University of Zurich[UZH]

Dynamic Distributed Constraint Optimization in Signal/Collect

Thesis January 14, 2015

Daniel Hegglin
of Oerlikon ZH, Switzerland

Student-ID: 08-721-102
dani.hegglin@gmail.com

Advisor: **Mihaela Verman**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

First and foremost, I'd like to thank my advisor Mihaela Verman, Research Assistant and PhD Candidate at the DDIS group for her excellent support during the course of this thesis. She was always reachable and helped me with many technical and formal questions. Without her continous efforts and valuable inputs, this work would not have been possible.
Special thanks go to Dr. Thomas Scharrenbach for arranging the thesis proposal and his guidance at the beginning of this work. I'd also like to thank Philip Stutz for his work on the Signal/Collect framework that made my research possible and his support in technical questions regarding the usage of the framework.
Finally, I'd like to thank the Dynamic and Distributed Systems Group (DDIS) at the University of Zurich and Prof. Abraham Bernstein for the opportunity to write my master thesis in their department.

Zusammenfassung

Distributed Constraint Optimization ermöglicht Problemlösungen in beispielweise Terminplanung, Verkehrsflusskontrolle oder dem Management von Sensor Netzwerken. Es ist ein gut erforschtes Feld und es wurden viele verschieden Algorithmen zur Berechnung vorgestellt. Allerdings wird häufig von einer statischen Problemdefinition ausgegangen und der Aspekt von in der Realität häufig auftretenden Änderungen an der Problemstellung findet wenig Beachtung. Ausserdem fehlt es an einem theoretischen Fundament und standardisierten Verfahren um die Performanz von DCOP Algorithmen hinsichtlich sich ändernder Probleme zu erfassen. Diese Arbeit hatte das Ziel das Verhalten und die Leistung von verschieden Arten von DCOP Algorithmen in dynamischen Umgebungen mit einem Fokus auf lokale, iterative Algorithmen und Hauptaugenmerk auf den Max-Sum Algorithmus zu untersuchen. Zum Vergleich wurde eine komplette und eine lokal, iterative "message-passing" sowie eine "best-response" Variante implementiert. Zum Test eines realen Problems wurde Terminplanung ausgewählt und als DCOP formuliert. Es wurde ausserdem ein Framework entwickelt, welches die dynamische Änderungen von Constraints, Variablen und der Problemdomäne ermöglicht. Die Algorithmen wurden mit Fokus auf Qualität über Zeit, sowohl in einer statischen wie auch in einer dynamischen Umgebung getestet. Diese Arbeit schlägt ausserdem eine Lösung zur Speicherung, Weiterverarbeitung und Überwachung der Resultate der Berechnungen in Echtzeit vor, welche die Performanz der Algorithm nicht beeinflusst.

Abstract

Distributed constraint optimization allows to solve problems in domains like scheduling, traffic flow management or sensor network management. It is a well-researched field and various algorithms have been proposed. However, the dynamic nature of many of these problems in the real world has often been overlooked from researchers and problems are assumed to be static during the course of the computation. The benchmarking of DCOP algorithms with changing problem definitions currently lacks a theoretical foundation and standardized protocols. This thesis aimed to measure the performance of different types of DCOP algorithms on dynamic problems with a focus on local-iterative algorithms and especially on the MaxSum algorithm. A complete, a local-iterative message-passing and a local-iterative approximate best-response algorithm for distributed constraint optimization have been implemented for comparison. The meeting scheduling problem has been mapped as distributed constraint optimization problem as a real-world use case for benchmarking. A framework has been designed that allows dynamic changes to constraints, variables and the problem domain during runtime. The algorithms have been benchmarked in a static as well as in a dynamic environment with various parameters and with a focus on solution quality over time. This thesis further proposes a solution to store, further process and monitor the results of the computation in real-time without affecting the performance of the algorithms.

Table of Contents

1

Introduction

1.1 Motivation & Goal

Distributed Constraint optimization allows to solve a broad category of problems where multiple agents are involved and a global utility function needs to be optimized. Problems range from graph coloring [Modi et al., 2005] to task allocation and scheduling [Zhang et al., 2002], traffic congestion management [Leeuwen et al., 2002] or in disaster recovery [Hiroaki et al., 1999]. The distributed variant of constraint optimization has been extensively adressed by researchers and the formulation of numerous algorithms with diverse design approaches. However, most of those algorithms were designed and most studies are conducted on the premise that problems are static in their predefined state and do not change over the course of the problem solving process. This might work as as a step by step procedure, but it does not work in a distributed manner with multiple agents [Petcu and Faltings, 2007]. As a matter of fact, many of these problems have dynamic properties and in an environment with ever increasing complexity and speed, those become ever more relevant for real-world applications. Constraints can change, but also the involved variables as well as the problem domain itself. Imagine for example a organizational software of a global logistics company where vehicles can get damaged and orders could be changed to other adresses or even canceled. Recalculation of the complete problem with a new static definition seems inefficient. Research in dynamic distributed constraint optimization is sparse. The benchmarking of DCOP algorithms with changing problem definitions currently lacks a theoretical foundation and standardized protocols [Mailler and Zheng, 2014].

This thesis tries to explore dynamic distributed constraint optimization by implementing a complete and two local-iterative variations of algorithm approaches and compare their performance in a dynamic environment. The complete algorithm acts as a baseline and the main focus is on the performance of the local-iterative types of algorithms. They do not guarantee complete solutions but are able to provide a good solution faster as they calculate on a local level with a lower communication overhead. They also have been proven to be more scalable because of this lack of organizational overhead [Chapman et al., 2011]. These attributes indicate their potential ability to adapt faster to problem changes and keep a better stability. The main focus is on the abilities of the

MaxSum algorithm, which is a local-iterative message-passing algorithm. It is further a goal to show ways of benchmarking these type of problems from various aspects. The real-world use case for the thesis will be meeting scheduling and a framework for dynamic changes will be implemented on top of Signal/Collect, a graph processing engine developed at the Dynamic and Distributed System Group at the Departement of Informatics of the University of Zurich.

The implemented software is made available online under the Apache license version 2.0.[1][2]

1.2 Structure

First, an overview will be given about various definitions and aspects of constraint optimization in general, as well as the aspects of the distributed and dynamic variations. Further, an examination will be provided about different approaches of algorithms to solve constraint optimization problems and their advantages and disadvantages in context of solution quality over time, scalability and adaptability to changes.

Secondly, the meeting scheduling problem definition and the mapping to a distributed constraint optimization problem, as well as the algorithm mapping to the Signal/Collect programming model will be detailled in the design chapter. Further, the design considerations for a framework for dynamic changes will be explained and the solution for data collection will also be briefly introduced.

Finally, the performed benchmarks will be evaluated and discussed. In a first series, the algorithms will be tested in static environments to evaluate the implementation. In a second series, various tests on changing constraints, variables and the domain with different rates and different problem densities will be run to determine the performance of the algorithms in dynamic environments. To wrap up, further work possibilities and limitations of the thesis will be pointed out and a conclusion about the achieved contributions and results will be given.

[1] https://github.com/danihegglin/DynDCO
[2] http://www.apache.org/licenses/LICENSE-2.0

2

Background & Related Work

In this section, constraint optimization and the distributed, as well as dynamic variants are briefly explained and brought into context of the related work. Also, the meeting scheduling problem is described and different algorithm designs and their advantages and disadvantages are explained and related work is mentioned.

2.1 Dynamic Distributed Constraint Optimization

A constraint optimization problem contains of a set of variables $V = \{V_1, V_2, ..., V_n\}$. These variables are assigned to a value or state $s_j \in S_j$, which is contained in a set of possible values defined by a finite problem domain $D = \{D_1, D_2, ..., D_n\}$. A constraint $C =< V_c, R_c >$ contains one (unary), two (binary) or multiple (k-ary) variables and their relationship. The constraint defines a rule for the variables that needs to be fulfilled. One of those rules could be that none of the variables should take the same value. This would for example be the case for a meeting scheduling problem where none of the meetings should take place at the same time.
A utility function $u_{c_k}(S_{c_k})$ needs to be formulated that defines a certain cost respectively reward for a given configuration of the involved states. The global utility function u_g would then be the summation of all utility functions of all constraints.

$$u_g(s) = u_{c_1}(s_{c_1}) \oplus \cdots \oplus u_{c_k}(s_{c_k}) \oplus \cdots \oplus u_{c_l}(s_{c_l})$$

Constraints can be attributed with varying levels of importance through weighting. One can, instead of so-called soft constraints, define hard constraints by multiplying their utility instead of using addition in the global utility function. By defining the utility of a violated hard constraint as 0, the global utility would also go to 0 if this hard constraint is not satisfied [Chapman et al., 2011, Petcu and Faltings, 2003]. A problem only containing hard constraints would represent a constraint satisfaction problem.

$$u_g(s) = \prod_{hc_k \in HC} u_{SC_g}(s) \Bigg(\sum_{sc_k \in SC} u_{SC_g}(s) \Bigg)$$

The definition of a distributed constraint problem extends the basic constraint optimization by distributing sets of variables to autonomous agents. These agents all

have the goal to maximize the utility of their variables in a private utility function and thereby also contribute to a global utility function. Agents whose variables are linked to a common constraint are called neighbours [Chapman et al., 2011, Farinelli et al., , Petcu and Faltings, 2003].

As a further extension to the optimization, the problem definition is moved from a static to a dynamic attribute. Constraints can change and therefore change neighbourhoods and the outcome of private and global utility functions. A change of constraints inherently changes the area of satisfying solutions if hard constraints have been included in a problem definition. Nguyen et al. state that changing the constraints might lead to the discovery of a better global optima.[Nguyen and Yao, 2012]. Mailler et al. define a dynamic DCSP as a sequence of DCSPs $\{P_0, P_1, ..., P_n\}$ where every DCSP is a static problem definition. P_i is therfore a result of the previous DSCP in the sequence and the added and removed constraints: $P_i = P_{i-1} + c_{i^a} - c_{i^r}$ [Mailler and Zheng, 2014]. This definition should also hold for DCOPs. Utility functions could also be dynamically changed. Modifying this property could especially have an impact on real-world problems like meeting scheduling, where it could move the global optima from one disconnected solution space to another[Nguyen and Yao, 2012]. Furthermore, variables could be added or removed in a dynamic setting and the problem domain D also could be changed during the course of the problem solving process.

2.2 Meeting Scheduling Problem

Scheduling is the problem of allocating tasks to a given set of ressources that is usually limited with a time component. The meeting scheduling problem is a exemplary type of this family of problems and is supposedly well-know to all of us. Participants of a meeting have private schedules with preferences when a meeting should be held according to their calendar and the challenge is to identify a time for a meeting that maximizes the preferences of all participants while being valid in then sense that every person is able to attend [Farinelli et al.,]. Angulo et al. have formally defined a meeting scheduling problem as:

- $P = p1, p, ...pn$ is the set of people where every person has a calendar that hilds r slots, $S = s1, s2, s3$
- $M = m1, m2, ..., m3$ is a set of k meetings
- $At = at1, at2, ..., atk$ defines all attendee's of a meeting

The c parameter has been neglected as it is not relevant to this thesis [**?**]. From the definition of a valid solution, one can derive two important criteria to the problem solving process:

Validity Criterium 1. All participants need to agree on the same time for the meeting.

Validity Criterium 2. Meetings need to be scheduled in a way that there are no overlaps of meeting times in the schedules of the participants.

There is further an inherent privacy aspect to the problem. Meeting participant's are often not willing to share their schedules with others except for finding a time for the specific meeting. It will later be shown that some of the algorithms can guarantee this privacy to a certain degree. The meeting scheduling problem will be mapped as a distributed constraint optimization problem in the design chapter[Farinelli et al., , **?**].

2.3 Algorithm Design Approaches

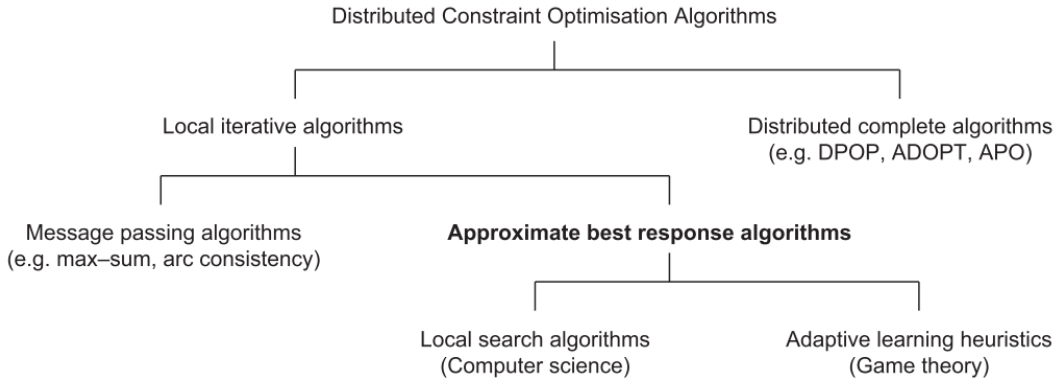

Figure 2.1: Categorization of DCO algorithms [Chapman et al., 2011]

Chapman et al. categorize distributed constraint optimization algorithms into local iterative and distributed complete algorithms. They further divide local iterative into message passing algorithms and approximate best response algorithms (Figure 2.1). All of these approaches fullfill the anytime property, i.e they can provide a solution at every timepoint during calculation and all of them can be run synchronously as well as asynchronously with varying performances. The following subsections are going to explain the differences between the three categories and introduce the concrete algorithms from these three different approaches that have been chosen for benchmarking. Advantages, as well as disadvantages will be described and which behaviour one can expect of these algorithms under certain parameters.

2.3.1 Complete

[Chapman et al., 2011]

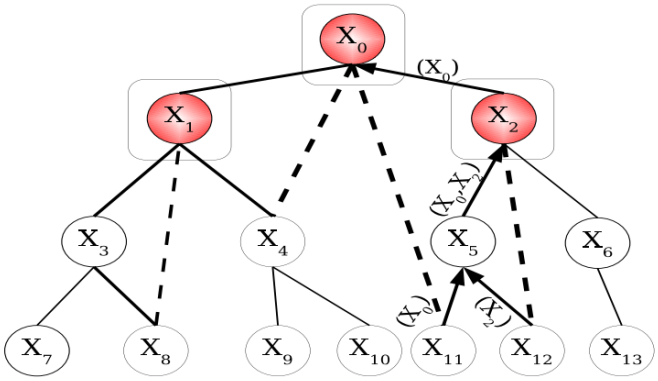

Figure 2.2: Pseudotree for DPOP [Petcu and Faltings, 2003]

2.3.2 Local-Iterative - Best Response

- neighbourhoods - only communicates his state - reacts to the states of others - privacy aspects

[Chapman et al., 2011] [Chapman et al., 2010] [Maheswaran et al., 2006]

- Local iterative approximate best-response algorithms, such as the distributed stochastic algorithm (Tel, 2000; Fitzpatrick Meertens, 2003), the maximum-gain messaging algorithm (Yokoo Hirayama, 1996; Maheswaran et al., 2005), fictitious play (Brown, 1951; Robinson, 1951), adaptive play (Young, 1993, 1998), and regret matching (Hart Mas-Colell, 2000). In this class, agents exchange messages containing only their state, or can observe the strategies of their neighbours. In game-theoretic parlance, this is known as standard monitoring2 , and, as the name suggests, is a typical informational assumption implicit in the literature on learning in games.

- advantages - disadvantages

- mgm description

2.3.3 Local-Iterative - Message Passing

[Chapman et al., 2011] -¿

- advantages -disadvantages: cycles, acyclic - privacy aspects - maxsum description

[?] -¿ fist paper

BIPARTITE GRAPH DEFINITION:

FACTOR GRAPH

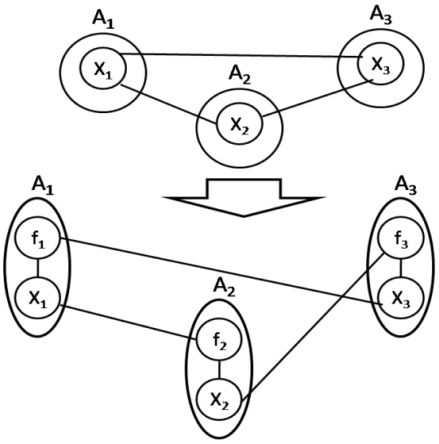

Figure 2.3: Conversion of a general DCOP to a factor graph[**?**]

Each node represent- ing a variable of the original DCOP is connected to all function-nodes that represent constraints that it is involved in. Similarly, a function-node is connected to all variable-nodes that represent variables in the original DCOP that are included in the constraint it represents. Agents

From variable to function: From function to variable:

Finally, we note that if messages are continuously propagated, and the states of the agents are continuously updated, then the algo- rithm may be applied to dynamic problems where the interactions between agents, or the utilities resulting from these interactions, may change at any time. For example, within tracking problems where the decentralised coordination algorithm is being used to fo- cus different sensors onto different targets (as described in [15]), then the utilities of each sensor are continually changing due to the changing position of targets, and the actions of other sensors. Thus, by continually propagatingmessages each agent is able tomaintain a continuously updated estimate of the state that it should adopt in order to maximise social welfare in this dynamic problem4. Farinelli

3

Design

In this section, the benchmark problem will be defined and mapped as DCOP, the framework design will be explained and the mapping of the algorithms on to the Signal/Collect framework will be described. Additionally, the design and considerations regarding the monitoring platform will be presented.

3.1 Meeting Scheduling Problem

3.1.1 Formal Definition as DCOP

The formulation of the meeting scheduling problem follows the basic definition of a distributed constraint optimization problem. Agents, variables and their relationships, as well as constraints shall be formulated. The components of a meeting scheduling problem are participants, their schedule, meetings and a given timeframe. For the sake of simplicity, it was decided to not take travel time between meetings or other parameters into consideration as certain researchers have done [Grubshtein et al.,]. It was also decided to use utilities instead of costs.

Definition 1. *Participant - has preferences and meeting he/she need to attend*

Definition 2. *Meeting - has participants and needs to be held at an agreable time*

Maheswarana et al. propose three different ways of mapping a meeting scheduling problem to variables (Figure 3.1). TSAV (Time Slots As Variables), EAV (Event As Variables) and PEAV (Private Events As Variables). In EAV, every participant holds a private variable containing the timeslot that should be used for a meeting. PEAV is a modification of the EAV paradigm where agents don't share their local valuations [Maheswaran and Tambe, 2004, , 2005]. It was decided to follow the PEAV principle and model every meeting participation of an agent as one variable instead of using timeslots as variables. An agent therefore can hold multiple variables. This paradigm has also been tried by other researchers, which further established confidence in the decision [Petcu and Faltings, 2003].

Definition 3. *Agent - holds one variable per meeting participation*

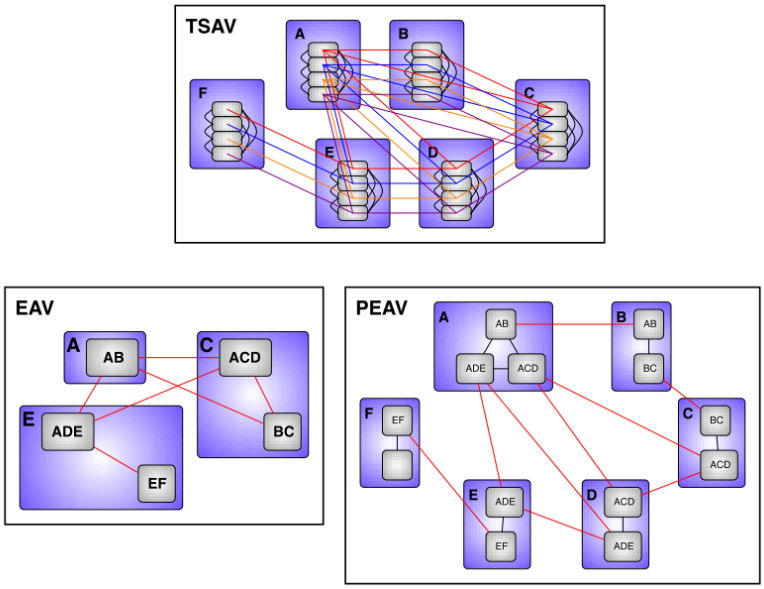

Figure 3.1: Different paradigms of mapping the meeting scheduling Problem [Maheswaran and Tambe, 2004].

Definition 4. *Variable - represents one meeting participation*

A variable takes on a value $s_i \in S_i$ in a defined problem domain D. In the formulation of the meeting scheduling problem, the domain represents a finite set of timeslots and the variable assigns to one of these timeslots. This value represents the currently locally chosen timeslot for a specific meeting.

Definition 5. *Domain - holds a finite set of possible timeslots to schedule a meeting*

Definition 6. *Value - assigns to a timeslot of the available timeslots in the Domain*

From the problem definition in the background & related work chapter one can derive soft and hard constraint. Soft constraints can possibly be constructed from the preferences of the participants and utilized to maximize the utility [Franzin et al.,]. As the global utility should be optimized, the participants could as a consequence have a low local utility. Three differently weighted soft constraints have been defined to model preferences of agents. Preferred timeslots gain the highest utility, followed by free timeslots and blocked timeslots, which gain no value at all. Further, a timeliness soft constraint was defined that adds a higher utility to earlier timeslots. All of the soft constraints have unary relationships, i.e. are local. Additionally, two hard constraints with k-ary relationships to variable neighbours need to be formulated. The first would be an equality constraint for assigned values for a meeting timeslot between all variables related to a meeting. The second is a difference constraint of assigned values between all variables of an agent [Farinelli et al., , **?**].

So, a local utility function $u_l(s)$ would include the sum of all soft constraints multiplied by the product of the hard constraints analogous to the global function defined in chapter 2.

$$u_l(s) = \prod_{hc_k \in HC} u_{SC_g}(s) \Big(\sum_{sc_k \in SC} u_{SC_g}(s) \Big)$$

3.1.2 Problem Dataset Generation

During the course of the thesis, it was necessary to find a dataset for benchmarking. The Frodo2 framework does provide a couple of datasets for meeting scheduling. But because it was considered that in the benchmarks one would need to be able to produce problems with different densities and scale to high numbers of participants, it was decided to generate meeting participations and schedules randomly.

The meeting participations are limited to a maximum of five. BECAUSE

Distributed Meeting Scheduling: CHAPTER 10

3.2 Mapping of DPOP

based on: A Scalable Method for Multiagent Constraint Optimization breakdown

Algorithm 1 DPOP Pseudocode

```
1: procedure MyProcedure
2:     stringlen ← length of string
3:     i ← patlen
4: top:
5:     if i > stringlen then return false
6:     j ← patlen
7: loop:
8:     if string(i) = path(j) then
9:         j ← j − 1.
10:        i ← i − 1.
11:        goto loop.
12:        close;
13:    i ← i + max(delta_1(string(i)), delta_2(j)).
14:    goto top.
```

3.2.1 Graph Structure

3.2.2 Vertices

3.3 Mapping of MGM

This is the pseudocde for the Maximum-Gain messaging algorithm that has been used to implement [Chapman et al., 2011].

Algorithm 2 MGM Pseudocode

```
    procedure MyProcedure
2:      stringlen ← length of string
        i ← patlen
4:  top:
        if i > stringlen then return false
6:      j ← patlen
    loop:
8:      if string(i) = path(j) then
            j ← j − 1.
10:         i ← i − 1.
            goto loop.
12:         close;
        i ← i + max(delta_1(string(i)), delta_2(j)).
14:     goto top.
```

3.3.1 Graph Structure

3.3.2 Vertices

3.4 Mapping of MaxSum

This is the pseudocode of the MaxSum algorithm that has been used for the implementation [**?**].

Max-sum (node n)

3.4.1 Graph Structure

– describe tests and thinking behind the chosen structure!!!!

Algorithm 3 Maxsum Pseudocode

```
    procedure MyProcedure
        stringlen ← length of string
 3:     i ← patlen
    top:
        if i > stringlen then return false
 6:     j ← patlen
    loop:
        if string(i) = path(j) then
 9:         j ← j − 1.
            i ← i − 1.
            goto loop.
12:         close;
        i ← i + max(delta1(string(i)), delta2(j)).
        goto top.
```

3.4.2 Vertices

3.5 Framework

3.5.1 Signal / Collect

It was decided to build a specific framework for benchmarking dynamic problems. The foundation of the framework is the Signal/Collect framework [Stutz et al., 2010][1], which is built on top of Akka[2] and written in Scala[3]. It is a graph processing engine with a programming model that features vertices and edges. The vertices have a state and send signals along the edges to other vertices, which can contain any datatype. The signal usually is their state or in context of their state. The vertices collect the signals and perform a compution on the collected data, before adjusting their state according to the calculations and sending out the newly generated signal. This model allows to reduce complex algorithms to a few lines of code and is applicable for many problems. The reason for choosing this framework in this thesis is the structural fit to distributed constraint optimization problems and the possiblity to add and remove vertices during runtime. This allows for dynamically changing problem computations. Further, the capability of running problems asynchronously or with synchronous signal steps and the possibility to distribute the system on multiple machines add to the advantages.

[1] http://uzh.github.io/signal-collect
[2] http://akka.io
[3] http://www.scala-lang.org

3.5.2 Parameters & Modes

This framework was designed with the benchmarks in mind. It should be possible to pass the general run parameters like which algorithm to use, which Signal/Collect run mode (synchronous/asynchronous) to use, normal run mode or dynamic variations (changeConstraints, changeVariable, changeDomain) with specific parameters. For testing, two test modes were implemented: SingleTest and MultiTest. Single Test is for single runs obviously and MultiTest allows to specifiy how many runs each specification should be doing to create a median of those runs, how much the agents respectively the meetings should increase after a specification has been processed and where to stop. By this, it is possible to explore an area of specifications. Finally, it should be possible to define different problems. For this thesis, the parameters for problem density (blocked timeslots percentage), timeslots, meetings and agents were defined.

3.5.3 Functionality & Structure

The considerations for the structure were that I needs to be hierarchical to allow different constraint optimization problems to be run on the framework. So, the hierachy goes from a Base to a Dynamic to a Problem level. It should include ProblemFactories and Problems, GraphFactories and Graphs, as well as Vertices.

- Problem Factory: Problem -¿ MeetingSchedulingProblem class diagram
- Graph Factory: Graph -¿ DPOP Graph, MGM Graph, MaxSum Graph explanation - Graph
- BasicVertex -¿ DynamicVertex -¿ MeetingSchedulingVertex -¿ DPOP, MGM, MaxSum Vertex class diagram!!!

3.5.4 Dynamics Controller

how does it work, what can it do

- Registration of vertices - change constraints - change variables - change domain
- Initial design consideration was to make it part of the graph as signal collect allows mixup of different types. Created weird problems because I need to pause the vertex for interval changes. not part of the graph but connected to vertices -¿ because it was a problem with convergence and there was a problem with thread sleep in akka

3.6 Monitoring Platform

It was decided to find a more sophisticated method to monitor the utility, quality, conflicts and stats of the calculations than to write to a logfile. Partially because File IO can possibiliy be limited in the number of open files and processing results to fit the log file format during calculations can affect the performance of the algorithm one tries to

benchmark. Mainly because processing log files was to tedious and visibility was not given to the calculations.

Sending the results with non-blockin http requests to a restful server was considered to be an alternative and the Play Framework[4] has been chosen because it is highly scalable, is able to handle thousands of simultanous connections, lightweight, non-blocking and allows to process results on-the-fly with code in Java or Scala. It was also chosen because Akka is tightly integrated and the actors concept are an integral part of the framework. It further allows to preview the calculation of the graph in realtime with websockets, which can be helpful during the implementation of algorithms.

By using an actor based approach it is possible to contain run specific data in a closed entity and store when limits are reached. Through the chosen architecture one can run multiple tests in parallel without problems. The server is of course also limited by it's ressources.

For asynchronous http request the Dispatch[5] library has been used.

[4]https://www.playframework.com
[5]http://dispatch.databinder.net

4

Benchmark & Discussion

In this chapter the three algorithms are benchmarked with different parameters and scenarios. The first series of tests is conducted in a static environment for a basic understanding of the algorithm performance and the second series is held in various dynamic settings to explore the dynamic capabilities of the algorithms. The testing environment was the minion cluster of the departement of informatics at the University of Zurich. The cluster consists of 16 machines and each machine has 128 GB RAM and two E5-2680 v2 at 2.80GHz processors. Every processor has 10 cores and the interlink between the machines is a 40Gbps Infiniband setup. The cluster has different partition speeds (slow, fast, superfast). All tests were conducted on superfast partitions.

4.1 Results I: Algorithms Performance in Static Environments

- Problem Density - Run Mode on Signal Collect
theoretical foundation: chapman benchmarking hybrid algorithms,!!! table, stuff

4.1.1 Solution Quality over Time

- Utility, Quality - Verlauf alle drei - Scalability: one, multiple machines - Quality Distribution

4.1.2 Time to Convergence

- More time to find a valid solution -¿ could be local optima - Scalability: one, multiple machines - Convergence Distribution: how often does it converge -¿ dpop should always converge

4.1.3 Conflicts over Time

- Verlauf alle drei - Scalability: one, multiple machines

4.1.4 Number of Messages

- Verlauf alle drei - Scalability: one, multiple machines - Message Distribution

4.2 Results II: Algorithms Performance in Dynamic Environments

petcu 2007

- Additional parameter - Different Scenarios: Constraints, Variables, Domain - Change one, multiple - Amount of Change: Percentage, Number

- Number of Agents: 10,100,1000 - Number of Meetings; 10,100, 1000 - Number of Timeslots: 1000

constraints: mode, interval, percentage variables: mode, interval, new Neighbourhood, new Agent, add / remove, how many domain: mode, interval, percentage, increase / decrease

4.2.1 Rebound Time

- utility, quality - Single change - Distribution - Verlauf alle drei - given timeframe -¿ lowest to highest avg.

4.2.2 Stability

[?]

- Utility, Quality - Multiple change: Peak Average - Median Utility

- Rate / Avg. Conflicts - Density!! Mailler2014,

- messages

5

Limitations & Future Work

The main limiting factor in the work was the time to test more algorithms and more problems and more aspects of dynamic constraint optimization problems.
It would be interesting to see how the performance of other problems is.
The meeting scheduling problem could also be further explored by increasing the amount of maximum meeting participations of an agent. The amount of participations was limited during the course of this thesis to not further expand the number of possible cases for benchmarking.
The aspect of changing domains was also not much researched despite being in the framework as a possible test. This was mostly because of the lack of impact respectively damaging impact changed timeslots had on the scheduling problem. One could further investigate this property of dynamic constraint optimization problems. [Yedidsion and Zivan,]
-¿ have explored this
The aspect of changing functions

6

Conclusions

A complete (DPOP), a local-iterative message-passing (MaxSum) and a local-iterative approximate best-response (MGM) algorithm have been implemented. The implementation on top of Signal/Collect was a good fit from a structural point of view.

The meeting scheduling problem has been mapped as a distributed constraint optimization problem, which also was a good fit.

The framework has proven to be a good starting point to dynamically change problems during runtime.

The monitoring platform has proven to be very helpful in the process of implementation as well as in the evaluation.

The results of test have shown that...

Benchmarks that made sense where...

References

[, 2005] (2005). Dankwoord. *Sociologie*, 1(4):494–494.

[Chapman et al., 2010] Chapman, A. C., Rogers, A., and Jennings, N. R. (2010). Benchmarking hybrid algorithms for distributed constraint optimisation games. *Autonomous Agents and Multi-Agent Systems*, 22(3):385–414.

[Chapman et al., 2011] Chapman, A. C., Rogers, A., Jennings, N. R., and Leslie, D. S. (2011). *A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems*, volume 26.

[Farinelli et al.,] Farinelli, A., Vinyals, M., Rogers, A., and Jennings, N. R. Chapter 12 Distributed Constraint Handling and Optimization. pages 1–43.

[Franzin et al.,] Franzin, M. S., Freuder, E. C., Rossi, F., and Wallace, R. Multi-agent meeting scheduling with preferences : efficiency , privacy loss , and solution quality.

[Grubshtein et al.,] Grubshtein, A., Gershman, A., and Meisels, A. Scheduling Meetings by Agents. *Citeseer*, (1):1–15.

[Hiroaki et al., 1999] Hiroaki, K., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). RoboCup Rescue : Search and Rescue in Large-scale Disasters as a Domain for Autonomous Agents Research and Robotics Can Make. pages 739–743.

[Leeuwen et al., 2002] Leeuwen, P. V., Hesselink, H., and Rohling, J. (2002). Scheduling Aircraft Using Constraint Satisfaction. 76.

[Maheswaran et al., 2006] Maheswaran, R., Pearce, J., and Tambe, M. (2006). A family of graphical-game-based algorithms for distributed constraint optimization problems. *Coordination of large-scale ...*, pages 1–19.

[Maheswaran and Tambe, 2004] Maheswaran, R. and Tambe, M. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. *Proceedings of the ....*

[Mailler and Zheng, 2014] Mailler, R. and Zheng, H. (2014). A new analysis method for dynamic distributed constraint satisfaction. *Proceedings of the 2014 international conference ...*, pages 901–908.

[Modi et al., 2005] Modi, P., Shen, W., Tambe, M., and Yokoo, M. (2005). Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180.

[Nguyen and Yao, 2012] Nguyen, T. T. and Yao, X. (2012). Optimization - The Challenges. 16(6):769–786.

[Petcu and Faltings, 2003] Petcu, A. and Faltings, B. (2003). A Scalable Method for Multiagent Constraint Optimization.

[Petcu and Faltings, 2007] Petcu, A. and Faltings, B. (2007). Optimal solution stability in dynamic, distributed constraint optimization. *Proceedings of the 2007 IEEE/WIC/ACM ....*

[Stutz et al., 2010] Stutz, P., Bernstein, A., and Cohen, W. (2010). Signal/collect: graph algorithms for the (semantic) web. *The Semantic Web-ISWC 2010*.

[Yedidsion and Zivan,] Yedidsion, H. and Zivan, R. Applying MaxSum to DCOP MST.

[Zhang et al., 2002] Zhang, W., Xing, Z., and Louis, S. (2002). Distributed Breakout vs . Distributed Stochastic : A Comparative Evaluation on Scan Scheduling. *Proceedings of the AAMAS-02 workshop on Distributed Constraint Reasoning*, pages 192–201.

A

Appendix 15

A.1 Results I: Selected Data

A.2 Results II: Selected Data

List of Figures

List of Tables