment. Each photo contains one or more faces. We will denote each dataset as 'Di' (where $1 \leq i \leq 17$ for each dataset). Table 5.1 describes each dataset in terms of number of photos, unique annotations in ground truth, the year they were captured and the type of the event.

Dataset	Unique People	No. of Photos	Year	CandidateSet Size	Event Type
D1	43	78	2012	660	conference
D2	24	108	2012	660	conference
D3	6	16	2010	660	conference
D4	7	10	2010	660	conference
D5	36	80	2009	660	conference
D6	18	65	2013	660	conference
D7	7	11	2013	660	conference
D8	12	25	2009	1894	conference
D9	14	65	2011	215	party
D10	13	131	2010	561	party
D11	6	85	2008	656	party
D12	50	74	2012	1049	party
D13	19	330	2009	691	party
D14	14	363	2009	711	trip
D15	2	208	2010	715	trip
D16	4	217	2011	711	trip
D17	7	23	2011	584	trip

Table 5.1: Profile of datasets used in the experiments.

We divide the sources into different categories to facilitate a more general discussion. The categories are "Personal Information" (same as Owner Information in section 4.6.1), "Event sources", and "Social Networks". Event sources include Facebook events, Yahoo Upcoming web service, our conference events database among other sources. Social networks include Facebook's social graph. Personal information contained information about the user, and a link to their personal calendars. An annotation is considered "Out of Context Network" if

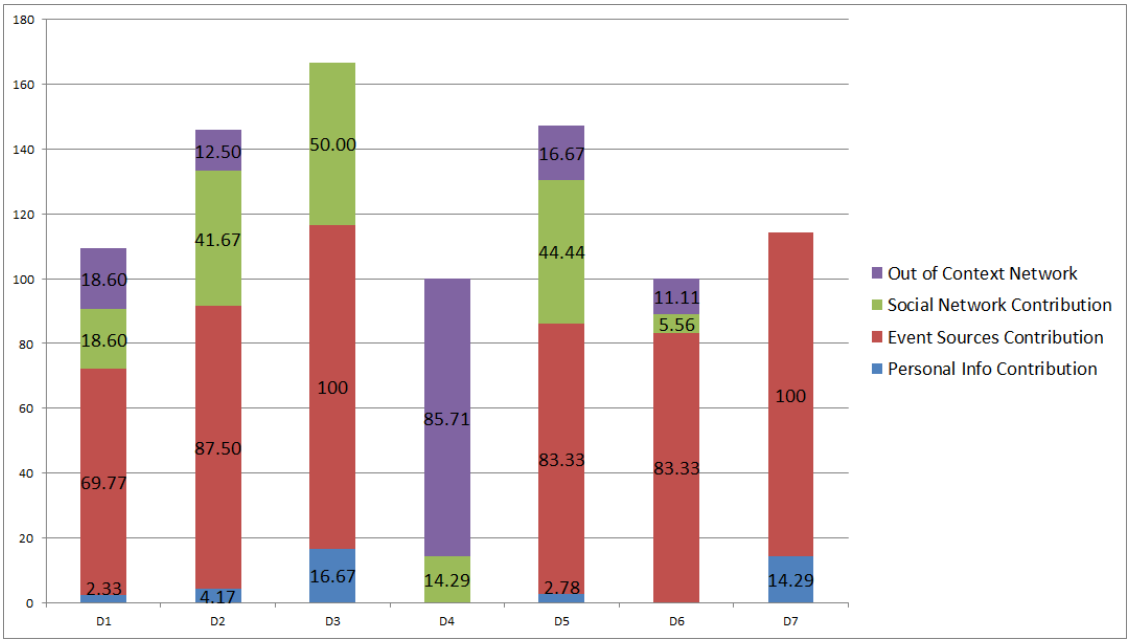


Figure 5.2: The distribution of annotations in the ground truth across various sources.

it is not in any of these sources.

Figure 5.2 shows the distribution of the ground truth annotations of the conference datasets across various sources, for each dataset. For example, the bar corresponding to D2 says that 87.5% of ground truth annotations were found in event sources, 41.67% in social networks, 4.17% in personal information and 12.5% were not found in any source, and therefore marked as "Out of Context Network". From this graph it is clear that event sources contain a large portion of ground truth annotations. Besides D4, a minimum of 70% of our annotations are found in event sources for all datasets, and for some datasets (D3, D7) all annotations are found in event sources. The sum total of contributions will add up to values more than 100% because they share some annotations among each other. For example, a friend on Facebook might show up at a conference to give the keynote talk.

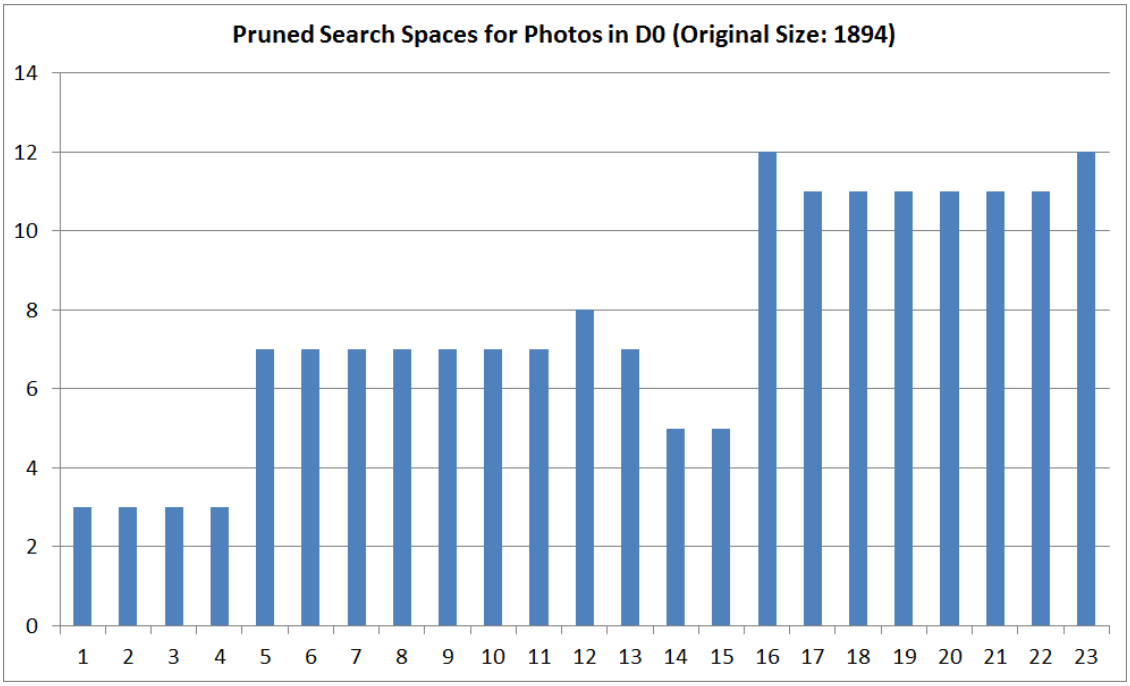


Figure 5.3: Pruned search space for photos in conference dataset D8.

5.1.2 Individual List Sizes in a Dataset

First, we look at how CueNet reduces the number of possible candidates for all photos in a dataset. For this setup, the complete candidate set L, contained 1894 labels (total number of people present at the conference, user's emails and social graph). The figure 5.3 shows various statistics for each photo, which includes the maximum size of the list which was generated by the discovery algorithm, the actual number of people in the photos, the number of true positives and false positives. As it can be seen, the size of the discovered set S, never exceeded 12. This is 0.5% of the original candidate list. Because the total number of possible participants (list size) was low, our False Positive rate (FP) was very low too. Most of the false positives were due to profile orientation of faces or obstructions (this was because the face detector was smart enough to pick up profile faces, but verification worked better only on frontal faces).

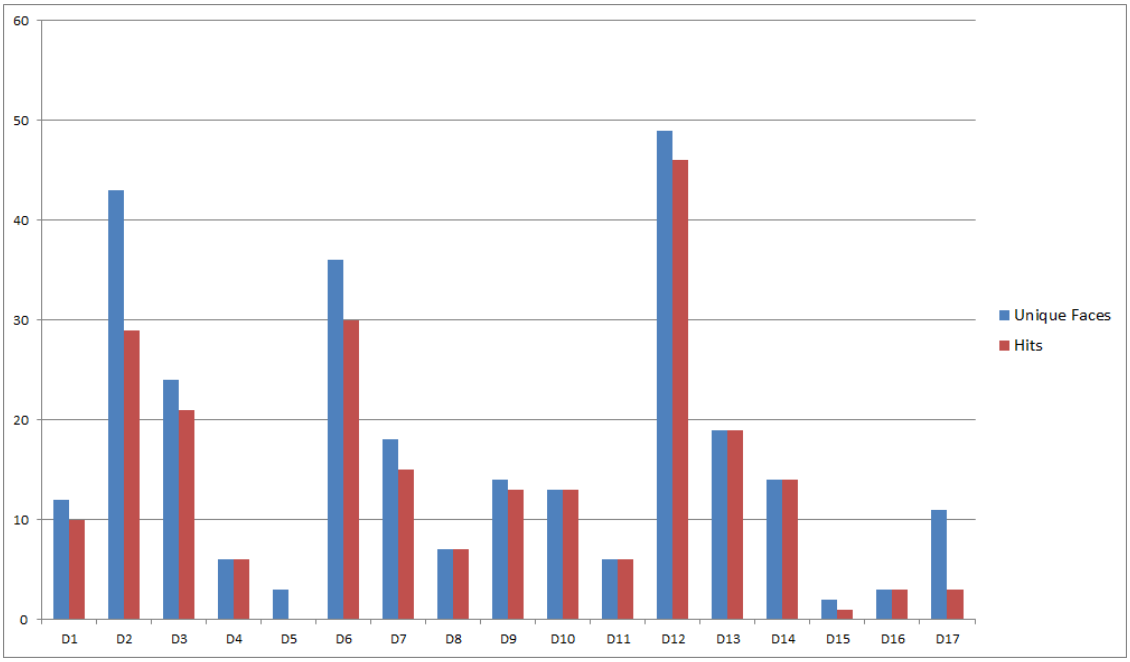


Figure 5.4: Hit counts for all datasets using Context Discovery Algorithm.

5.1.3 Results using Context Discovery

In order to evaluate 17 different datasets, we perform the following modification on each dataset. For a dataset, we select the photo with all annotations, and attempt to tag it. The numbers seen below will reflect the results for such a photo. Alternatively, the numbers below can be interpreted as tagging the event itself, and not individual photos. We do this modification to reduce the number of graphs drawn per experiment. We define a **hit** as a correctly tagged face. Figure 5.4 shows the hits for datasets using the context discovery algorithm. The blue bars show the number of unique faces in the dataset. As we can see the number of hits is equal to the number of unique faces, which implies that our context discovery algorithm was able to find the correct face tags for almost all datasets. Only for dataset 4, this difference is significant. This is because there was no contextual information related to the people in this dataset.

Most of the datasets contain hundreds of candidates. If we have to perform face verification on all datasets, then the discovery algorithm is not performing effectively. The fewer the

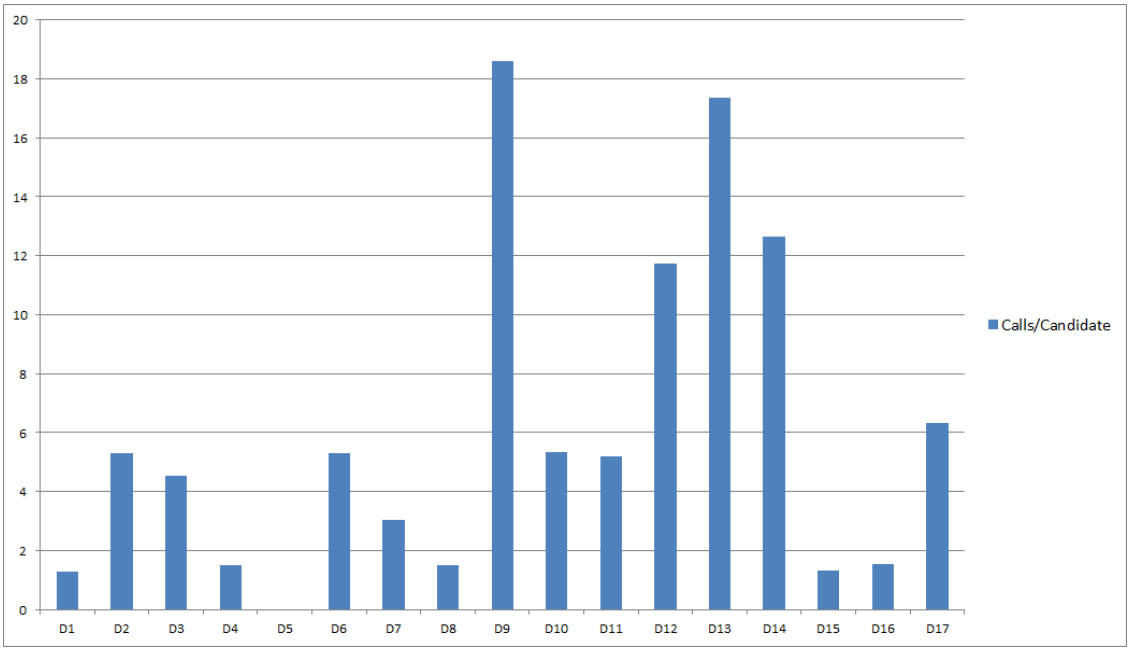


Figure 5.5: Verification Ratio (x 100) obtained using the Context Discovery Algorithm.

number of verifications, the better the accuracy of the algorithm. In order to quantify this, we introduce the metric *Verification Ratio*, which is simply the number of verifications done per candidate in the `CandidateSet`. If this number is equal to 1, that means we have performed a very large number of verifications. The closer it is to 0, the fewer the number of verifications that were done. This ratio also enables us to compare the differences across dataset, where the number of candidates vary. As it can be seen in figure 5.5, the verification ratio never exceeds 19% (the maximum value being 18.3%).

In order to compare these results, we perform a tagging experiment using location information alone. Candidates are ordered according to their last known location, and presented to the user in the same style as the information presented with the context discovery algorithm. In this case, it must be noted that some users do not have location information related to them, and are excluded from the candidate set. The effect of this fact is seen in the reduced hit count seen using location information.

Similarly, we measure the verification ratio using location only. The results in figure 5.7

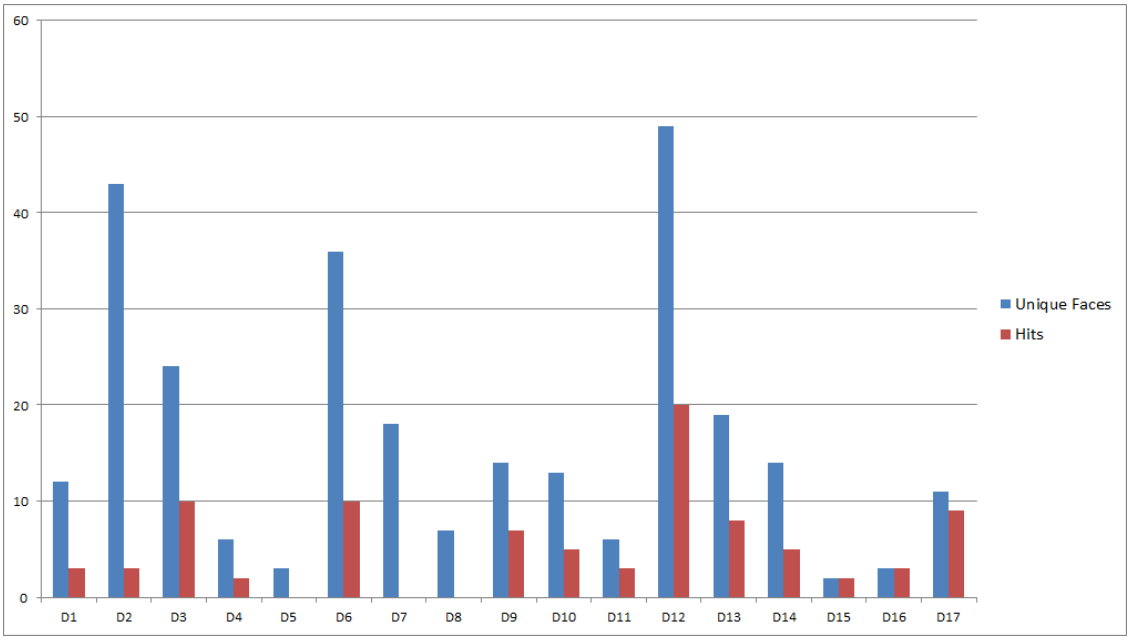


Figure 5.6: Hit counts for all datasets using Location only.

show that the maximum number has now increased to almost 29%.

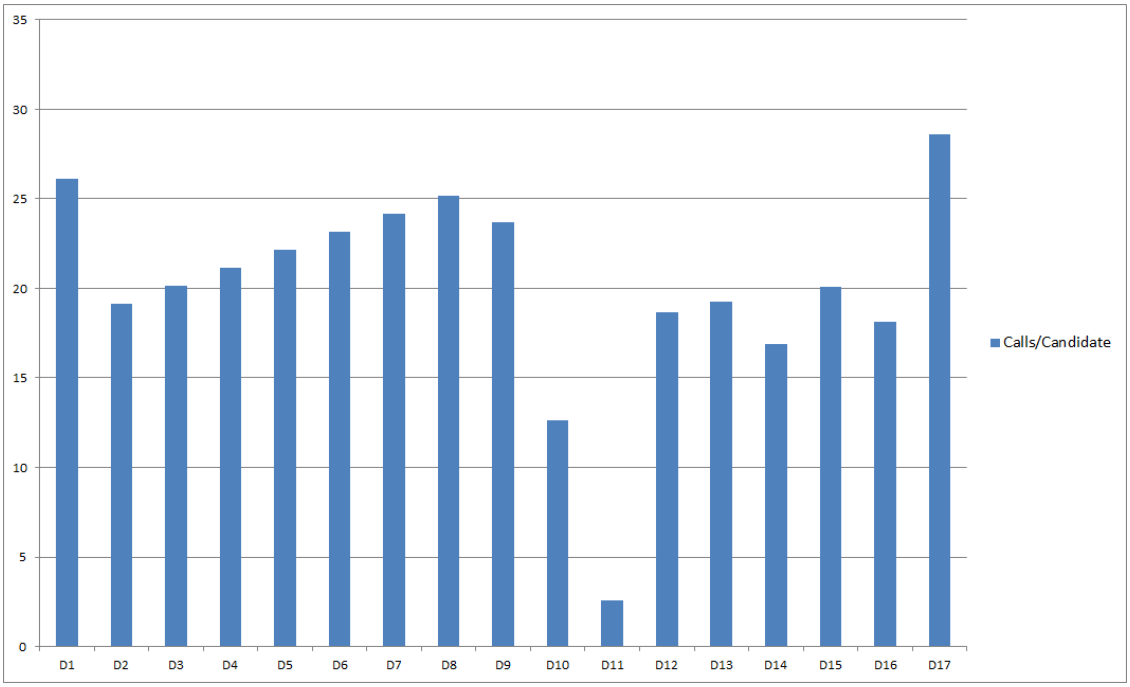


Figure 5.7: Verification Ratio (x 100) for all datasets obtained using Location only.

In figure 5.8, we compare the **hits ratio** of the two algorithms. The hit ratio is the number of hits per unique face in a dataset. As it can be seen, the context discovery algorithm

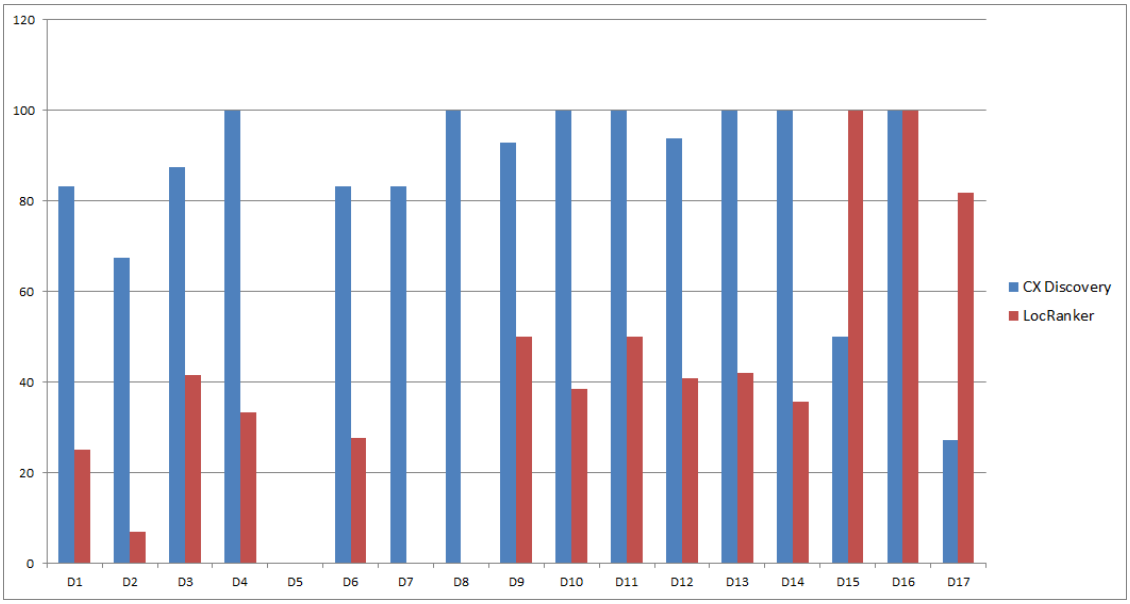


Figure 5.8: Comparing Hits Ratio (x 100) for all datasets.

outperforms the simple location based ranking algorithm. The more important insight in this graph is that the context discovery algorithm performs **consistently across different types of events**, whereas the location based metric is good only for the party events. Figure 5.9 compares the different verification ratios. The relatively lower numbers for the context discovery algorithm indicate its superior ranking over the location based algorithm. Again, the important thing is to note its **performance across event types**.

5.1.4 Conclusion

We summarize the results of this experiment as follows. The context discovery algorithm performs consistently well across different types of events. Depending on the type of the event, it discovers the best data sources and ranks the candidate set in the appropriate way. For a single dataset, the ranking of candidates can significantly vary depending on who has been tagged in the photo. We have not implemented any cross-photo context propagation into our algorithm, but this is an active area of research, and there are many possible ways of introducing that technique into our context discovery algorithm.

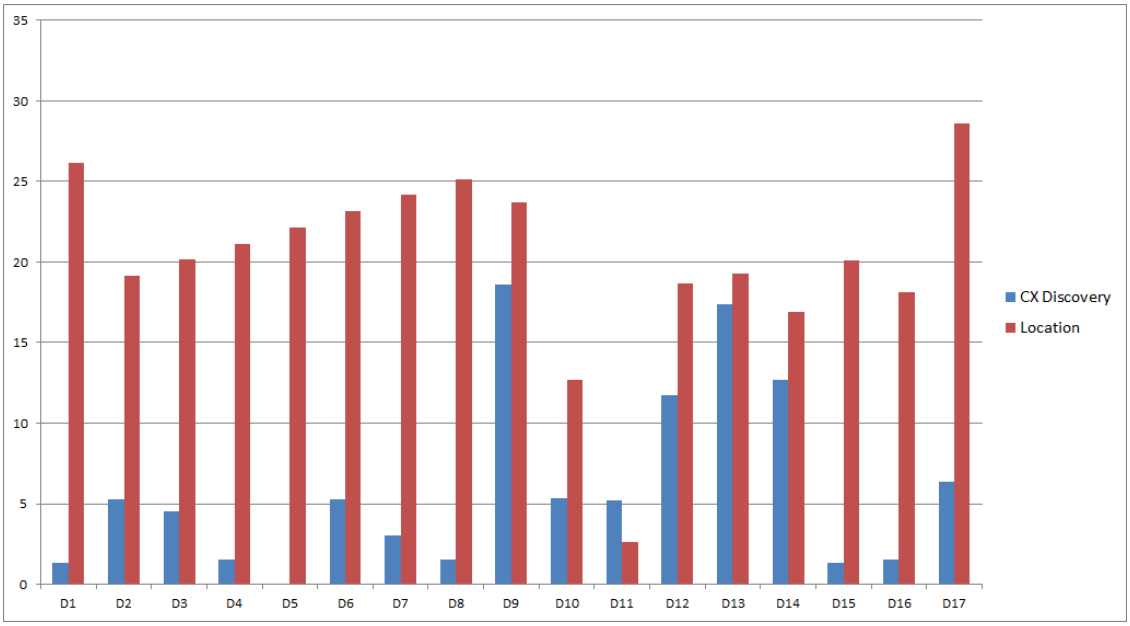


Figure 5.9: Comparing Verification Ratio (x 100) for all datasets.

5.2 Performance Analysis

In this section, we will look at the performance characteristics of the CueNet algorithm. More specifically, the merge function in the discovery algorithm. In order to achieve this, we will present a generative model to generate large workloads of context networks, which will be merged with each other. Using the generative model, networks of varying sizes will be created so that the time and space complexity of the algorithm can be studied.

5.2.1 Generative Models

The generative model requires three main components. A set of places where events occur. An ontology which determines what events occur, and how certain events are more likely to occur as sub-events of already occurring events and the event instances themselves. We will describe each of the components in turn.

A large number of road network databases are available online. Instead of designing a

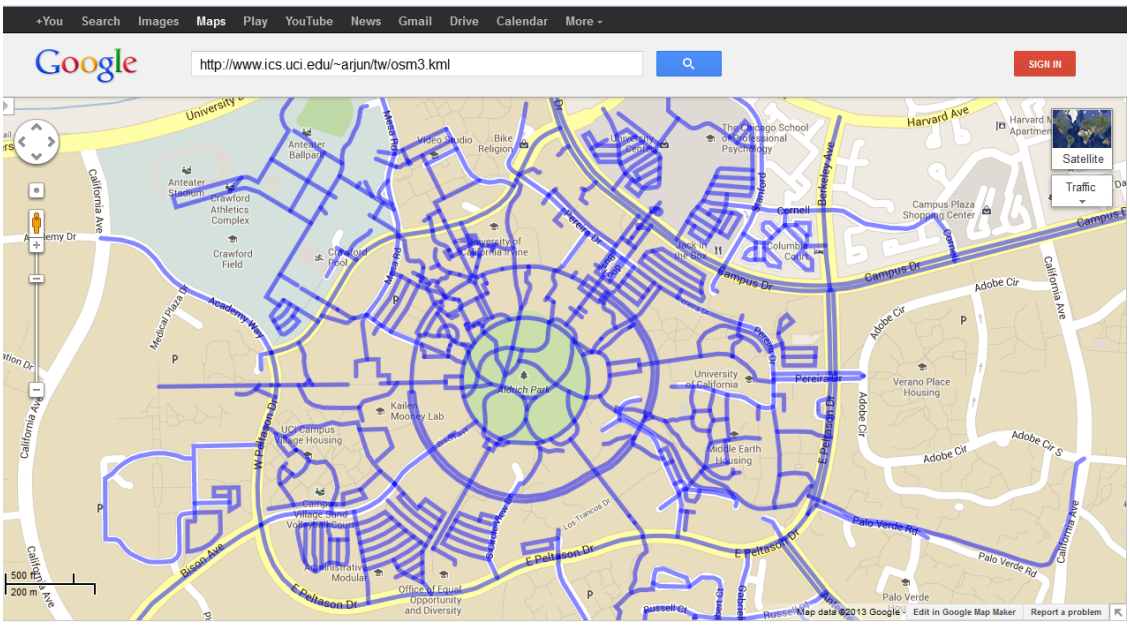


Figure 5.10: Roadmap containing places where events could occur.

generative model for places, it is advisable to use one of these real world databases and select a sample of it to obtain a list of places where events can occur. Some commonly used databases are shapefiles [5] for road network information about a particular area. Stanford's SNAP database contains the entire road networks of the USA [2]. The FSU dataset available at [4] provides detailed road networks with GPS coordinates from various parts of the country. For our work, we found the spatial database available from OpenStreetMap as the best resource. Their database provides information about road networks, as well as the places themselves. At the same time, road maps can be downloaded from their website using the download tool or maps for entire cities or states or countries can be downloaded from [1]. The road map of UCI downloaded from their website and converted to KML format is shown in figure 5.10. The map data also contains information about different types of objects which can be found on these roads. Please look at their extensive taxonomy [3] regarding what types of objects can be found here. We select a sample of the nodes in such a network to be our place set.

The ontology generation module provides three parameters. O_d, which controls the depth of subevent trees. O_{ms} which is the maximum number of subevents an event can have. D,

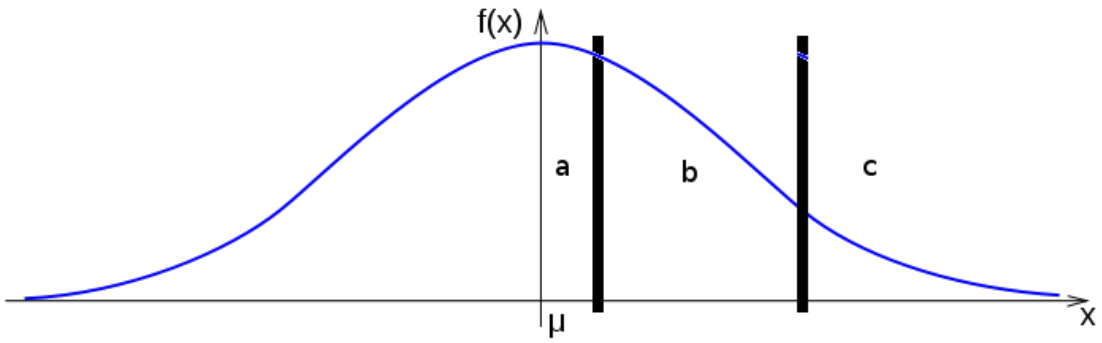


Figure 5.11: Gaussian distribution split to generate randon events in ontology.

which is the distribution type. The distribution type decides which events are selected into a hierarchy. A distribution could be Gaussian or Poisson or Uniform distribution. Once the distribution is selected, parameters of the distribution are also provided. We will use a Gaussian distribution as an example to explain the generation of ontologies. But the ideas can be applied to any kind of distribution.

We want to model events into three categories depending on their generate rate of occurrence. Some events are very commonly occurring, a majority of events occur average number of times, and events in the third category occur very rarely. Figure 5.11 shows how a gaussian random value generator can be used to bucket events into these three zones. The random values are compared with the values a, b and c. If the $|value|$ is between 0 and a, then the event falls into the first category. If it lies between a and b, then it lies in the second category. Otherwise it is in the third category.

We consider events in the first category to be atomic events. That is, they occur independently and do not have a subevent hierarchy. An example could be a tweet event or a `photo-capture` event. We want to create two types of hierarchies. One which is short and fat, and the other which is long and tall. We choose category two to correspond to the former, and category three to contain the latter type of hierarchies. A hierarchy is generated as follows: choose an event, if it is from category 2 or 3, a hierarchy with atmost depth O_d and each super event having max O_{ms} subevents is generated. The events are randomly

chosen based on their "real-world" frequency. These hierarchies are serialized into files. For events in category 3, we double the value of O_{ms}. This will allow us to get *fatter* instance hierarchies as we will see later.

In order to generate instances, we randomly select an event based on its "real-world" frequency, and load the hierarchy structure from the file, and for subevent class, generate n instances. Thus, if an event x has a subevent y in its hierarchy, then the instance x_i will have subevent instances $\{y_1, y_2...y_{|Y|}\}$ where $|Y| < n$. The number n is decided on the basis of an input metric C, where for events in category 2, $n = c/2$, and for events in category 3, $n = C$. It must be noted that the n is the maximum number of subevents a super event can contain. The actual number of subevent instances is decided by a uniform random number generator. These hierarchies are serialized into files and stored for the merge tests.

5.2.2 Procedure

In order to understand the time and space complexity of the merge function, we load multiple networks from instance hierarchy into in-memory data structures, and perform a sequence of merges. In order to simulate the workload similar to the discovery algorithm, we select a network, and perform a sequence of merges on it. We will refer to the initially selected network as a **primary** network and the networks which are merged into it as **secondary** networks. Thus, given a primary network and a set of secondary networks, we perform the following set of experiments.

First, we fix the size of the primary networks (in terms of node count), and merge secondary networks with varying size, and measure the time taken for the merges to complete.

Second, we fix the size of the primary networks (in terms of depth), and merge secondary networks with varying depth, and measure the time taken for the merges to complete.

Third, we merge a context network with an exact replica of itself, and measure the time taken for the operation. We also change the size of the heap size, and see if there is any effect on the merge operation.

Fourth, we measure the sizes of the networks once they are in memory.

It must be noted that the event instances being used in the experiments do not have event metadata associated with the in-memory instance object. It is assumed that these are being stored separately. The networks used below are simply events with type, spatio-temporal and identification information and references to their subevents and to entities who participate in the events. Any additional information, for example the image object which constitutes the experiential aspect of an event, is stored separately.

We use the following system setup in all our experiments. We use a 8 core Intel(R) Xeon(R) E5504 processor clocking at 2.01Ghz with 8 gigabytes of main memory. We use a single thread to load networks from the file on the disk and proceeds to merge them. The memory footprints of the networks are measured by the classmexer library [31]. The `Xmx` and `Xms` JVM parameters were set to be 6000m and 256m respectively.

5.2.3 Results

The graph shown in figure 5.12 shows the time taken for merging secondary networks of increasing sizes to a primary network. The size of the primary network is increased from 10 nodes to a million nodes. For each such primary network, a number of secondary networks are merged. As we can see, the size of the primary network does not contribute to the time complexity of the algorithm. This is because the merge algorithm effectively searches the subevent structure to find the node which can best accommodate the secondary network. This is a walk down a single path of the tree. Thus, as long as small secondary networks

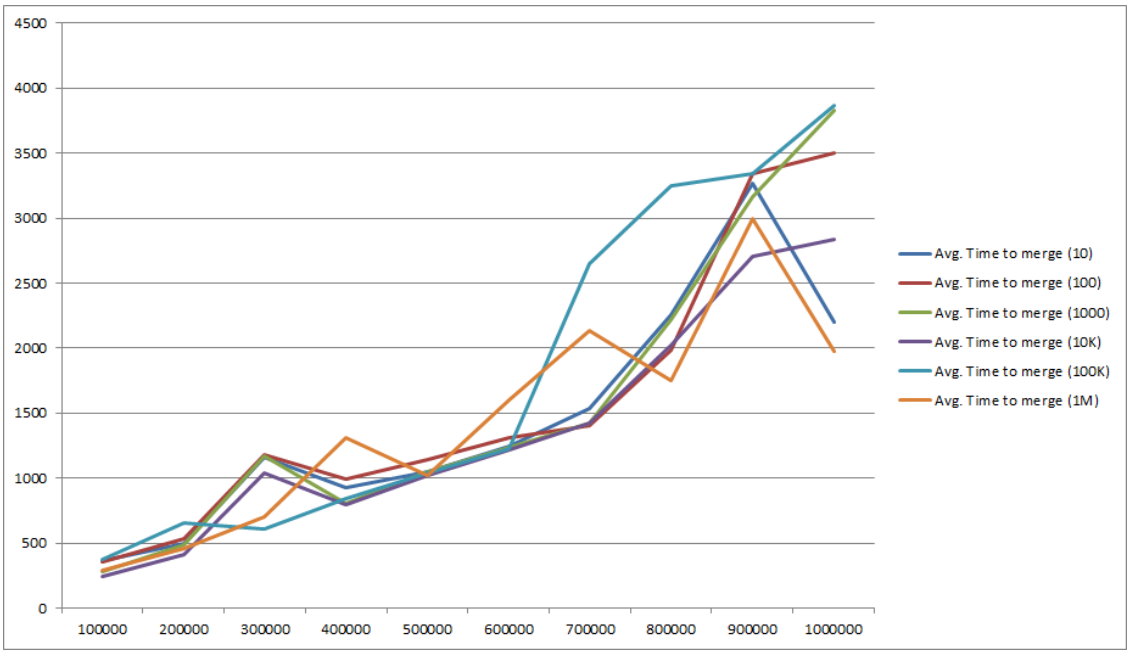


Figure 5.12: Graph showing merge times for different sizes of primary networks.

are obtained from the sources, the merge operations will be less expensive. This is the reason why we tailor our discover queries to provide only very specific information. On the other hand, as the size of the secondary network increases, we find that the performance increases linearly upto a point, after it which it exponentially increases. This is due to a combination of tree traversal on the secondary network, and the JVM's garbage collector removing outdated objects to accommodate the new objects being created in the primary network. In our implementation, we create new objects in the primary network, as opposed to copying them from the secondary network.

Similarly, in figure 5.13, we test how merge time varies with increasing depth of the primary network. Here we fix the depth of the primary network to 3, 5 or 7 levels, and load a set of secondary networks from depth 1 to 9. The secondary networks are merged in sequence to the primary network. Each run of the algorithm corresponds to a single primary network (effectively, we reboot the JVM for each primary network). Here the exponential rise after the depth of the secondary network reaches 8 levels is more obvious. This again, shows the importance of constructing selective queries during the discover phase of the algorithm. The

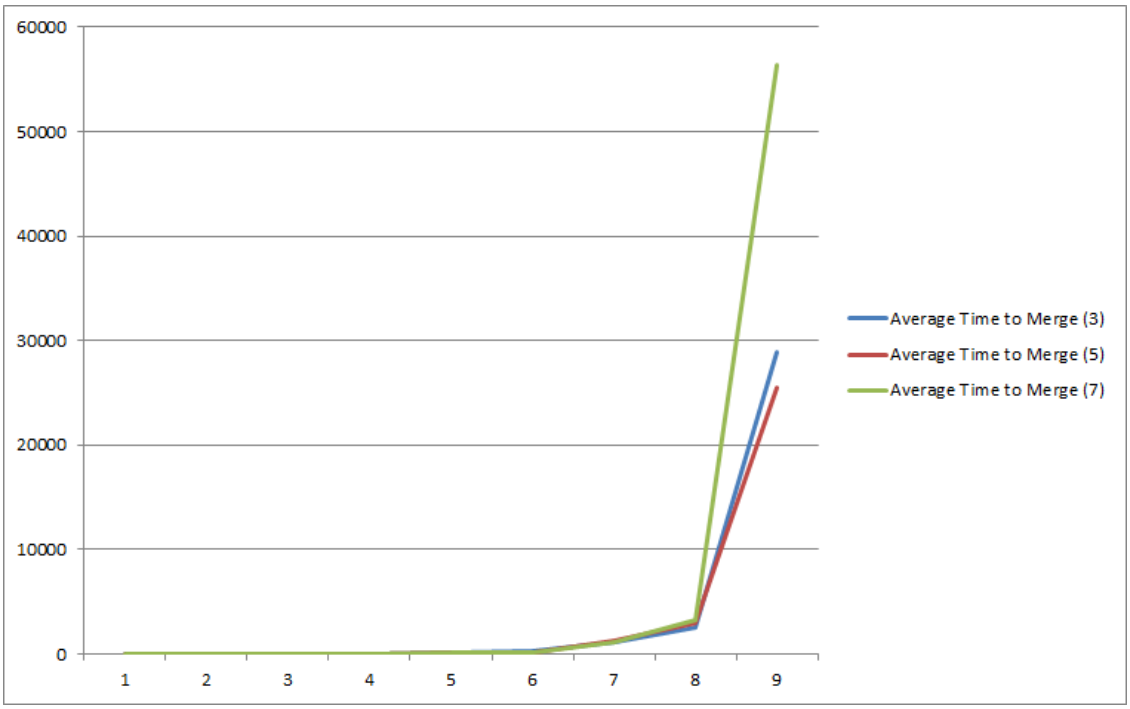


Figure 5.13: Time to merge networks with varying depth.

loading of all the secondary networks prior to the merge sequence allows us to study the cost of merges in an environment where memory is not available, and is shared by other citizen objects of the JVM heap.

Figures 5.14 and 5.15 show the time complexity of self merges. A self merge is merging a network with an identical copy of itself. The perforance characteristics are very similar to the merge tests seen before.

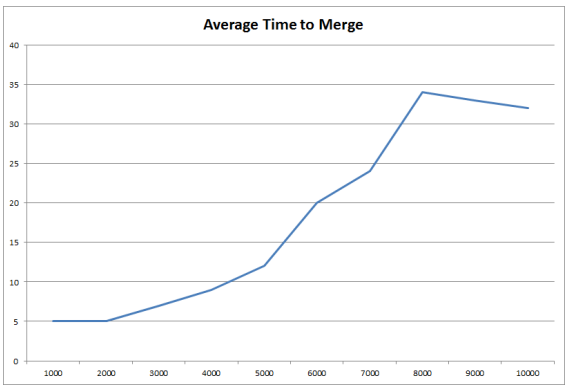


Figure 5.14: 1K-10K Nodes Self Merge

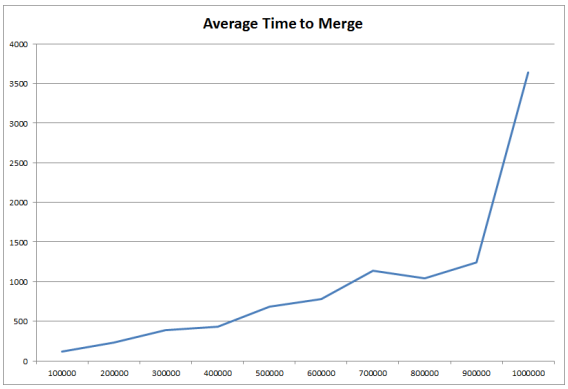


Figure 5.15: 100K-1M Nodes Self Merge

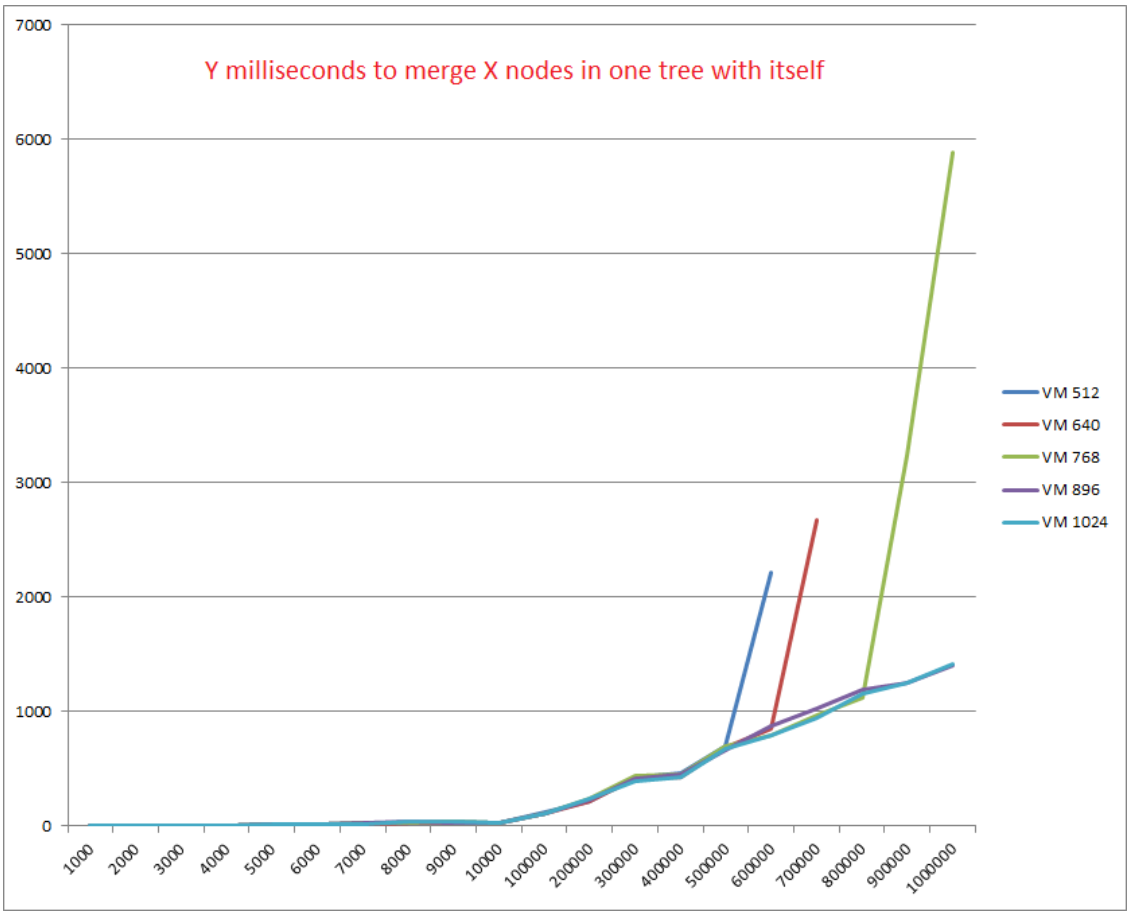


Figure 5.16: Time to merge networks with varying available heap space.

Now, we will look at how the merges behave with varying heap size (the `Xmx` runtime parameter provided to the JVM when it is initialized). In figure 5.16, we see 5 different curves, corresponding to `Xmx` values of 512m, 640m, 768m, 896m, and 1024m. We load a network twice, perform a merge (a *self-merge*), and record the time the operation takes. The experiments is repeated 5 times for each merge on different randomly created networks. We perform these operations on networks containing 1000, 2000, 3000 nodes upto 700K, 800K, 900K and 1M nodes. As we can see in the figure, we run out of heap space when the networks contain 600K and the Xmx value is 512m. Beyond a 896m, we see that the merge operations only grow linearly, and can be considered a safe limit when self-merges of 1M node networks are to be performed. For heap spaces of less than 896m, the performance time increases exponentially when the network size exceeds 600K nodes. This is, again, because of the

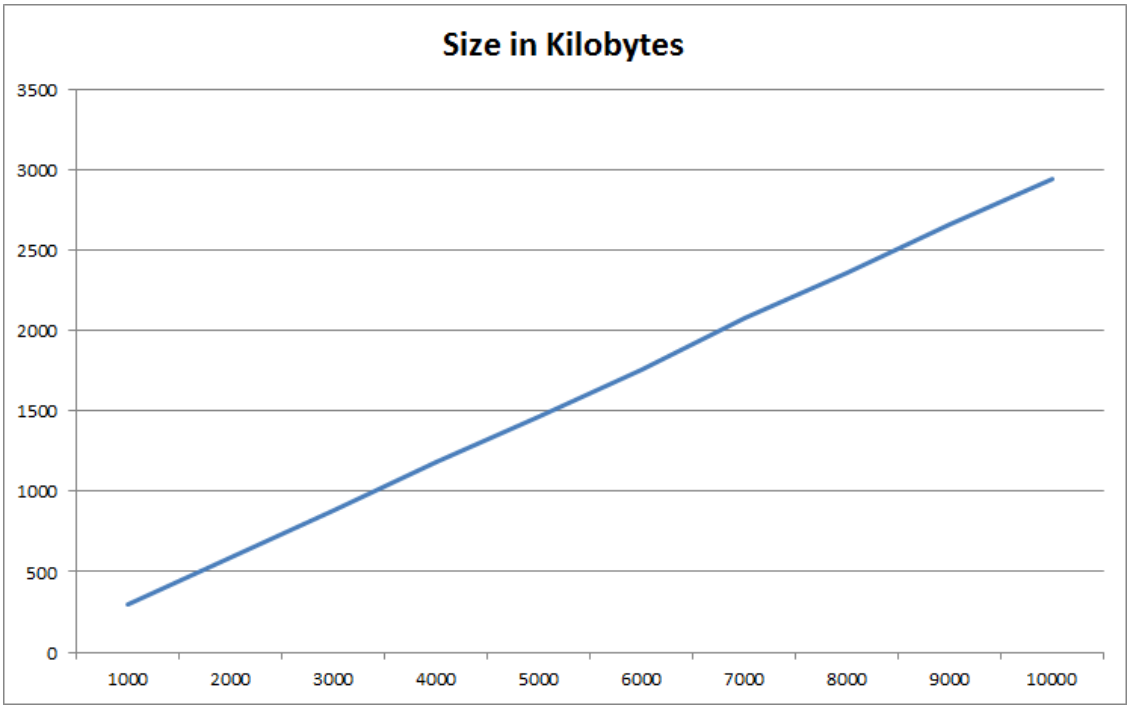


Figure 5.17: Sizes of in-memory context networks contain a few thousand nodes.

creation of nodes in the primary networks.

Finally, we use the Classmexer library [31] to measure the memory footprint of the in-memory context network. We generate random networks of size 1000, 2000 ... 10,000 nodes, and measure the size in kilobytes. The results are shown in figure 5.17.

5.2.4 Conclusion

In analysing the above figures, we have seen that the in-memory merge algorithm performs very well for networks upto 500K nodes. Beyond this point, the performance exponentially decays, both as a combination of large graphs, and the JVM's garbage collector trying to clean up unwanted objects. And therefore it is best recommended to approach an disk based solution or partition the context networks across different machines.

Since performing many merges (especially with large secondary networks), it is advisable to

not perform the merges when the rate of incoming networks is very high. Rather, the system can cache the networks, and utilize techniques like iterative dynamic programming or *batch* the merges such that the overall load on the CPU is reduced. Such an algorithm, to the best of my knowledge, does not readily exist, and can be a very fruitful future research to improve the utility of context networks in various domains.

Chapter 6

Ranking Context Networks

So far, our treatment of context networks had made an assumption: context solely consists of events co-occurring with the photo-capture events, and these events are obtained from data sources. The second part of the assumption is not always true in practice. For example, a person's roommate might be a last minute addition to a road trip, who was neither on the email chain or the facebook event, will be ranked very low by the discovery algorithm. A professor whose students are receiving best paper award but was not part of the author list herself might be present at the award ceremony because the conference is being hosted 50 kilometers from her univeristy. Similarly, people at a concert might run into acquaintances because they share the same musical interests. In these cases, the data sources will provide incomplete contextual information which leads the discovery algorithm to rank some candidates poorly.

In this chapter, we will attempt to address this problem of boosting the performance of the CueNet framework by introducing a technique to rank candidates based on their participation profile in previous context networks, personal interests or information. We will present our intuition through a series of examples, and introduce a technique similar to PageRank,

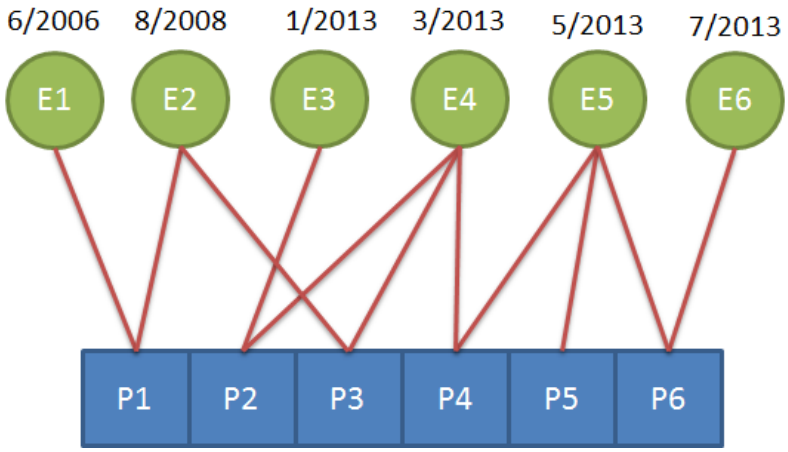


Figure 6.1: Propagating values through temporal relations.

which is used to rank pages on the world wide web. In our case, we will rank people in the `CandidateSet` given set of context networks. Our main differences will lie in the initialization of the score matrix and the propagation of scores *across* the different context networks.

6.1 Intuition

Consider the very simple context networks shown in figure 6.1. Each network contains an event, and they have a set of participants. Lets assume that all events `E1` - `E6` are of the same type, and occurred at the same location. But as shown in the figure the time of occurrence varies. Lets say that the hit ratio for tagging `E1` - `E5` is 100%. But `E6` has a few untagged faces, and no more data source is able to provide additional context. The context discovery algorithm has a basic ranking algorithm which looks all social networks (temporal or not), and ranks entities based on whether they have any relations with its participants (in this case, `P6`). In this case, we could find that `P6` has participated with `P5` and `P4` in event `E5`. The voting algorithm stops if `P5` and `P4` are not found in the `E6`'s photo. But looking at all the networks, it makes sense to say that since these are all events of the same type, we could say that `P2`, `P3` and `P4` have participated in this same event a few months before `E6`'s

occurrence, and therefore it is possible that P2 and P3 also participating in E6. With the same idea, we can also P2 and P1 participated in the same event many years, and therefore is a very low but non-zero chance that P1 is present in the photo.

Similarly, look at figure 6.2 where the context networks are exactly as before, but the types of events now vary. We show the type of the event above the event nodes. Their timestamps and location properties are exactly as before. Here, can we exploit some ontological properties about event similarities to propagate scores. If we know that L1 is a conference, L2 is a music-event and L3 is a party, then can we say that L2 events are more likely to contain the untagged person than the conference events.

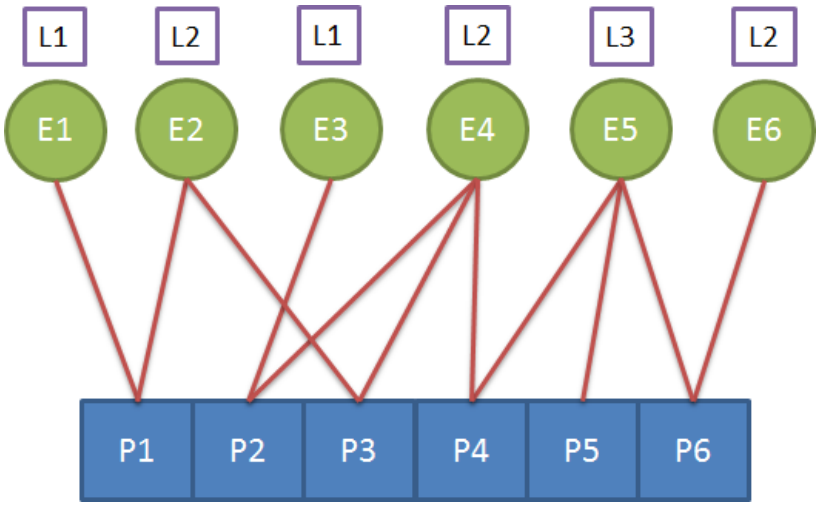


Figure 6.2: Propagating values through ontological event relations.

Similarly, we can use social and spatial relationships to propagate rank scores to more individuals than just making a single hop as in the context discovery algorithm to reach a wider set of candidates.

6.1.1 Rank Propagation

The rank propagation technique is used to rank nodes in a directed graph. Most commonly seen versions are the HITS algorithm invented by Jon Kleinberg [62] and the PageRank

algorithm developed by Larry Page and Sergey Brin [79]. The idea in the latter is to assign a set of initial values to a subset of nodes, and propagate a fraction of these values to their neighbours. Each iteration of the algorithm propagates their scores to the neighbouring nodes until the overall ordering of the nodes, according to their scores, does not change. In practice, a few iterations (≤ 10) on web scale graphs is sufficient.

In this chapter, we will modify the rank propagation technique to rank entities on a given set of context networks, to determine who might be participating in a new events. We will initialize scores of the nodes on the basis of what is known about the new event – its type, the participating entities, and propagate the scores based on a set of propagation functions which we will introduce later. Since the basic propagation algorithm is an expensive one, we will look at some ways to reduce the size of the context networks, so the algorithm converges quickly, and still provides useful results.

6.2 Preliminaries

6.3 Rank Propagation in Context Networks

6.3.1 Propagation Functions

6.4 Experiments

Chapter 7

Future Work and Conclusion

7.1 Known Issues

In this section, we list some of the issues encountered in designing and building CueNet. Some of these are active areas of research, and whereas others are specific to our framework, and can be considered potential areas of research. Our experience with CueNet indicates that the following issues should be approached in a holistic manner, i.e., in conjunction with each other. Approaching these problems in the context of each other reduces the individual complexity of each sub-problem by possibly increasing the complexity of the entire framework, but making the problem more tractable.

Noise in Social Media: The problem of noise filtering in web data is a prominent one, and is being addressed by various communities in different ways. These range from entity matching and record linkage problems [41] to correcting missing data in information networks [88]. These problems get trickier because of the different variations in representing tiny details such as representation names of people, addresses of places, and time. In fact, there is a whole school of anthroponomastics [92] dedicated to studying variations in human names. Ideas in

this field indicate that these differences arise due to cultural, historical and environmental issues[8]. Such issues cannot be trivially addressed.

Face tagging in social media sources like Facebook can also be very noisy. This strictly prohibits directly using this data to train verification/recognition models. Also, the quality of photos are poor, resulting in weaker features, which would have otherwise allowed better matching.

An exhaustive scientific characterization of noise in social media is beyond the scope of this paper, and is being investigated step by step in social media research communities.

CPU Efficiency: The query engine in CueNet is responsible for extracting data from different sources. If a very large number of photos are being tagged, our scheme of query generation and merging will prove inefficient. Processing many photos from different people provides a very rich opportunity to develop interesting heuristics using event semantics for the multi-query optimization problem. Also, partitioning the discovery algorithm such that the computations can occur in a distributed manner is a complex problem. Such steps will be required if the application workload is of the scales of Facebook or processing photos in real time at the scales of Instagram.

Face Verification: Even though face recognition has been studied in research for the last two decades, face verification, and its specific application to faces in the wild has been a relatively recent venture. Although the accuracy of these systems is commendable, the problems of occlusion, image quality, face alignment and differing lighting conditions exist. These hard problems need to be solved before "perfect" or "near-perfect" verification can be established.

Execution Patterns: When is a good time to execute the algorithm? When a user takes a photo? Or before she uploads it to her favorite photo sharing site? For the current evaluation, contextual sources are assumed to be immutable. This is not true in the real

world. Contextual sources are constantly being appended with new information, and old information is being updated. These updates may be vital in tagging a certain photo. So the question of when to execute the algorithm, or how and when to query the sources is an open question. If a large number of photos are to be tagged, and a busy source like Facebook is being used for context, the CueNet query engine must take into account various freshness metrics and crawling policies of the sources.

Open Datasets: The unavailability of a large public data set over which different techniques can be evaluated against each other is an open problem. As seen in our experiments, personal information is vital to contextual approaches, and this data is largely personal, and therefore cannot be shared openly. Optimal anonymization techniques need be to invented such that the privacy of the experiment participants are maintained, and at the same time the data is meaningful to be applied in contextual approaches to problems. This need to be solved so that new context discovery techniques can be evaluated independently and against each other, over a common platform.

Problems/Opportunities:

1. Absence of a language/tool which can be used to express real world models. Ontologies are logic driven. Usually deal with true/false.

2. Starting points of such a language – physical variables, graphics world modeling, entity identification.

3. Backtracking based algorithms.

Bibliography

[1] http://downloads.cloudmade.com.

[2] http://snap.stanford.edu/data.

[3] http://wiki.openstreetmap.org/wiki/tags.

[4] http://www.cs.fsu.edu/ lifeifei/spatialdataset.htm.

[5] http://www.esri.com/data/download/census2000-tigerline.

[6] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12), 2006.

[7] J. P. Ahrens, B. Hendrickson, G. Long, S. Miller, R. Ross, and D. Williams. Data-intensive science in the us doe: Case studies and future challenges. *Computing in Science & Engineering*, 13(6):14–24, 2011.

[8] A. W. Q. G. Al-Zumor. A socio-cultural and linguistic analysis of yemeni arabic personal names. *GEMA: Online Journal of Language Studies*, 9(2):15–27, 2009.

[9] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[10] M. Ames and M. Naaman. Why we tag: motivations for annotation in mobile and online media. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2007.

[11] V. Astakhov, A. Gupta, S. Santini, and J. S. Grethe. Data integration in the biomedical informatics research network (birn). In *Data Integration in the Life Sciences*, pages 317–320. Springer, 2005.

[12] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54. ACM, 2006.

[13] P. Barthelmess, E. Kaiser, and D. McGee. Toward content-aware multimodal tagging of personal photo collections. In *Proceedings of the 9th international conference on Multimodal interfaces*. ACM, 2007.

[14] M. Bazire and P. Brézillon. Understanding context before using it. In *Modeling and using context*, pages 29–40. Springer, 2005.

[15] P. Belhumeur, J. Hespanha, and D. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7), 1997.

[16] C. G. Bell, J. Gemmell, and C. Rosson. *Total recall*. Dutton, 2010.

[17] T. Berg and P. N. Belhumeur. Tom-vs-pete classifiers and identity-preserving alignment for face verification. In *BMVC*, volume 1, page 5, 2012.

[18] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *Database TheoryICDT99*, pages 217–235. Springer, 1999.

[19] M. Bicego, A. Lagorio, E. Grosso, and M. Tistarelli. On the use of sift features for face authentication. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pages 35–35. IEEE, 2006.

[20] S. Boll, P. Sandhaus, A. Scherp, and U. Westermann. Semantics, content, and structure of many for the creation of personal photo albums. In *Proceedings of the 15th international conference on Multimedia*, pages 641–650. ACM, 2007.

[21] M. Boutell and J. Luo. Photo classification by integrating image content and camera metadata. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 4. IEEE, 2004.

[22] M. Boutell and J. Luo. Beyond pixels: Exploiting camera metadata for photo classification. *Pattern recognition*, 38(6), 2005.

[23] E. Brachmann, M. Spehr, and S. Gumhold. Feature propagation on image webs for enhanced image retrieval. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 25–32. ACM, 2013.

[24] P. Brézillon. Context-based modeling of operators practices by contextual graphs. In *Proceedings of the 14th Conference on Human Centered Processes*, pages 129–137, 2003.

[25] L. Cao, J. Luo, and T. Huang. Annotating photo collections by label propagation according to multiple similarity cues. In *Proceeding of the 16th ACM international conference on Multimedia*. ACM, 2008.

[26] L. Cao, J. Luo, H. Kautz, and T. Huang. Annotating collections of photos using hierarchical event and scene models. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008.

[27] F. Capra. *The web of life: A new scientific understanding of living systems*. Anchor, 1997.

[28] G. Chen and D. Kotz. A survey of context-aware mobile computing research. 2000.

[29] Z. Chen, L. Wenyin, F. Zhang, M. Li, and H. Zhang. Web mining for web image retrieval. *Journal of the American Society for Information Science and Technology*, 52(10):831–839, 2001.

[30] M. Chlistalla, B. Speyer, S. Kaiser, and T. Mayer. High-frequency trading. *Deutsche Bank Research*, 2011.

[31] N. Coffey. http://www.javamex.com/classmexer/.

[32] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, CVPR 2005. IEEE Computer Society Conference on*, volume 1. IEEE.

[33] R. Datta, D. Joshi, J. Li, and J. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys (CSUR)*, 40(2), 2008.

[34] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[35] N. Diakopoulos and P. Chiu. Photoplay: A collocated collaborative photo tagging game on a horizontal display. *UIST Adjunct Proceedings*, 2007.

[36] J. M. Diamond and D. Ordunio. *Guns, germs, and steel*. Norton New York, 1997.

[37] A. Doan and A. Y. Halevy. Semantic integration research in the database community: A brief survey. *AI magazine*, 26(1):83, 2005.

[38] D. Dou, D. McDermott, and P. Qi. Ontology translation on the semantic web. In *Journal on data semantics II*, pages 35–57. Springer, 2005.

[39] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *Proceedings of the 26th ACM SIGIR conference on Research and development in Informaion Retrieval*. ACM, 2003.

[40] A. Dumitrescu and S. Santini. Context based semantics for multimodal retrieval. In *IS&T/SPIE Electronic Imaging*, pages 72550C–72550C. International Society for Optics and Photonics, 2009.

[41] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1), 2007.

[42] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[43] C. Frankel, M. J. Swain, and V. Athitsos. Webseer: An image search engine for the world wide web. 1996.

[44] D. Frohlich, A. Kuchinsky, C. Pering, A. Don, and S. Ariss. Requirements for photoware. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. ACM, 2002.

[45] C. Galleguillos and S. Belongie. Context based object categorization: A critical survey. *Computer Vision and Image Understanding*, 114(6):712–722, 2010.

[46] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. Mylifebits: fulfilling the memex vision. In *Proceedings of the tenth ACM international conference on Multimedia*. ACM, 2002.

[47] C. Geng and X. Jiang. Sift features for face recognition. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 598–602. IEEE, 2009.

[48] A. Graham, H. Garcia-Molina, A. Paepcke, and T. Winograd. Time as essence for photo browsing through personal digital libraries. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2002.

[49] A. Gupta and R. Jain. Managing event information: Modeling, retrieval, and applications. *Synthesis Lectures on Data Management*, 3(4), 2011.

[50] J. Hailpern, N. Jitkoff, A. Warr, K. Karahalios, R. Sesek, and N. Shkrob. Youpivot: improving recall with contextual search. In *Proceedings of the 2011 annual conference on Human factors in computing systems*. ACM, 2011.

[51] A. Y. Halevy. Answering queries using views: A survey. *The VLDB JournalThe International Journal on Very Large Data Bases*, 10(4):270–294, 2001.

[52] J. Hays and A. Efros. Im2gps: estimating geographic information from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. Ieee, 2008.

[53] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive Computing*, pages 167–180. Springer, 2002.

[54] D. Henter, D. Borth, and A. Ulges. Tag suggestion on youtube by personalizing content-based auto-annotation. In *Proceedings of the ACM multimedia 2012 workshop on Crowdsourcing for multimedia*, pages 41–46. ACM, 2012.

[55] J. R. Hobbs and F. Pan. Time ontology in owl. w3c working draft, 27 september 2006. *World Wide Web Consortium. http://www. w3. org/TR/owl-time*, 2006.

[56] J.-y. Hong, E.-h. Suh, and S.-J. Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4):8509–8522, 2009.

[57] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. 2007.

[58] G. B. Huang, M. Mattar, T. Berg, E. Learned-Miller, et al. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on Faces in'Real-Life'Images: Detection, Alignment, and Recognition*, 2008.

[59] R. Jain and P. Sinha. Content without context is meaningless. In *Proceedings of the international conference on Multimedia*. ACM, 2010.

[60] O. Jesorsky, K. J. Kirchberg, and R. W. Frischholz. Robust face detection using the hausdorff distance. In *Audio-and video-based biometric person authentication*, pages 90–95. Springer, 2001.

[61] T. Kindberg, M. Spasojevic, R. Fleck, and A. Sellen. The ubiquitous camera: An in-depth study of camera phone use. *Pervasive Computing, IEEE*, 4(2):42–50, 2005.

[62] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[63] M. J. Krawczyk. Communities in social networks. In *Biometrics and Kansei Engineering, 2009. ICBAKE 2009. International Conference on*, pages 111–116. IEEE, 2009.

[64] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Describable visual attributes for face verification and image search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Special Issue on Real-World Face Recognition*, Oct 2011.

[65] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Describable visual attributes for face verification and image search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, October 2011.

[66] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[67] S. D. Levitt and S. J. Dubner. *Freakonomics Rev Ed LP: A Rogue Economist Explores the Hidden Side of Everything*. HarperCollins, 2006.

[68] X. Li, M. Larson, and A. Hanjalic. Geo-visual ranking for location prediction of social images. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 81–88. ACM, 2013.

[69] X. Li, C. G. Snoek, M. Worring, and A. W. Smeulders. Fusing concept detection and geo context for visual search. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, page 4. ACM, 2012.

[70] X. Liu and B. Huet. Heterogeneous features and model selection for event-based media classification. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 151–158. ACM, 2013.

[71] J. Luo, Y. Ma, E. Takikawa, S. Lao, M. Kawade, and B.-L. Lu. Person-specific sift features for face recognition. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 2, pages II–593. IEEE, 2007.

[72] F. Monaghan and D. O'Sullivan. Automating photo annotation using services and ontologies. In *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*. IEEE, 2006.

[73] G. K. Mostefaoui, J. Pasquier-Rocha, and P. Brezillon. Context-aware computing: a guide for the pervasive computing community. In *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on*, pages 39–48. IEEE, 2004.

[74] M. Naaman. *Leveraging geo-referenced digital photographs*. PhD thesis, Citeseer, 2005.

[75] M. Naaman, R. Yeh, H. Garcia-Molina, and A. Paepcke. Leveraging context to resolve identity in photo albums. In *Digital Libraries, 2005. JCDL'05. Proceedings of the 5th ACM/IEEE-CS Joint Conference on*. IEEE, 2005.

[76] B. Nowack. Confoto: Browsing and annotating conference photos on the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(4), 2006.

[77] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record*, 33(4):65–70, 2004.

[78] N. O'Hare and A. Smeaton. Context-aware person identification in personal photo collections. *Multimedia, IEEE Transactions on*, 11(2), 2009.

[79] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[80] Y. Pang, Y. Yuan, X. Li, and J. Pan. Efficient hog human detection. *Signal Processing*, 91(4):773–781, 2011.

[81] D. J. Patterson. *Assisted cognition: compensatory activity assistance technology*. PhD thesis, 2005.

[82] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek. Overview of the face recognition grand challenge. In *Computer vision and pattern recognition, 2005. CVPR 2005. IEEE computer society conference on*, volume 1, pages 947–954. IEEE, 2005.

[83] P. J. Phillips, H. Wechsler, J. Huang, and P. J. Rauss. The feret database and evaluation procedure for face-recognition algorithms. *Image and vision computing*, 16(5):295–306, 1998.

[84] R. Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[85] E. PrudHommeaux, A. Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.

[86] T. Rattenbury and M. Naaman. Methods for extracting place semantics from flickr tags. *ACM Transactions on the Web (TWEB)*, 3(1), 2009.

[87] P. Reignier, O. Brdiczka, D. Vaufreydaz, J. L. Crowley, and J. Maisonnasse. Context-aware environments: from specification to implementation. *Expert systems*, 24(5):305–320, 2007.

[88] E. Sadikov, M. Medina, J. Leskovec, and H. Garcia-Molina. Correcting for missing data in information cascades. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 55–64. ACM, 2011.

[89] N. Sambasivan, E. Cutrell, K. Toyama, and B. Nardi. Intermediated technology use in developing communities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2583–2592. ACM, 2010.

[90] N. Sawant, J. Li, and J. Wang. Automatic image semantic interpretation using social action and tagging data. *Multimedia Tools and Applications*, 2011.

[91] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.

[92] M. Schneider. What's in my name? toward a generative anthroponomastics. *Anthropoetics*, 15(1):1–9, 2009.

[93] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, and F. L. Wong. Sensay: A context-aware mobile phone. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, volume 248, 2003.

[94] P. Sinha and R. Jain. Concept annotation and search space decrement of digital photos using optical context information. In *Electronic Imaging 2008*, pages 68200H–68200H. International Society for Optics and Photonics, 2008.

[95] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, et al. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255, 2007.

[96] Z. Stone, T. Zickler, and T. Darrell. Autotagging facebook: Social network context improves photo annotation. In *Computer Vision and Pattern Recognition Workshops, CVPRW'08.* IEEE, 2008.

[97] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1), 1991.

[98] V. Vieira, P. Tedesco, and A. C. Salgado. Designing context-sensitive systems: An integrated approach. *Expert Systems with Applications*, 38(2):1119–1138, 2011.

[99] A. Vinciarelli, M. Pantic, and H. Bourlard. Social signal processing: Survey of an emerging domain. *Image and Vision Computing*, 27(12):1743–1759, 2009.

[100] U. Westermann and R. Jain. Toward a common event model for multimedia applications. *Multimedia, IEEE*, 14(1):19–29, 2007.

[101] F. Wolter and M. Zakharyaschev. Spatio-temporal representation and reasoning based on rcc-8. *KR2000: Principles of Knowledge Representation and Reasoning*, 2000.

[102] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5:975–1005, 2004.

[103] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*. IEEE, 2010.

[104] W. Yang, Y. Wang, and G. Mori. Recognizing human actions from still images with latent poses. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2030–2037. IEEE, 2010.

[105] R. Yerneni, C. Li, H. Garcia-Molina, and J. Ullman. Computing capabilities of mediators. *ACM Sigmod Record*, 28(2):443–454, 1999.

[106] Z. Zeng, M. Pantic, G. I. Roisman, and T. S. Huang. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1):39–58, 2009.

[107] L. Zhang, D. V. Kalashnikov, and S. Mehrotra. A unified framework for context assisted face clustering. 2013.

[108] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *Acm Computing Surveys (CSUR)*, 35(4):399–458, 2003.

[109] A. Zimmermann, A. Lorenz, and R. Oppermann. An operational definition of context. In *Modeling and using context*, pages 558–571. Springer, 2007.

Appendix A

Appendix

Supplementary material goes here.

A.1 Source Mapping

```
(:source google-calendar
   (:attrs event email time location title description attendee)
   (:rel ev type-of event)
   (:rel owner type-of person)
   (:rel ti type-of time-interval)
   (:rel ev occurs-during ti)
   (:rel participant type-of person)
   (:rel owner participant-of ev)
   (:rel participant participant-of ev)
   (:io disk (:db mongodb))
   (:type personal)
```

```
   (:axioms
      (:map ev event)
      (:map ti time)
      (:map owner.email email B)
      (:map ev.occurs-during time)
      (:map ev.occurs-at location)
      (:map participant attendee)
      (:map ev.title title U)
      (:map ev.description description U)))

(:source fb-user
   (:attrs id name birthday location work email)
   (:rel person type-of person)
   (:rel named-place type-of named-place)
   (:rel address type-of address)
   (:rel person works-at named-place)
   (:rel person lives-at address)
   (:io disk (:db mongodb))
   (:type personal)
   (:axioms
      (:map person.name name)
      (:map person.dob birthday)
      (:map address.street-address location.name)
      (:map named-place.name work.name)))


(:source email
```

```
   (:attrs from to cc)
   (:rel pf type-of person)
   (:rel pt type-of person)
   (:rel pc type-of person)
   (:io disk (:db mongodb))
   (:type personal)
   (:axioms
      (:map pf.email from)
      (:map pt.email to)
      (:map pc.email cc)))


(:source fb-relation
   (:attrs name1 name2)
   (:rel p1 type-of person)
   (:rel p2 type-of person)
   (:rel p1 knows p2)
   (:io disk (:db mongodb))
   (:type personal)
   (:axioms
      (:map p1.name name1 F)
      (:map p2.name name2 U)))

(:source academix
   (:attrs name1 name2)
   (:rel p1 type-of person)
   (:rel p2 type-of person)
```

```
   (:rel p1 knows p2)
   (:io disk (:db mongodb))
   (:type public)
   (:axioms
      (:map p1.name name1 F)
      (:map p2.name name2 U)))

(:source conferences
   (:attrs time location ltitle stitle url)
   (:rel conf type-of event)
   (:rel time type-of time-interval)
   (:rel loc type-of location)
   (:rel conf occurs-at location)
   (:rel conf occurs-during time)
   (:axioms
      (:map time time)
      (:map loc location)
      (:map conf.title ltitle)
      (:map conf.name stitle)
      (:map conf.url url)))


(:source confattendees
   (:attrs url name time location ltitle stitle)
   (:rel conf type-of conference)
   (:rel time type-of time-interval)
   (:rel loc type-of location)
```

```
   (:rel attendee type-of person)
   (:rel attendee participant-in conf)
   (:rel conf occurs-at location)
   (:rel conf occurs-during time)
   (:axioms
      (:map time time)
      (:map loc location)
      (:map conf.title ltitle)
      (:map conf.name stitle)
      (:map conf.url url)
      (:map attendee.name name)))


(:source keynotes
   (:attrs url time location title name)
   (:rel conf type-of conference)
   (:rel k type-of keynote)
   (:rel k subevent-of conf)
   (:rel attendee participant-in k)
   (:axioms
      (:map conf.url url)
      (:map attendee.name name)
      (:map k.location location)
      (:map k.time time)
      (:map k.title title)))

(:source sessions
```

```
   (:attrs url time location title name)
   (:rel conf type-of conference)
   (:rel k type-of session)
   (:rel k subevent-of conf)
   (:rel attendee participant-in k)
   (:axioms
      (:map conf.url url)
      (:map attendee.name name)
      (:map k.location location)
      (:map k.time time)
      (:map k.title title)))


(:source talks
   (:attrs url time location title name)
   (:rel conf type-of conference)
   (:rel k type-of talk)
   (:rel k subevent-of conf)
   (:rel attendee participant-in k)
   (:axioms
      (:map conf.url url)
      (:map attendee.name name)
      (:map k.location location)
      (:map k.time time)
      (:map k.title title)))

(:source conflunches
```

```
   (:attrs url time location title name)
   (:rel conf type-of conference)
   (:rel k type-of lunch)
   (:rel k subevent-of conf)
   (:rel attendee participant-in k)
   (:axioms
      (:map conf.url url)
      (:map attendee.name name)
      (:map k.location location)
      (:map k.time time)
      (:map k.title title)))


(:source tweets
   (:attrs url name)
   (:rel conf type-of conference)
   (:rel attendee type-of person)
   (:rel attendee participant-in conf)
   (:axioms
      (:map conf.url url)
      (:map attendee.name name)))

(:source fb-events
   (:attrs event name time)
   (:rel ev type-of event)
   (:rel p1 type-of person)
   (:rel ti type-of time-interval)
```

```
(:rel ev occurs-during ti)
(:rel p1 participant-of ev)
(:axioms
   (:map p1.name name)
   (:map ev event)
   (:map ev.occurs-during time)
   (:map ti time)))
```