Abstract

The field of multi-agent systems is a field of research with increasing interest. In a multi-agent system, autonomous agents interact in a shared environment to attain personal interests and/or to complete common goals. Distributed Constraint Optimization Problems (DCOP) are a framework that models the agents autonomous behaviour, coordination and communication according to the problem's structure.

Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems (CSP) form the foundation of the Distributed Constraint Optimization Problems (DCOP). CSP are decision problems that deal with setting values to variables, under a set of specified constraints on how variable values should associate with each other. CSP is a popular method to model complex combinatorial problems.
A CSP is a tuple (X, D, C) where:

- $X = \{x_1,\dots, x_n\}$ is a finite set of variables.
- $D = \{D_1,\dots, D_n\}$ is a set of finite domains for the variables in X with D_i being the set of possible values for the variable x_i.
- C is a finite set of constraints over subsets of X.

A CSP is formulated by a set of variables x_1, x_2,..., x_n and a set of constraints, c_1, c_2,...., c_m. Every variable x_i has a domain D_i of possible values. Each constraint c_i includes some subset of the variables and describes the allowed combinations of values for that subset. By assigning values to some or all of the variables is how the CSP model is usually implemented. When an assignment does not violate any constraints is a **consistent assignment**. A partial assignment is a value assignment for a subset of variables from X that is consistent with their respective domains. An assignment where every variable is referenced is called a **complete assignment**. A complete assignment that satisfies all the constraints is a solution to a CSP.

Distributed Constraint Satisfaction Problems (DiCSP)

By applying the CSP framework among a set of autonomous agents the resulting model is called Distributed Constraint Satisfaction Problem (DiCSP). DiCSP is an extension of the CSP for multi-agent systems. A DisCSP is a tuple $\langle A, X, D, C, \alpha \rangle$, where X, D and C are the set of variables, their domains and the set of constraints as defined in a CSP and:

- $A = \{a_1, \dots, a_m\}$ is a finite set of autonomous agents.
- $\alpha = X \rightarrow A$ is a mapping function from variables to agents, which assigns the control of each variable $x \in X$ to an agent $\alpha(x)$.

In DiCSP systems, agents communicate with each other to find an assignment to all variables they control so as to satisfy all the problem constraints.

Distributed Constraint Optimization Problems (DCOP)

DiSCP generalizes to DCOP where constraints specify a form of preference over their violation, rather than a boolean satisfaction metric. DCOP are at the same time an extension of the COP framework for multi-agent systems. In a DCOP framework agents control variables and constraints and have to guide the assignment of the values for the variables they control in order to maximize a global goal function. A DCOP is a tuple (A, X, D, F, α), where:

- A = {a_1, ….., a_m} is a finite set of agents.
- X = {x_1,…., x_n} is a finite set of variables, with $n \geq m$.
- D = {D_1,....., D_n} is a set of finite domains for the variables in X with D_i being the domain of the variable x_i.
- F = {f_1,....., f_k} is a finite set of cost functions. The cost functions f_i are also called constraints, utility functions or reward functions.
- $\alpha = X \rightarrow A$ is a mapping from variables to agents, which assigns the control of each variable $x \in X$ to an agent $\alpha(x)$.

A partial assignment is a value assignment for a proper subset of variables of X. An assignment is complete if it assigns a value to each variable in X. A complete assignment is a solution of a DCOP if it optimizes all its cost functions.

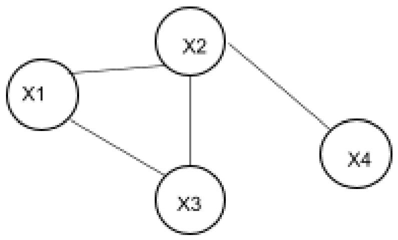
Distributed Pseudo-Tree Optimization Problem

DPOP is the extension of the bucket elimination algorithm, which belongs to the distributed constraint optimization problem (DCOP) algorithms and acts as an implementation method of that type of problems. Since DCOP establishes a specific format for the initialization of the problem, the DPOP algorithm extends these definitions by organizing its own features and specifications. It borrows the attributes of the multi-agent configuration i.e. the agents (a1,...,am), the variables X (X1,...,Xn), n>=m, the domain D of the variables (d1,...,dn) and the utility function U that denotes the **cost** table for every combination among the variables of each agent.
Its purpose is to allocate the agents in pseudo-tree structure in order to communicate with each other using messages that inform other agents of their choice of action, which leads to the optimization of their utility(cost) function and the total welfare of the system, taking constraint properties among the variables into account. The latter refers to the maximization of the aggregate of the utility values of each agent.
The algorithm realizes the above knowledge by separating the process in three discrete phases:

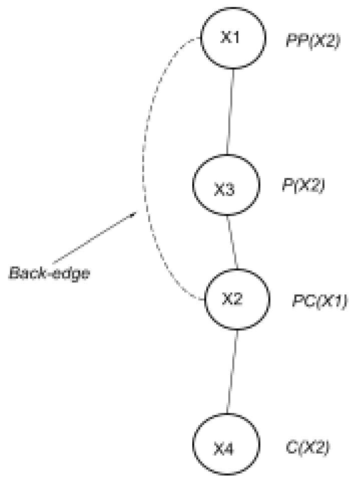
1) DFS-Tree construction
2) Utility propagation
3) Value propagation

DFS-Tree Construction

Initially, the variables are manipulated by the agents and are connected through a cyclic graph and its edges are the constraints among the variables and its nodes are the variables themselves. In the tree construction phase, DPOP starts with converting the graph into an acyclic structure by performing a depth first search(DFS) traversal of the graph and it serves as a communication means between the nodes. In order to retain the information of connection among the nodes that form cycles in the initial graph, the algorithm creates pseudo-edges - the back-edges- and they are virtual branches of the tree. Equivalently, the parents of each node X that has a back-edge to it, are called pseudo-parents PP(X) and the children nodes of a node X that are linked with a back-edge are called pseudo-children PC(X). The normal parents and children of a node X in the tree structure are denoted as P(X) and C(X) respectively. Thus, the resulting tree is called a pseudo-tree. The pseudo-connections can be visualized as links between two nodes which have a distance greater than one tree level.



Example of a variables cyclic graph



Pseudo tree

Utility Propagation

During this phase, the actual communication takes place through message exchanging among the nodes bottom-up. Since each agent has its own view of the layout of all the variable constraints, it knows which variables it controls and which agents it is connected with. The leaves of the pseudo tree start this process by locating their parents and pseudo-parents and create relationship tables for every allowed combination of the values of their domain defined by the constraint branch, one for each relative of the leaf. Each cell of the table contains a value computed by the utility(cost) function. Then the aggregate of the tables is calculated and it also represents the message it should be sent. It is important to be noted that the messages are sent only through normal edges of the tree and not through the back-edges.

Each step afterwards the nodes repeat the leaf process, with the addition of combining the messages they received from the children and calculating the optimal utilities for each node included in the received message and their parents. The operation is repeated until the root node is reached, keeping the optimal value for itself and informs its children for its choice, proceeding to the final phase, the value propagation.
The nodes start the message handling when they collect the messages from all their children. Let us denote |PP(X)| as the number of the pseudo-parents of the current node X and |MN| as the number of the nodes that are mentioned in the received message (if any). Each time the message produced by the current node X has dimensions equal to |PP(X)| + |MN| + 1, where 1 denotes the parent of X.

Value Propagation

This last step covers the settling of the value of each node's variable that comply with the corresponding constraint. When the root chooses the best value for itself, it then passes that value to all of its children. In their turn, the children consider the received option so as to select their best solution for the optimization aspect of the procedure, resuming this strategy top-down. This phase finishes when the leaves are reached again, so every variable has the value that maximizes the overall utility for the whole system of variables/agents.

Implementation Details

Problem Representation

The problem is described by a series of tuples <Agent, Meeting, Preference> which denote the wish of an agent A_i to participate in meeting M_i. Each agent has its own utility for each considered meeting. Multiple tuples can have the same meeting id, which they form a group meeting with multiple agents as participants. Furthermore, each agent has a priory preference function. There are 8 available time slots which an agent can assign a unique utility value and can form an overall preference of its meeting assignments. For example, an agent might prefer meetings to be held later in the day or earlier or it can be neutral. The tuple <Agent, Slot, Util> represents such behavior. We coded the DPOP implementation in Python using *networkx* package mostly for graph visualization.

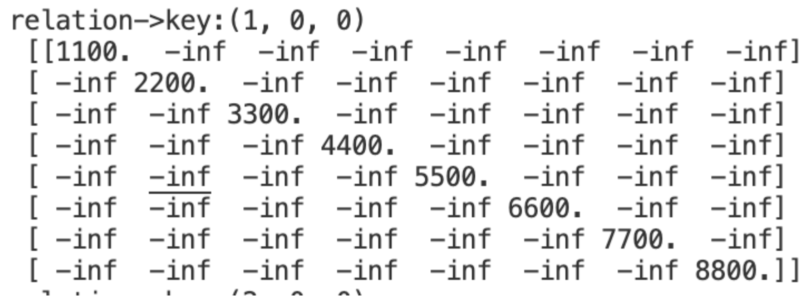
Before going into details we need to establish some conventions. As stated in the literature[1, section 7.2] pseudo-tree should be constructed having agents as nodes instead of the variables that they possess. This might be in contrast to the formal definition of DCOP problems, in which variables are preferred. Though, we argue that such an approach is inefficient in our case. Intra-agent messages are virtual and not transmitted over the network. Furthermore, our approach respects agent's privacy as "local" preferences, variables and meetings kept within its "territory".

Code Overview

A high overview of the algorithm execution is shown below.

- Read input file
- Create *Agent* Class and populate data structures that hold all information regarding: preference, meetings.
- Build constraint graph, each edge represents an Equality constraint. Connected nodes wish to attend the same meeting. We iterate through all meetings and agents which share a meeting id ; thus they will be connected together.
- DFS traversal to construct a DFS tree.
- All edges excluded during DFS traversal have been kept and included in the tree at later stages. Blue edge color indicates a back-edge while Black a DFS edge.
- The utility propagation starts by identifying all leaves of the tree. *send_util_msg* function is triggered with initial leave nodes. The function is then called recursively after all children have sent their UTIL_MSGS to respective parents. These parents are treated as the "new" leave nodes and the process repeats. *send_util_msg* terminates when it reaches root.
- During utility propagation, we keep track of the order to message passing, reverse it and start value propagation.

We represent a relation between two agents given a shared meeting id by two-dimensional matrix. Since inter-agent relation implies an equality constraint, agents need to assign the meeting at the same time slot. Thus, equality constraint is represented by a diagonal matrix which contains the sum of the two products of two agents, Preference X MeetingUtil. The rest of the matrix is filled with negative infinity. In Python, such relations can be represented as a dictionary whose key is the tuple <Child, Parent, Meeting> and value the diagonal matrix. Snippet below shows such implementation.

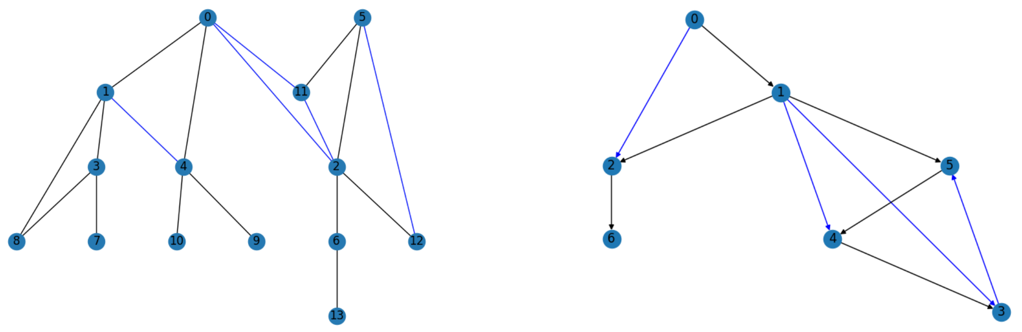


```
relation->key:(1, 0, 0)
 [[1100.  -inf  -inf  -inf  -inf  -inf  -inf  -inf]
 [ -inf 2200.  -inf  -inf  -inf  -inf  -inf  -inf]
 [ -inf  -inf 3300.  -inf  -inf  -inf  -inf  -inf]
 [ -inf  -inf  -inf 4400.  -inf  -inf  -inf  -inf]
 [ -inf  -inf  -inf  -inf 5500.  -inf  -inf  -inf]
 [ -inf  -inf  -inf  -inf  -inf 6600.  -inf  -inf]
 [ -inf  -inf  -inf  -inf  -inf  -inf 7700.  -inf]
 [ -inf  -inf  -inf  -inf  -inf  -inf  -inf 8800.]]
```

Relation between agent A1 and agent A0 in respect to meeting M0.

Code Assessment

In this section we will assess the functionality of our implementation. As stated above, DPOP algorithm consists of 3 distinct phases. We will start by the first step. Figures below show the output of Constraint Graph and Pseudo-tree which represent the problem. Here we have 10 agents, 5 meetings and 14 variables. On the left side we see the constraint graph and on the right the pseudo-tree after DFS traversal. Note that back-edges are presented as **blue** edges.

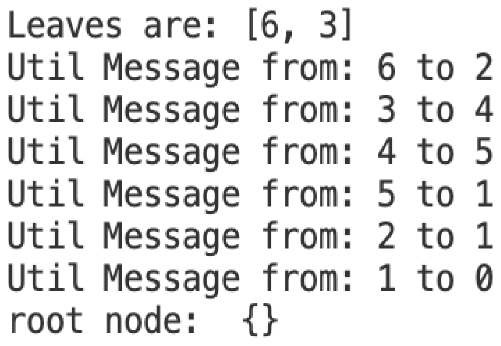


Example of constraint graph and pseudo tree.

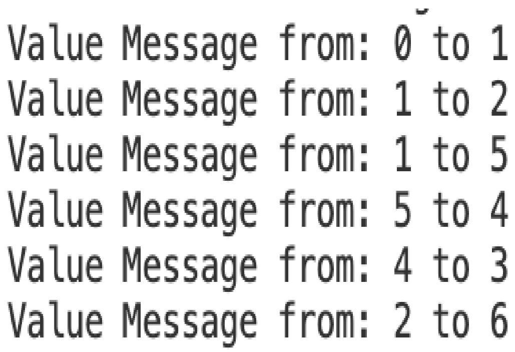
Arbitrarily, we chose the agent with id 0 to be the root of the pseudo-tree. Back-edges marked with blue color. For example, the clique comprised of nodes/agents { 0, 1, 2 } shows the wish of the three to participate in the same meeting, in this case meeting with id 0.

The second phase is the utility propagation. Util propagation relies on hierarchical order introduced by DFS construction steps. Given the example we presented above we will examine whether our implementation follows precisely this hierarchical order. It is clear that the leaves nodes are [3,6]. Thus they need to send the utility messages to their parents [2, 4]. Then node 4 continues the process by sending information to node 5. Nodes [2,5] transmit util messages to node 1 which finalizes the process by sending util to root node 0. The figure below shows the output of our algorithm which seems to respect the hierarchical order.

Utility and value propagation order.



```
Leaves are: [6, 3]
Util Message from: 6 to 2
Util Message from: 3 to 4
Util Message from: 4 to 5
Util Message from: 5 to 1
Util Message from: 2 to 1
Util Message from: 1 to 0
root node:  {}
```



```
Value Message from: 0 to 1
Value Message from: 1 to 2
Value Message from: 1 to 5
Value Message from: 5 to 4
Value Message from: 4 to 3
Value Message from: 2 to 6
```

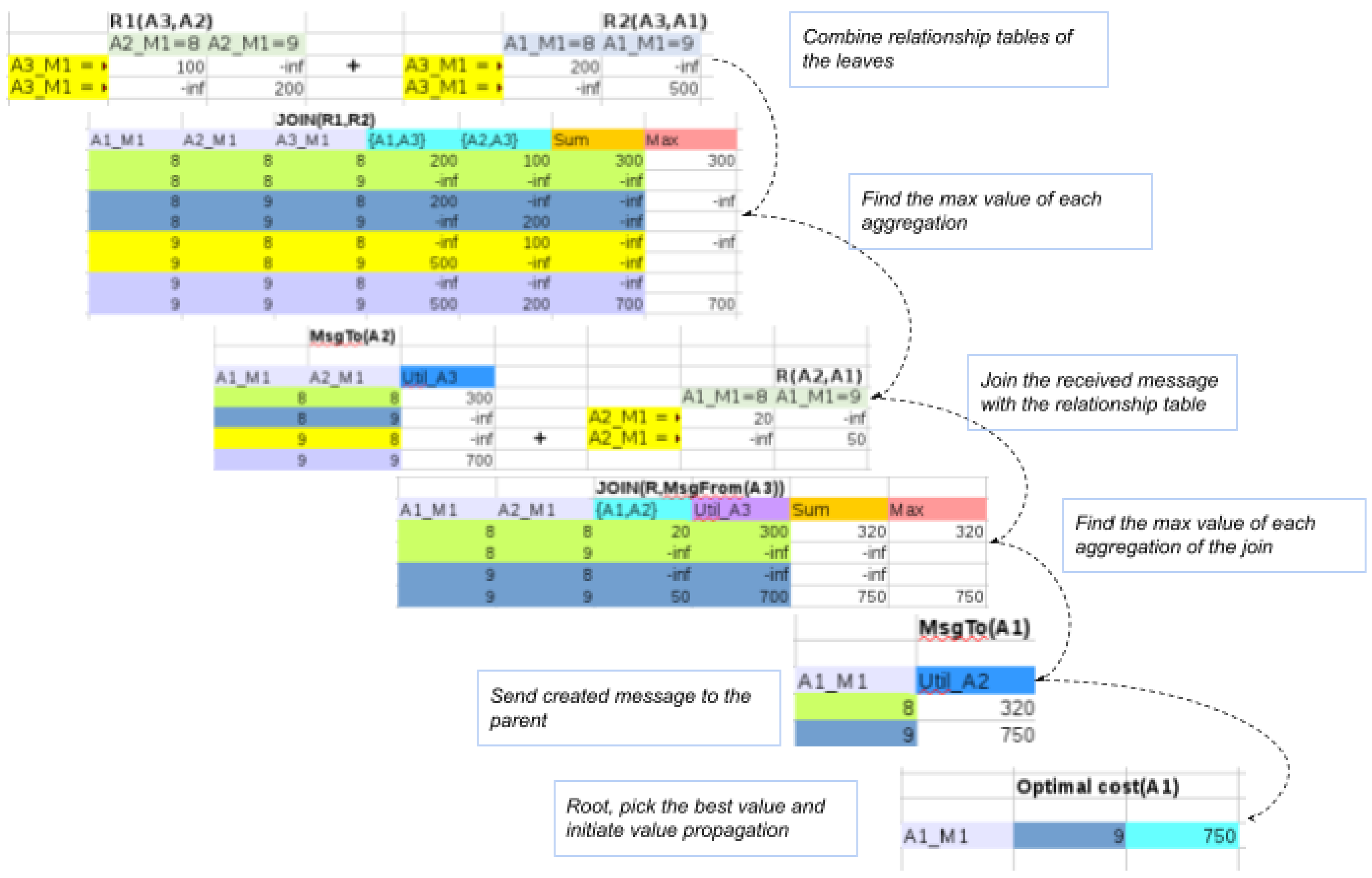
Similarly to utility propagation, value propagation is initiated essentially in reversed order. Output is shown on the right.

Drawbacks

As far as we are concerned the main drawback of our implementation lies in the way we calculate the utility messages. The biggest advantage, or tradeoff of DPOP algorithm, is the eagerness to reduce the number of transmitted messages among agents by allowing larger message sizes. In this way, the number of messages increases linearly during util phase though, the message size evolves in an exponential manner. Our implementation **fails to capture this as our hypercubes remain constant in size**.

Theoretical background utility message

Specifically, we kept the theoretical background behind the DPOP message propagation and formed the mechanism of how the messages would be created and traverse the pseudo-tree. The main idea is that each leaf agent creates relationship tables with every connection it has in the tree. The rows of the table act for the time slots for a particular meeting of the current agent node, while the columns are the time slots of the connected agent for that meeting. Furthermore, the cells accommodate the utilities awarded by summing up the products of the preference the agents have for their common meeting and the preference for the candidate time slot. Then the leaf agent optimizes his “best move” by joining its tables, adding their containing utilities together, for every possible combination of the time slots and keeping the maximum value from each combination with the connected agents’ time slots. Subsequently, the created message consists of a table that holds those maximum values, under the cells with the corresponding time slots.
The nodes that are not leaves receive this type of messages and save it for later use. When they receive a message from every child, the regular nodes inspect their parents and pseudo-parents, but this time they also examine the nodes that are referred to in the received messages. This means that the nodes determine the relationship tables with the mentioned nodes and then combine them together, combine the messages together and, finally, join the newly formed combinations in one uniform table, containing the sums of all possible time slot combinations. All the above mergings follow the same principles as the leaf node process. The operations for these joins can be pictured as: $(\oplus \{Msgs\}) \oplus (\oplus \{RelTables\})$, where $\oplus$ denotes the join/combination symbols described above, $Msgs$ stands for the obtained messages and $RelTables$ are the created relationship tables.
It can be observed that the current node is not included in the message sent to the parent, which infers that every next forwarded message does not increase in size during the passage of time, as well as due to the fact that the closer we are to the root, the shallower the tree becomes and, thus, less nodes appear in small tree levels. The example below depicts a simple process of message creation and propagation, performing the operations of summation, maximum finding and join:


Combine relationship tables of the leaves
Find the max value of each aggregation
Join the received message with the relationship table
Find the max value of each aggregation of the join
Send created message to the parent
Root, pick the best value and initiate value propagation
+
+

R1(A3,A2)

	A2_M1=8	A2_M1=9
A3_M1 =	100	-inf
A3_M1 =	-inf	200

R2(A3,A1)

	A1_M1=8	A1_M1=9
A3_M1 =	200	-inf
A3_M1 =	-inf	500

JOIN(R1,R2)

A1_M1	A2_M1	A3_M1	{A1,A3}	{A2,A3}	Sum	Max
8	8	8	200	100	300	300
8	8	9	-inf	-inf	-inf	
8	9	8	200	-inf	-inf	-inf
8	9	9	-inf	200	-inf	
9	8	8	-inf	100	-inf	-inf
9	8	9	500	-inf	-inf	
9	9	8	-inf	-inf	-inf	
9	9	9	500	200	700	700

MsgTo(A2)

A1_M1	A2_M1	Util_A3
8	8	300
8	9	-inf
9	8	-inf
9	9	700

R(A2,A1)

	A1_M1=8	A1_M1=9
A2_M1 =	20	-inf
A2_M1 =	-inf	50

JOIN(R,MsgFrom(A3))

A1_M1	A2_M1	{A1,A2}	Util_A3	Sum	Max
8	8	20	300	320	320
8	9	-inf	-inf	-inf	
9	8	-inf	-inf	-inf	
9	9	50	700	750	750

MsgTo(A1)

A1_M1	Util_A2
8	320
9	750

Optimal cost(A1)

A1_M1	9	750

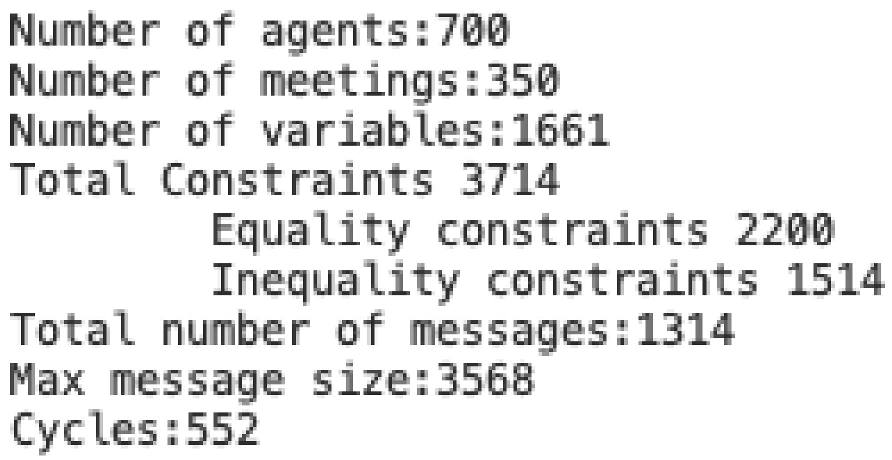
Example process of message creation and propagation

Results

We assess the performance of DPOP based on the few characteristics shown below. Cycle is the number of synchronous cycles which is two times the height of the pseudo-tree.

- Agents
- Meetings
- Variables
- Constraints: Equality constraints (dfs-edges) + Inequality constraints
- Messages
- Max message size:
- Cycles: the number of synchronous cycles is two times the height of the pseudo-tree

Output of Python implementation

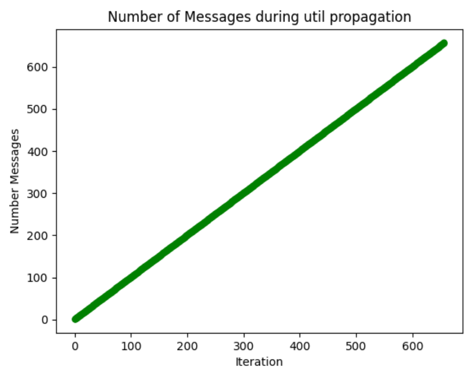


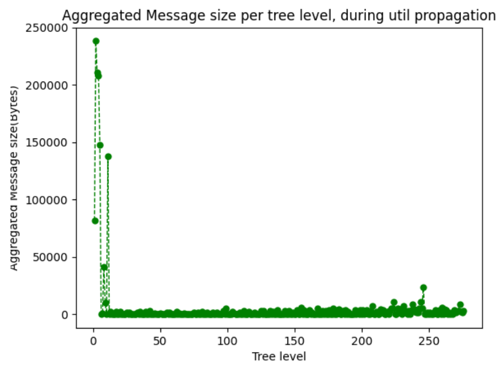
```
Number of agents:700
Number of meetings:350
Number of variables:1661
Total Constraints 3714
        Equality constraints 2200
        Inequality constraints 1514
Total number of messages:1314
Max message size:3568
Cycles:552
```

Agents	700
Meetings	350
Variables	1661
Constraints	2200 + 1514 = 3714
Messages	1314
Max message size*	**3568Bytes***
Cycles	552

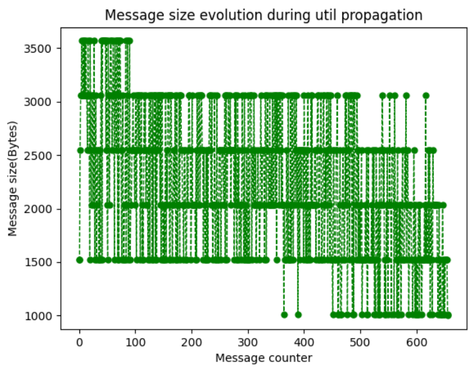
**Max message size is incorrect for reasons mentioned above*

It seems that our implementation follows the DPOP definition for linear number of messages.





Also, the aggregated size of messages as we go up in tree level seems to decay while message



size per utility calculation and transmission seems to take discrete values.

References

1. Andrian Petcu, Boi Faltings, DPOP: A Scalable Method for Multiagent Constraint Optimization, IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005
2. Ferdinando Fioretto, Enrico Pontelli, William Yeoh, Distributed Constraint Optimization Problems and Applications: A Survey, Journal of Artificial Intelligence Research 61 (2018) 623-698
3. Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham, Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling, 2004
4. Andrian Petcu, A class of algorithms for distributed constraint optimization, Frontiers in Artificial Intelligence and Applications, 2009