University of Zurich[UZH]

Dynamic Distributed Constraint Optimization in Signal/Collect

Thesis January 15, 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

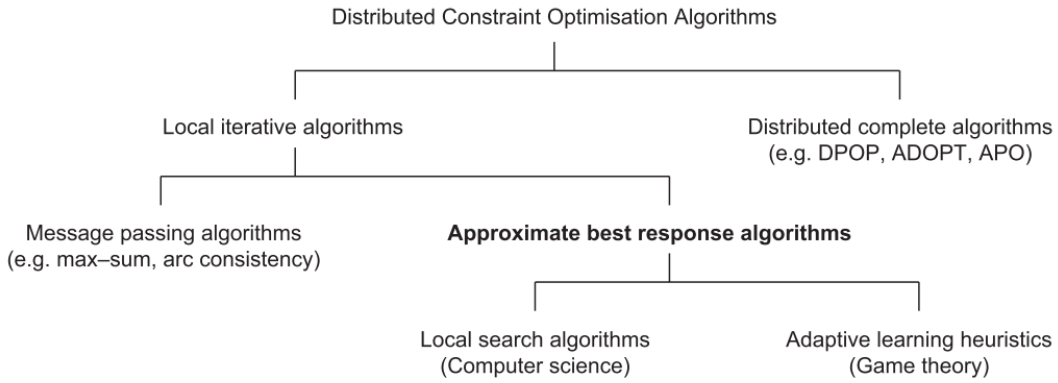

Figure 2.1: Categorization of DCO algorithms [Chapman et al., 2011]

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). 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 Distributed Complete

Distributed complete algorithms always discover a configuration of value assignments to variables that maximizes the global utility function. This completness guarantee increases the complexity of computation and leads to exponentially growing message numbers or calculations when increasing the amount of variables in a problem. Messages between agents contain often complex structures and constraint problems usually need to be transformed to a extensive graph structure [Chapman et al., 2011]. These types of algorithms therefore are not expected to scale well and quickly find qualitative solutions, but they fit well if one wants to find the maximal utility of a problem.

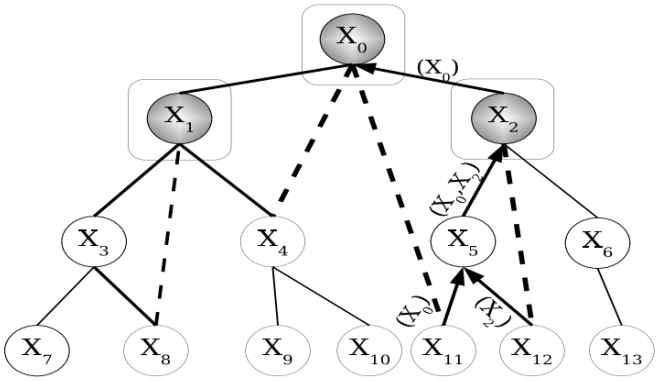

Figure 2.2: Pseudotree for DPOP [Petcu and Faltings, 2003]

For this thesis, it was decided to implement the Dynamic Programming OPptimization algorithm (DPOP) proposed by [Petcu and Faltings, 2003] as a comparison to the local-iterative approaches. In this algorithm, constraint optimization problems need to be converted to a pseudotree (Fig. 2.2), which is an modification of a DFS Tree. The original DCOP graph is transformed in a way that previous neighbours are placed in the same branches of a binary tree. They are connected trough ordinary tree edges, but additionaly back-edges between unconnected previous neighbours are established. The leaf nodes propose UTIL messages with their utility values for each value assignment upwards the tree and the Root node sends a VALUE message downwards containing the best value to chose. Nodes in the middle of the tree propagate UTIL and VALUE messages. The message structure is fairly complex as it involves all the pseudoparents from the back-edges and their context to a hypercube, which increases the message size exponentially. The number of messages on the other hand is linear. Through the integration of contextual values, privacy is not guaranteed in this setting.

2.3.2 Local-Iterative - Best Response

In a local-iterative best-response algorithm, agents only communicate their current state, e.g. their value assignment and react to these value messages in the best possible way from their perspective. The agents are only connected to their neighbours with whom they share constraints and there is no complex graph structure controlling the message flow [Chapman et al., 2011]. Through this local property, the types of algorithms should be inherently scalable as the messages and computations do not increase exponentially. Further, this approach is optimal from a privacy perspective as the neighbours only share their current preference and no other details of their schedule.

For this thesis, it was decided to implement the Maximum-Gain Messaging algorithm (MGM) proposed by [**?**]. In this algorithm, agents calculate the maximal gain in utility they can achieve when assigning to another value and send this value as a message. If of all messages they received, they have the highest gain, the value is changed. Otherwise

the local value stays the same. This algorithm fullfills the anytime property, i.e it can provide a solution at every timepoint during calculation and also reaches good solutions quickly [Chapman et al., 2010]. As the decision of an agent depends on a complete set of message of all it's neighbours, this algorithm will not perform well in asynchronous running mode. This type of algorithm does further not always converge and does not always provide the optimal solution to a problem.

2.3.3 Local-Iterative - Message Passing

The difference of message-passing to best-response algorithms is that the agents send and receive messages containing a specific data structure, which contains the utilities respectively costs that various assignments hold for a local variable. Received messages are used to calculate the next message that is sent to the connected neighbours. These types of algorithms are like best-response algorithms able to provide an acceptable solution in a short period of time, but also share the charasteric to sometimes not converge or not providing an optimal solution [Chapman et al., 2011].

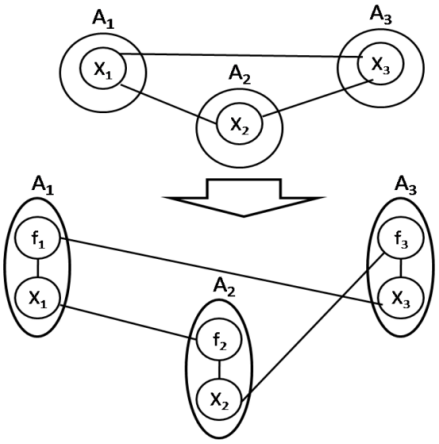

Figure 2.3: Conversion of a general DCOP to a factor graph[**?**]

For this thesis, it was decided to implement the MaxSum algorithm introduced by [**?**]. The algorithm has currently gotten a lot of attention from researchers. For this work, the algorithm is especially interesting because of it's proposed abilities in at least synchronously run dynamic environments. [**?**] wrote in their paper:

> [...] we note that if messages are continuously propagated, and the states of the agents are continuously updated, then the algorithm may be applied to dynamic problems where the interactions between agents, or the utilities resulting from these interactions, may change at any time.

In MaxSum the original DCOP graph is transformed to a factor, which is a form of a bipartite graph and of cyclic nature (Fig. 2.3. An agent contains after the transformation

of a variable and a function node, whereas variables are connected to all corresponding function nodes of their previous neighbours. The function nodes are vice versa connected to all previous neighbours of it's variable node. The messages sent from variable nodes differ from the function nodes. A message from variable to function contains for every value $d \in D_x$ the sum of utilities regarding this value, which the node has received from all connected function nodes. It is important to note that this sum does not include the values provided by the message target. The values are normalized at this point to avoid steadily to infinity increasing sums. A message from a function node to a variable node holds for every value $d \in D_x$ the summation of all costs received from all connected variable nodes except the message receiver and the original cost of the constraint represented by the function node [**?**].

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*

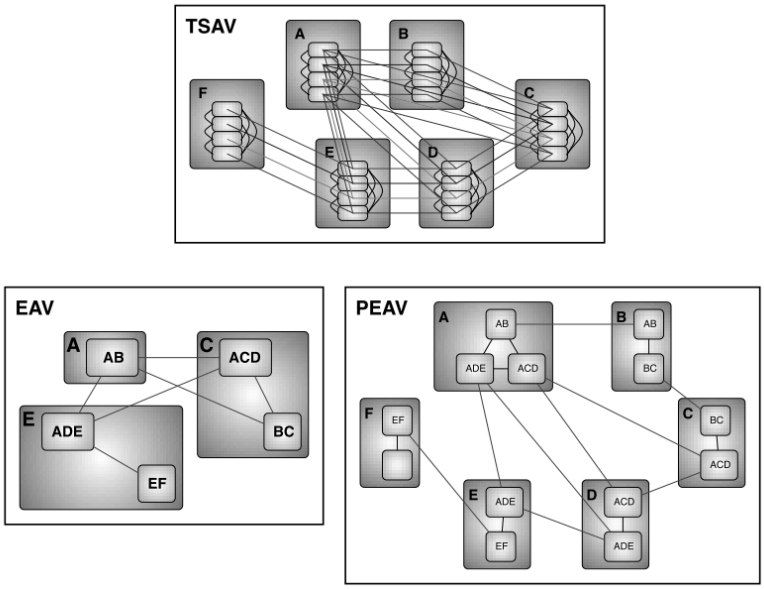

Figure 3.1: Different paradigms of mapping the meeting scheduling Problem [Maheswaran and Tambe, 2004].

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., , **?**].

A local utility function $u_l(s)$ would therefore include the sum of all soft constraints multiplied by the product of the hard constraints analogous to the global function defined in chapter 2. These definitions and this form of a local utility function have been implemented at later stages of the thesis into the algorithms:

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

3.1.2 Problem Dataset Generation

During the course of the thesis, it was necessary to find a dataset for the benchmarking. The Frodo2[1] framework or for example the the dataset from AAMAS, 2004 [2] do 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, as well as change constraints dynamically it was decided to generate meeting participations and agent schedules randomly. It was also chosen to limit the number of meeting participations per agent to keep the benchmarks more realistic analogously to [**?**].

- Schedules are blocked based on the percentage given trough the density parameter
- Preferences for meetings are chosen randomly from free timeslots in the schedule
- The number of meeting participations is chosen randomly from 1-5
- The meeting participations are chosen at random

3.2 Framework

3.2.1 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][3], which is built on top of Akka[4] and written in Scala[5]. 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

[1] http://frodo2.sourceforge.net
[2] http://teamcore.usc.edu/dcop/
[3] http://uzh.github.io/signal-collect
[4] http://akka.io
[5] http://www.scala-lang.org

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.

3.2.2 Structure & Functionality

The considerations for the structure of the framework were that I needs to be hierarchical to allow different constraint optimization problems to be run. This mainly concerns the Vertices. For a basic convergence function and control parameters on the number of sent signals a `BasicVertex` has been implement. Further a `MeetingSchedulingVertex` has been implemented. This Vertex acts a generalization

It was decided to have a variable represent each meeting participation of a user and connect every of those variables of one agent to the agent vector, where meeting times are registered. This agent vector acts as an difference hard constraint between the participations of a user and is not a Signal / Collect edge, but a shared reference. Further, all variables attending a meeting also share a reference to the meeting vector where every agent shares his current preference which is in essence a equality hard constraint. The local utility function is based on the definitions in chapter 2.2. The reason why the utility function has been generalized on this level is because the structure has repeated itself in all three implemented algorithm approaches. It is further helpful for comparison to have the exact same utility function implemented.

that includes a convergence function for meeting scheduling, the local utility function and data storage functions. To generalize dynamic change functions, a `DynamicVertex` was created, which implements methods for changing constraints and the value domain of the vertex. The concrete vertex implementations from the three approaches extends the `DynamicVertex`.

3.2.3 Dynamics Controller

The initial design consideration to introduce change to the constraints in the agents was to create a special vertex as part of the graph in Signal/Collect because the framework supports multiple types of vertices and messages in one graph execution. However, the vertex needed to be paused in the case of interval changes for a certain time and this caused errors during execution. Akka distributes multiple actors to the same thread and through pausing the dynamics vertex, other vertices were blocked. Through the abstractions in the Signal/Collect framework, it was not possible to gain access to the Akka dispatcher to schedule the activation of the vertex as a possible solution. It therefore was decided to run the changes controller in it's own thread alongside the graph execution.

The controller is first of all able to change constraints at a given interval and percentage (of all constraints in the problem). It is also possible to run a change only once. In

the case of the meeting scheduling problem implementation this is only related to soft constraints as hard constraints can be handled by adding or removing variables with the second function. The variable change function also can be run at a certain interval or at one timepoint. One can add parameters to create a new neighbourhood or use existing relationships and add new variables or remove existing one's. Instead of given by percentage, this change is defined by number. As a third function, the controller does change the domain in the whole problem for all agents. In the use case of meeting scheduling, this increases or reduces the available timeslots.

3.2.4 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.

3.3 Mapping of DPOP

The following sections describes the mapping of the graph structure of the DPOP algorithm and the behaviour of it's nodes to the Signal/Collect Framework in regards to the meeting scheduling problem. The implementation is based on [Petcu and Faltings, 2003].

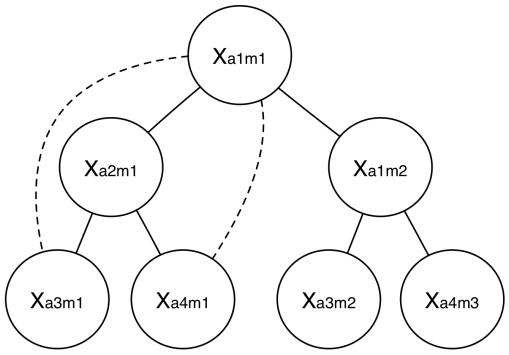

Figure 3.2: An example DPOP pseudograph

hard constraint connection not show in image cycles A pseudo-tree arrangement of a

graph G is a rooted tree with the same vertices as G and the property that adjacent vertices from the original graph fall in the same branch of the tree (e.g. X0 and X11 in Figure 1). Petcu2003

The process of the algorithm in short

Algorithm 1 DPOP Util Message Handler

```
1: DPOP (X, D, R)
   Each agent Xi executes:
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(delta1(string(i)), delta2(j)).
14: goto top.
```

Algorithm 2 DPOP Value Message Handler

```
1: DPOP (X, D, R)
   Each agent Xi executes:
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(delta1(string(i)), delta2(j)).
14: goto top.
```

Signal: Collect:

3.4 Mapping of MGM

The following section describes the mapping of the graph structure of the Maximum-Gain Messaging algorithm and the behaviour of it's nodes described in chapter 2 to the Signal/Collect Framework in regards to the meeting scheduling problem. The implementation is based on [Chapman et al., 2010]

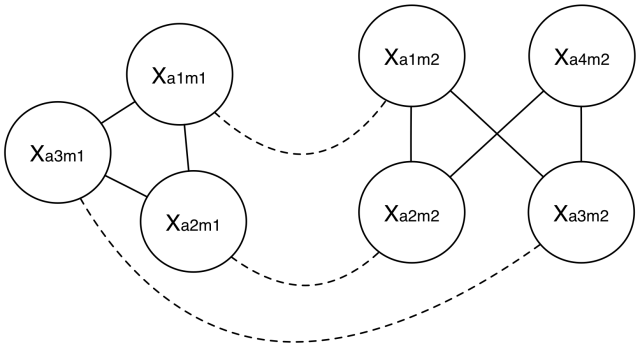

Figure 3.3: An example MGM graph

Mapping the meeting scheduling problem to the structure of MGM was quite straightforward. The participant vertices of one meeting are all connected to each other through a Signal / Collect edge.

Algorithm 3 MGM Pseudocode

currentReward = u(s = currentState, S_v)
for **do** $k = 1 : K$
$jstateGain(k) = u(s = k, sv) - currentREward$

In the signal
Collect:
and send each other value messages.

3.5 Mapping of MaxSum

The following sections describe the mapping of the graph structure of and the behaviour of it's nodes described in chapter 2 to the Signal/Collect Framework in regards to the meeting scheduling problem.

– describe tests and thinking behind the chosen structure!!!! -¿ bipartite graph -¿ unary connections in factor graph should be possible - had a problem with the property to send only to others with only 1 meeting.

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

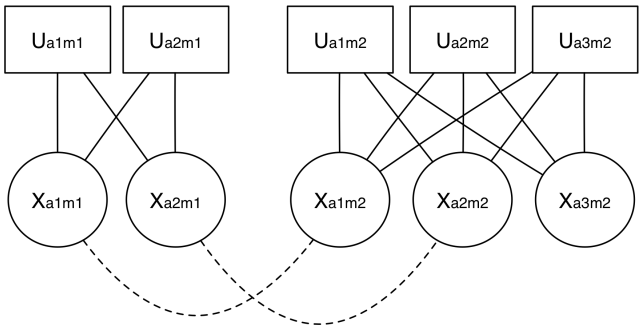

Figure 3.4: An example MaxSum graph

Algorithm 4 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.
```

Signal: Collect:

3.6 Monitoring Platform

The storage and monitoring of the utilities, quality levels, conflicts and run statistics of the calculations is in bechmarks usually done by writing the results to a log file. It was decided to use an alternative method during the course of this thesis. Mainly because it was a desirable function to automatically postprocess the results of the bechmarks on a detached machine and because real-time visibility was considered to be useful during the implementation of the algorithms.
Sending the results with non-blocking asynchronous HTTP requests[6] to a restful API on a dedicated server was considered to be a viable option and worth trying out. The Play

[6]http://dispatch.databinder.net

Framework[7] has been chosen for implementation because it is highly scalable and able to handle thousands of simultanous connections, is lightweight as well as non-blocking and allows to process results on-the-fly with code written in Java or Scala. It was also chosen because the Akka framework, which is also the foundation of Signal/Collect is tightly integrated and the actors concept is an integral part of the platform. For every benchmarking run, an actor is created that handles all the relevant incoming messages. It is therefore possible to run multiple benchmarks in parallel. The framework further allows to visualize the global utility of the graph in real-time via websockets.

[7] https://www.playframework.com

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

4.2 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