University of
HUDDERSFIELD

University of Huddersfield Repository

Mantle, Matthew, Batsakis, Sotirios and Antoniou, Grigoris

Large Scale Reasoning Using Allen's Interval Algebra

Original Citation

Mantle, Matthew, Batsakis, Sotirios and Antoniou, Grigoris (2017) Large Scale Reasoning Using Allen's Interval Algebra. In: Advances in Soft Computing: 15th Mexican International Conference on Artificial Intelligence, MICAI 2016, Cancún, Mexico, October 23–28, 2016, Proceedings, Part II. Lecture Notes in Computer Science (11062). Springer, pp. 29-41. ISBN 9783319624280

This version is available at http://eprints.hud.ac.uk/id/eprint/29804/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/

Large Scale Reasoning Using Allen's Interval Algebra

Matthew Mantle, Sotirios Batsakis, and Grigoris Antoniou

University of Huddersfield
{m.e.mantle,s.batsakis,g.antoniou}@hud.ac.uk

Abstract. This paper proposes and evaluates a distributed, parallel approach for reasoning over large scale datasets using Allen's Interval Algebra (IA). We have developed and implemented algorithms that reason over IA networks using the Spark distributed processing framework. Experiments have been conducted by deploying the algorithms on computer clusters using synthetic datasets with various characteristics. We show that reasoning over datasets consisting of millions of interval relations is feasible and that our implementation scales effectively. The size of the IA networks we are able to reason over is far greater than those found in previously published works.

Keywords: Qualitative Temporal Reasoning, Distributed Computing, MapReduce

1 Introduction

Temporal information often exists in a qualitative form, for example *'Alice brushed her teeth before Bob went to bed'*. In this description no quantitative, numeric measurements of time are used, instead events are described in terms of how they relate temporally, one event occured before another. A number frameworks provide formalisms for representing and reasoning over qualitative temporal data such as this. One of the most widely used is Allen's Interval Algebra (IA)[3]. IA has been widely used in planning [4] and scheduling [10][8], but also seen application in areas as diverse as medicine [18] and analysis of crime [17].

Recent years have seen rapid growth in the volume of data computing practitioners are required to deal. Data from business transactions, web traffic, social media and smart devices is being generated at a scale that creates challenges for analysis, reasoning and querying. Much of this vast data has a temporal aspect. For example, the introduction of sensors and meters into a wide variety of objects has resulted in huge amounts of timestamped data. One open area for investigation is the application of qualitative temporal reasoning techniques to these large scale datasets.

– Many huge datasets contain temporal information that is only available in qualitative form, e.g. those originating in natural language such as social

media posts, email archives, case notes in electronic medical records. Reasoning over this data e.g. to build a timeline of medical history may only be possible using qualitative techniques.
- Even when the origin of the data is numeric it may be beneficial to represent time qualitatively. For example, in smart homes events such as entering a room are recorded with timestamps. Converting these timestamps to time intervals representing activities such as eating, sleeping etc. would allow for qualitative reasoning. Inferring the relations between these activities could then be used as a basis for scheduling automated tasks.
- Qualitative reasoning is often suited to situations where datasets are incomplete. Data collected from sensor devices is often noisy and records are often missing or incomplete. Rather than discarding records or assigning assumed values, qualitative reasoning frameworks are able to represent indefinite information, allowing them to provide correct (though less precise) solutions [9].

The main contribution of this paper lies in the development of parallel agorithms for qualitative temporal reasoning. An implementation of Allen's Interval Algebra has been developed for use with large scale datasets in a distributed environment. The rest of the paper is organised as follows. Section 2 provides background information on Allen's Interval Algebra and the Apache Spark cluster computing framework. Section 3 describes related work. Section 4 describes our implementation of Interval Algebra for the Spark platform. Experiments and results are provided in Section 5, and conclusions and future work in Section 6.

2 Background

2.1 Allen's Interval Algebra

Allen's Interval Algebra provides a formalism for qualitative descriptions of time [3]. Specifically, IA is concerned with time intervals. Allen describes 13 possible binary relations that can exist between a pair of intervals. These are shown in *Table 1.* Six of the relations have an inverse e.g. the inverse of during (d) is contains (di). Where the relation between two intervals is indefinite, a set is used to describe a disjunction of possible basic relations that could hold between the two intervals. For example $X\{b,m,o\}Y$, is interpreted as interval X either happens *before*, *meets* or *overlaps* interval Y. If no information is known, the relation could any of the thirteen basic relations $\{b,bi,m,mi,o,oi,s,si,d,di,f,fi,e\}$, this is denoted by I.

Relation	Symbol	Inverse	Visual Representation
X *before* Y	b	bi	XXXX YYYY
X *meets* Y	m	mi	XXXXYYYY
X *overlaps* Y	o	oi	XXXX YYYY
X *starts* Y	s	si	XXXX YYYYYYYY
X *during* Y	d	di	XXXX YYYYYYYY
X *finishes* Y	f	fi	XXXX YYYYYYYY
X *equals* Y	eq	eq	XXXX YYYY

Table 1. Allen's Basic Relations

A collection of three or more intervals with relations defined between these intervals can be represented as a directed graph, an *IA network*, where each node is a time interval and the label on each edge is the relation between a pair of intervals. Fig.1 shows an example. For simplicity, loops are not shown and nor are inverses of relations. Plus, for clarity, if there is no information regarding the relation between two intervals, rather than showing a disjunction of all 13 basic relations, *I*, the label simply is not shown.

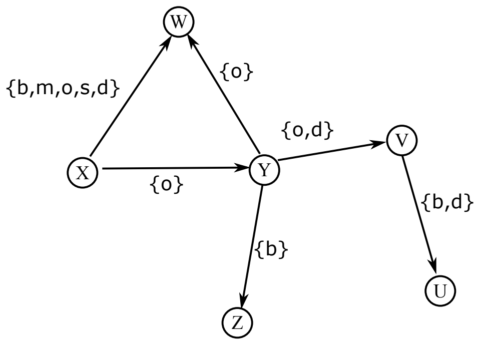

Fig. 1. A Simple IA Network

Reasoning using Interval Algebra The reasoning problem for Allen's algebra involves determining the smallest set of possible relations between all pairs of intervals. This can be viewed as a type of constraint satisfaction problem. A widely used approach for solving such problems, and the one proposed by Allen is path consistency [3], specified by the following formula, and described in more detail below.

$$\forall i, j, k R_s(i,k) \leftarrow R_i(i,k) \cap (R_j(i,j) \circ R_k(j,k))$$

Given any three intervals in the network *i*, *j* and *k*, the relation between *i* and *j*, and the relation between *j* and *k*, imply a relation between *i* and *k*. For example, in Fig. 1 the relations *X*{*o*}*Y*, and *Y*{*b*}*Z*, imply a relation between the intervals *X* and *Z*. Allen provides a table that describes all possible compositions of basic relations [3]. Part of this table can be seen in *Table 2*. Referring to *Table 2* we can deduce that the relation between *X* and *Z*, the composition of *overlaps* and *before*, is *before*, *X*{*b*}*Z*. Given two disjunctive relations R_1 and R_2 the composition of these relations is the union of the composition of each basic relation in R_1 with each basic relation in R_2. In Fig. 1, the relation between Y and U, implied by *Y*{*o,d*}*V* and *V*{*b,d*}*U* is such an example. Again looking up the compositions of the basic relations in *Table 2* and taking the union of the results, the composition of these disjunctive relations is {*b,o,s,d*}.

Table 2. Part of the Composition Table for Interval Algebra

	b	bi	d	di	o
b	{b}	*I*	{b,m,o,s,d}	{b}	{b}
bi	*I*	bi	{bi,oi,mi,d,f}	{bi}	{bi,mi,oi,d,f}
d	{b}	{bi}	{d}	*I*	{b,m,o,s,d}
di	{b,m,o,di,fi}	{bi,mi,oi,si,di}	{o,oi,s,si,d,di,e,f,fi}	{di}	{o,di,f}
o	{b}	{bi,mi,oi,si,di}	{o,s,d}	{b,m,o,di,fi}	{b,m,o}

The inference of relations places constraints on an IA network. For example, the inference arising from the composition of *X*{*o*}*Y* and *Y*{*o*}*W* implies the label on the edge *(X,W)* can be updated to {*b,m,o*}. Updating this value is only possible if the newly inferred relation is consistent with the existing relation between the pair of intervals, {*b,m,o,s,d*}. Consistency is checked by taking the intersection of the existing relation and the newly inferred relation. Basic relations are pairwise disjoint. Therefore, in the case where the intersection of two sets of possible relations is empty, this denotes an inconsistency in the network.

Reasoning over a temporal network is an iterative process. As new information is added to the network, and relations get updated, these updated relations form the basis for new inferences to be made, and further constrain existing relations. Once *Y*{*b,o,s,d*}*U* is added to the network, it is then possible, through the composition of *X*{*o*}*Y* and *Y*{*b,o,s,d*}*U*, to infer the relation between *X* and *U*. Therefore, the above path consistency formula is applied repeateadly until a fixed point is reached or an inconsistency is detected.

It is important to note that path consistency does not guarantee the consistency of the entire network [3], it only ensures the consistency of three node subsets of time intervals. As such it provides an approximation. In order to guarentee consistency, we would have to consider consistency between all nodes in a network, *n-consistency* (where n is the number of time intervals in the network). However, the computational complexity of implementing *n-consistency* is exponential with respect to n. An alternative would be to employ backtracking search alongside path consistency, which also has a runtime which is exponential. However, if expressiveness is limited by constructing sub-algebras, there are tractable sets of IA relations for which path consistency is a sound and complete method [11]. Despite path consistency being an approximation, it is a useful one. As Allen [3] states *'it provides us with enough temporal reasoning to participate in these tasks'* (comprehension, problem solving).

2.2 The Apache Spark Framework

Processing large quantities of data is typically accomplished using a cluster computing approach. A large dataset is split, distributed over a number of different machines, and then processed in parallel. Cluster computing frameworks provide programming models for developing distributed applications, as well as handling aspects such as managing resources, load balancing and fault tolerance. The implementation in this paper uses the Apache Spark framework [1]. The Spark API provides many operations that can be executed in parallel. These borrow many ideas from functional programming: *map*, *reduce*, *join*, *filter* etc. A key feature of the Spark framework is the capacity to maintain datasets in memory through a *cache* action [16]. This is especially useful for iterative tasks where the same dataset needs to be visited several times. In the more established MapReduce based frameworks data is written to disk after each *map* or *reduce* action. Spark datasets can be stored in memory allowing them to be easily re-used without this performance overhead. The implementation of a temporal reasoner is such an example. As an existing IA network needs to be merged with newly inferred relations repeateadly, it can take advantage of Spark's caching capabilities.

3 Related Work

When analysing related work, the closely related area of spatial reasoning is also considered. There are examples of frameworks that provide analysis of large scale spatial-temporal datasets. For example, SpatialHadoop [2] is a MapReduce framework designed specifically to work with spatial datasets. It extends the core Hadoop code base to provide spatial index structures and spatial operations e.g. R-trees, range queries, kNN and spatial join [7]. Experiments conducted on datasets consisting of billions of data items show SpatialHadoop offering significant performance advantages over standard Hadoop based implementations. However, SpatialHadoop uses metric point based representations, rather than qualitative ones. There appear to be no examples of qualitative spatial-temporal reasoning over large scale datasets using a distributed computing approach.

There are several examples of applications that provide qualitative spatial-temporal reasoning in a non-distributed environment. GQR (Generic Qualitative Reasoner) [15] is a tool that can reason over any qualitative constraint calculi, including IA. PelletSpatial [12] is a qualitative spatial reasoner built on top of the Pellet reasoner. Both these tools are limited in terms of the size of IA networks they are able to handle. The largest example of temporal reasoning using Allen's algebra appears to be by Anagnostopoulos et al [5]. who have successfully reasoned over datasets consisting of 10,000 relations on a single machine using the CHRONOS framework. To our knowledge, the approach taken in this paper is the first attempt to implement an IA reasoner using a distributed, parallel architecture, and to work with larger scale datasets, over 1 million relations.

4 Temporal Reasoning using Spark

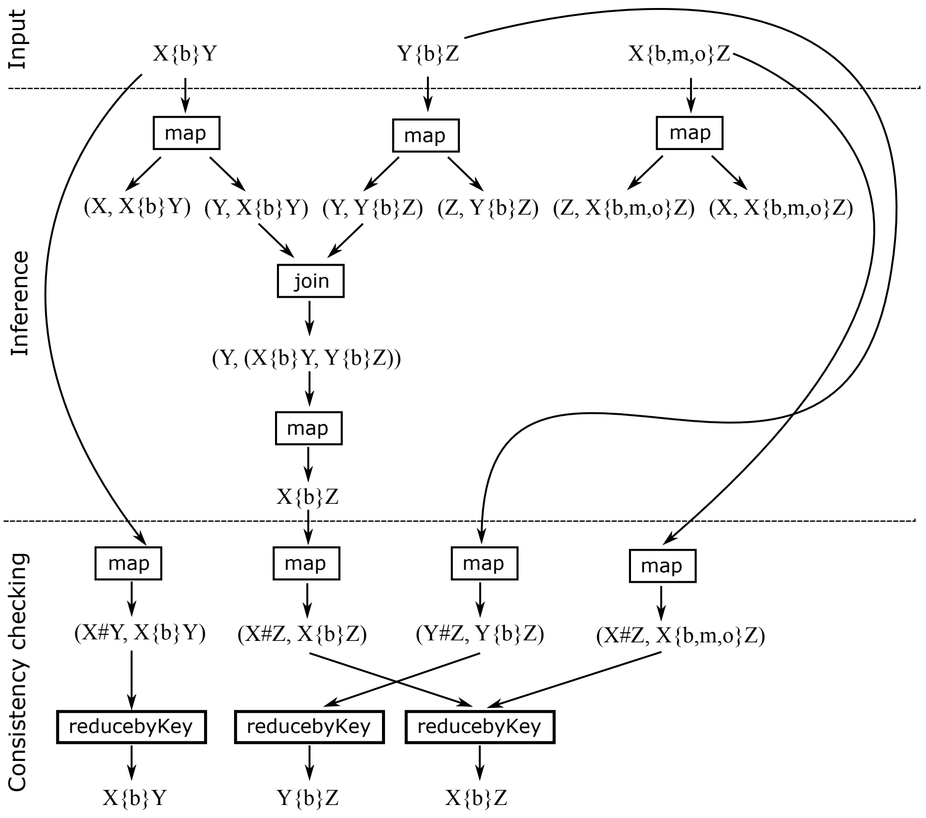

Fig. 2. Overview of Temporal Reasoning Using Spark

Fig. 2, provides an overview of our implementation of Allen's path consistency algorithm using Spark.

The input consists of an IA network. Specifically, the input is a collection of edges in the form *(i,R_{ij},j)*, where i and j are time intervals and R_{ij} the relation between these intervals. The full set of relations, I, is never used. The composition of I with any other relation always results in I, therefore no useful information can be obtained from inference based on the full set of relations.

The first phase is inference which corresponds to the composition operation in Allen's algebra. This is accomplished via a join between edges with common time intervals. The initial map operation outputs two key-value pairs for each edge in the input, once for the head interval, and once for the tail interval. At this point it is unknown, which, if any of the intervals, will form the basis for the join. In the above example, there is only one join possible, the one between $X\{b\}Y$ and $Y\{b\}Z$. Following the join operation, the actual inference takes place via a map transformation on pairs of edges. A Scala Map is used to implement Allen's composition table. The inferred relation is then deduced by looking-up the composition of each basic relation in the first edge with each basic relation in the second edge. The outputs of the inference stage are possible new relations between intervals, in this example, again, there is only one output, the edge $X\{b\}Z$. The application then moves onto a consistency checking phase where the output from inference is combined with the initial input. This combined dataset is mapped to key-value pairs where the key specifies a pair of intervals. Edges between the same pair of intervals will be sent to the same reduce process which computes the intersection of these relations for the final output.

The above diagram is a simplification, intended to give an overview of the application. Additional steps and optimisations were implemented as follows:

– It is necessary to add the inverse of each relation into the initial dataset. For example, consider two edges, $X\{b\}Y$ and $Z\{bi\}Y$. It is not possible to perform inference using these edges, as joins can only be performed where the head node of one edge matches the tail node of a second edge. If the inverse of each relation is inserted into the dataset ($Y\{bi\}X$ and $Y\{b\}Z$), inference is now possible. Therefore the transpose of the initial IA network was added as a pre-processing step.
– As mentioned above, path consistency is an iterative process. The two phases, inference and consistency checking are applied repeatedly until no new inferences are made. Algorithm 1 shows an high level overview of the temporal reasoner. Further details on the *inference* and *consistency* operations are provided below.

Algorithm 1 Temporal Reasoning Overview

```
IAnetwork=IAnetwork ∪ IAnetwork$^T$
IAnetwork=consistency(IAnetwork)
count=0
i=1
while IANetwork.count() $\neq$ count
    count=IAnetwork.count() //the size of the network taking into account new inferences
    newEdges=inference(IAnetwork,i)
    IAnetwork=consistency(IAnetwork ∪ newEdges)
    i++
end while
```

Limiting duplicates

One problem with a naive implementation is the derivation of duplicate edges. Consider the simple IA network shown in Fig. 3. In iteration 1, it is possible to infer a relation between the time intervals W and Y, and between X and Z. These two new edges ($W\{b\}Y$ and $X\{b\}Z$) will be added to the network. In iteration 2 it is then possible to infer the relation between the intervals W and Z. However, the reasoner will also infer $W\{b\}Y$ and $X\{b\}Z$ again, and add these edges to the network. These duplicates will be removed in the consistency checking phase but they add a significant and unnecessary data in the inference phase and will be inferred repeatedly with every iteration.

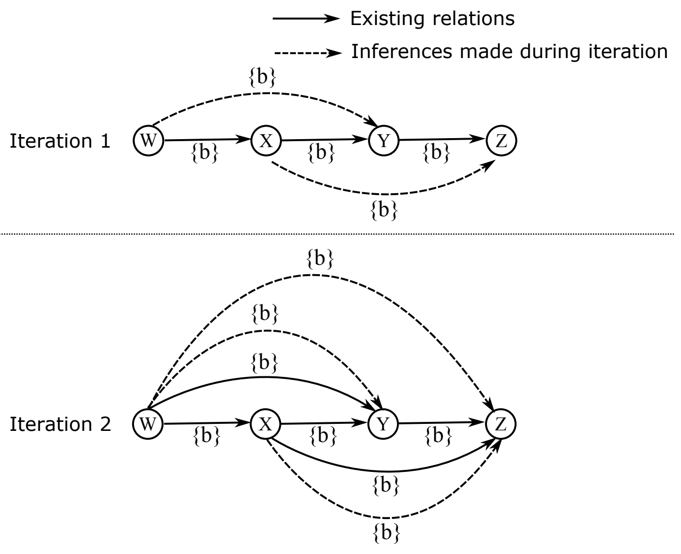

Fig. 3. Derivation of duplicate inferences

Urbani et al. faced a similar problem when reasoning over RDF triples using MapReduce [13]. They limited the joins that were possible based on the distance between nodes in a graph, specifically on one side of the join only edges with a distance of 2^{i-1} or 2^{i-2} were allowed (where i is the iteration number), on the other side of the join only edges with a distance greater than 2^{i-2} were allowed. The same approach is adopted in our temporal reasoner. In order to implement this, the distance between nodes is stored as part of each edge. When a new relation is inferred, the new edge is assigned a distance that is the sum of the joined edges' distances. During consistency checking the minimum distance is assigned to the reduced edge. The implementation can be seen in Algorithm 2 and Algorithm 3. It is important to note that duplication still takes place. This is because the same relation can be derived through different joins within the same iteration. In Fig. 3, $W\{b\}Z$ can be inferred twice within iteration 2, through the composition of $W\{b\}X$ with $X\{b\}Z$ and through the composition of $W\{b\}Y$ with $Y\{b\}Z$.

Algorithm 2 Inference

```
inference(iaNetwork, i )
    //iaNetwork: A collection of edges e.g. [(X,{b},Y,1), (Y,{b},Z,1), ...]
    //i: The iteration number e.g. 1
    tailEdges=iaNetwork
            .filter(edge ⇒ edge.distance=$2^{i-1}$∨edge.distance=$2^{i-2}$)
            .map(edge ⇒ (edge.tailInterval, edge)
    headEdges = iaNetwork.filter(edge ⇒ edge.distance> $2^{i-2}$)
            .map(edge ⇒ (edge.headInterval, edge)
    joinedEdges = tailEdges.join(headEdges)
    newEdges=joinedEdges.map((key,(tailEdge,headEdge))⇒(
            tailEdge.headInterval,
            lookUp(tailEdge.relation,headEdge.relation),
            headEdge.tailInterval,
            tailEdge.distance+headEdge.distance
            ))
    return newEdges
```

Algorithm 3 Consistency Checking

```
consistency(iaNetwork)
    //iaNetwork: A collection of edges e.g. [(X,{b},Y,1), (Y,{b},Z,1), ...]
    keyedEdges=iaNetwork
            .map(edge ⇒ (edge.tailInterval+'#'+edge.headInterval, edge))
    consistentEdges=keyedEdges.reduceByKey((edgeA,edgeB)⇒(
                                edgeA.tailInterval,
                                edgeA.relation ∩ edgeB.relation,
                                edgeA.headInterval,
                                Math.min(edgeA.distance,edgeB.distance)
            ))
    return consistentEdges
```

5 Evaluation

The purpose of the evaluation was to investigate the potential and limitations of the algorithms described above. In particular the evaluation was concerned with the issue of scalability and the capacity of the reasoning application to deal with large datasets.

Platform

The experiments were conducted by implementing the algorithms in the Scala programming language using the Apache Spark framework. Some initial experiments were also carried out using the Hadoop MapReduce framework. However, as expected, the algorithms implemented in Spark ran much faster than those in Hadoop. Spark's in-memory caching provided a significant advantage over the Hadoop based implementation. The experiments were carried out using Amazon's Elastic Cloud Compute (EC2) platform. Although EC2 has potential drawbacks such as lack of data locality, virtualised hardware, it is a widely used system as it negates the need for users to build and manage a cluster themselves and provides a typical real-world implementation for a distributed application. For the majority of the experiments a cluster consisting of four machines was used, each with four virtual CPUs and 30GB of memory. One of the experiments focussed on assessing the impact of increasing computing resources. In this case the size of the cluster was varied.

5.1 Experiments

The experiments were conducted using synthetic datasets. The features of each input dataset are discussed under each experiment description.

- **Experiment 1**. A number of time intervals were generated by randomly selecting points on a line for the start and end points. For each time interval, the relation between it and one other time interval was calculated. This gave rise to an input graph with an equal number of nodes (time intervals) and edges (relations between time intervals). Tests were run over datasets consisting of between 2 and 10 million edges.
- **Experiment 2**. Experiment 2 focussed on exploring the effect of increased computing resources on reasoning time. An input dataset of 10 million edges, generated in the same way as in Experiment 1, was used. The application was then run using a cluster size of between 2 and 16 machines.
- **Experiment 3**. Reasoning over an IA network as described in Experiment 1 does not result in a complete graph. To investigate the worst-case scenario of reasoning over an IA network where the relation between every pair of intervals can be inferred, a connected graph consisting of a consecutive sequence of time intervals was used. Datasets of between 2,000 and 32,000 relations were generated based on this worst-case scenario.

5.2 Results

- Fig.4 shows the results for Experiment 1. The reasoning application demonstrates close to linear time performance and even considering the relatively small size of the cluster, the application is easily able to cope with an input of ten million edges. This is far larger in scale than previous applications that have reasoned over IA networks (limited to 10,000 edges), and shows the benefit of a distributed, parallel approach.
- Fig.6 shows the results for Experiment 2, the scalability of the reasoner in terms of the number of nodes in the cluster. As would be expected, an increase in the size of the cluster results in reasoning time decreasing. Fig. 7 shows scaled speed-up, a standard metric used when evaluating the performance of parallel systems. Ideally a parallelised system should display linear

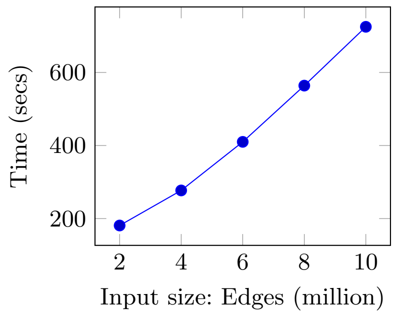

Fig. 4. Experiment 1: Runtime as a function of input size

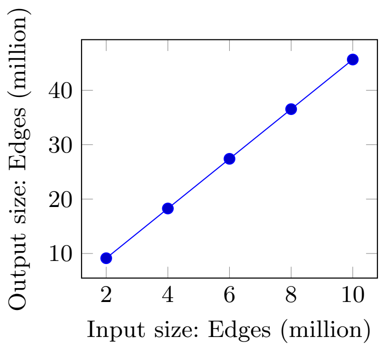

Fig. 5. Experiment 1: Output (number of inferred edges) as a function of input size

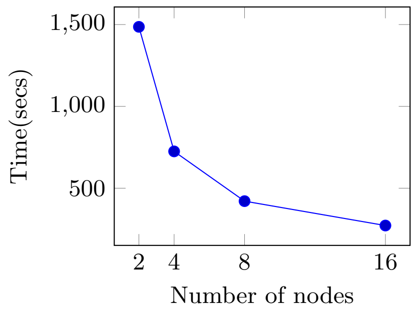

Fig. 6. Experiment 2: Runtime as a function of number of nodes

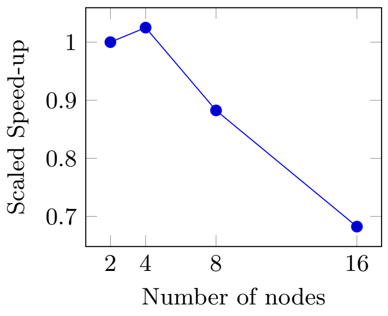

Fig. 7. Experiment 2: Scaled speed-up

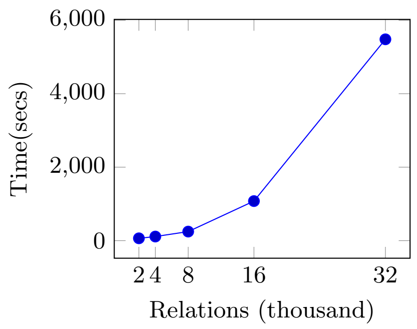

Fig. 8. Experiment 3: Runtime as a function of input size (worst-case scenario)

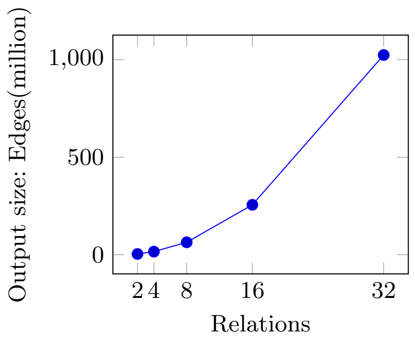

Fig. 9. Experiment 3: Output (number of inferred edges) as a function of input size (worst-case scenario)

speed-up, as we double the number of machines in the cluster the reasoning time should half. Apart from an increase with 4 nodes, the application shows sub-linear speed-up. However, this is consistent with many other ap-

plications running on distributed frameworks, and can be explained by the inevitable overheads of starting and managing jobs. The number of jobs remains constant regardless of cluster size. At some point adding additional nodes has little impact on processing time.
- The results for Experiment 3, Fig. 8, show two significant differences in the runtime compared to Experiment 1. Firstly, even though the input datasets are considerably smaller, in many cases the runtime is longer. Secondly, runtime does not increase linearly with regards to the input size. These differences can be attributed to the volume of data generated. Fig. 5 and Fig. 9 show the numbers of generated edges in the corresponding experiments. In Experiment 1, even when reasoning over an initial graph with 10 million edges, the final output is less than 50 million edges. In Experiment 3, a complete graph is generated. For an input size of 32,000 edges over 1 billion edges are generated. Plus, unlike Experiment 1, when generating a complete graph the output does not grow linearly with regards to the input size.

6 Conclusions and Future Work

To our knowledge this is the first attempt to explore the feasibility of large scale reasoning using Allen's Interval Algebra. The results are encouraging. We have shown that, depending on the characteristics of the dataset, it is possible to reason over very large IA networks. Future work will expand the approach to look at:

- The use of real world datasets. The experiments above use datasets with specific characteristics that we have designed. The next step is to work with real world datasets.
- Other areas of qualitative reasoning. The approach described above can be modified to work with other qualitative calculus. For example, Vilain and Kautz's Point Algebra [14] or spatial reasoning using Region Connection Calculus (RCC-8) [6].
- Temporal knowledge bases often feature both qualitative and metric information. We plan to extend the approach described here to reason using both quantitative and qualitative information in large scale datasets.

References

1. Spark. `https://spark.apache.org/`. Accessed: 2015-06-30.
2. Spatialhadoop. `http://spatialhadoop.cs.umn.edu/index.html/`. Accessed: 2016-07-06.
3. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
4. James F Allen. Temporal reasoning and planning. In *Reasoning about plans*, pages 1–67. Morgan Kaufmann Publishers Inc., 1991.

5. Eleftherios Anagnostopoulos, Euripides G. M. Petrakis, and Sotiris Batsakis. CHRONOS: improving the performance of qualitative temporal reasoning in OWL. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 309–315, 2014.
6. Anthony G. Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.
7. Ahmed Eldawy and Mohamed F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1352–1363, 2015.
8. Richard W. Focke, L. O. Wabeke, J. P. de Villiers, and Michael R. Inggs. Implementing interval algebra to schedule mechanically scanned multistatic radars. In *Proceedings of the 14th International Conference on Information Fusion, FUSION 2011, Chicago, Illinois, USA, July 5-8, 2011*, pages 1–7, 2011.
9. Yumi Iwasaki. Real-world applications of qualitative reasoning. *IEEE Expert*, 12(3):16–21, 1997.
10. Lenka Mudrová and Nick Hawes. Task scheduling for mobile robots using interval algebra. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 383–388, 2015.
11. Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 356–361, 1994.
12. Markus Stocker and Evren Sirin. Pelletspatial: A hybrid RCC-8 and RDF/OWL reasoning and query engine. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, 2009.
13. Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using mapreduce. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, pages 634–649, 2009.
14. Marc B. Vilain and Henry A. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, pages 377–382, 1986.
15. Matthias Westphal, Stefan Wölfl, and Zeno Gantner. GQR: A fast solver for binary qualitative constraint networks. In *Benchmarking of Qualitative Spatial and Temporal Reasoning Systems, Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-02, Stanford, California, USA, March 23-25, 2009*, pages 51–52, 2009.
16. Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*, 2010.
17. Abbas K Zaidi, Mashhood Ishaque, and Alexander H Levis. Combining qualitative and quantitative temporal reasoning for criminal forensics. In *Mathematical Methods in Counterterrorism*, pages 69–90. Springer, 2009.
18. Li Zhou and George Hripcsak. Temporal reasoning with medical data - A review with emphasis on medical natural language processing. *Journal of Biomedical Informatics*, 40(2):183–202, 2007.