ASODPOP: Making Open DPOP Asynchronous

Student: Brammert Ottens
Supervisor: Boi Faltings
<first name>.<last name>@epfl.ch

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Artificial Intelligence Lab
CH-1015 Lausanne

Abstract. In this paper we show how ODPOP can be adapted to an asynchronous environment where agents might have to decide their values before the algorithm has ended, giving us Asynchronous ODPOP (ASODPOP). We have compared the algorithm with both ADOPT and distributed local search (DSA). Compared to ADOPT we show that our approach sends fewer messages, converges to a reasonable solution faster, and uses an equal amount of NCCCs. We also show that this convergence is much faster than local search, whilst the solution that local search converges to is far from optimal.

1 Introduction

In constraint satisfaction and optimization problems, variables are often controlled by different agents. We call such problems *distributed* constraint satisfaction problems (DisCSP) ([2]). For various reasons, such as the difficulty of problem formalization in a common framework, the lack of a recognized central authority, or the desire for privacy, it is desirable to solve such problems using distributed algorithms based on message exchanges among the agents.

Open DPOP (ODPOP) [8] is an adaptation of the well known DPOP algorithm [7] to the Open Constraint Programming paradigm (OCP)[3]. In ODPOP agents are first ordered using a DFS tree, and then solutions are aggregated in a bottom up, best first manner until the optimal solution has been found. As argued for below, ODPOP has certain disadvantages when used in a situation where agents need to act independently, i.e. asynchronously, and under possible time constraints. One could think of multi-agent coordination problems with a large number of agents, where response time is essential, or large sensor networks where the communication delay becomes significant. In such situations the agents do not always have time to wait for all the other agents and might have to make decisions on partial information.

In this paper we introduce Asynchronous ODPOP (ASODPOP), an extension of ODPOP to situations where agents work under time constraints and have to make decisions based on partial information

2 Concepts & Notation

Distributed Constraint Optimization(DCOP) [2] models the problem where a set of variables is distributed over a set of agents, with valued constraints between different vari-

Dagstuhl Seminar Proceedings 08461
Planning in Multiagent Systems
http://drops.dagstuhl.de/opus/volltexte/2009/1871

ables. Every agent is only aware of the constraints its variables participate in. The goal is to find a variable assignment that optimizes a global objective function, where this objective function consists of all the valued constraints between the different variables.

Definition 1 (A DCOP problem) *A discrete distributed constraint optimization problem is a tuple* $\langle \mathcal{X}, \mathcal{D}, C \rangle$ *where*

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ *is a finite set of variables.*
- $\mathcal{D} = \{d_1, \ldots, d_n\}$ *is a set of finite domains.*
- $C = \{c_1, \ldots, c_m\}$ *is a set of valued constraints where* $c_i : d_{i_1} \times \ldots d_{i_k} \rightarrow \mathbb{R}$.

To make the paper more readable, we assume that every agent owns exactly one variable and every constraint is a binary constraint. As a consequence, x_i can be used to both denote the variable and the agent that owns it.

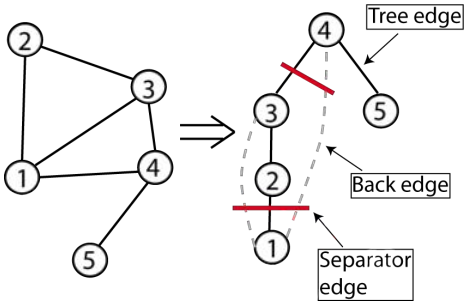

Fig. 1: A constraint graph and a rooted DFS tree based on it.

In order to efficiently find an optimal solution, agents must be prioritised in some way. An often used method is to create a DFS tree. A *DFS tree* is a rooted and directed spanning tree of the constraint graph such that two neighbours in the original graph are both in the same branch (see Fig. 1).

A *back edge* is an edge present in the constraint graph but not in the DFS tree. An agent connected via a back edge with an ancestor is called a *pseudo child* and the ancestor is called a *pseudo parent*. Furthermore, every agent x_i has a *separator* sep^i, consisting of its parent, its pseudo parents and its ancestors that are pseudo parents of its descendants. In other words, the separator contains all the agents that have an influence on the agents in the sub tree rooted at x_i. Finally, let $sep^i_+ = sep^i \cup \{x_i\}$.

If D^i_k contains the knowledge[1] agent x_i has of the domain of x_k, then an *assignment* s for agent x_i assigns to each $x_k \in sep^i_+$ an element from D^i_k, i.e. an assignment for agent x_i assigns a value to itself and all the variables in its separator. Ass^i is the set of all the assignments agent x_i knows about.

Definition 2 (Compatible) *Two assignments* s *and* s' *are said to be* compatible*, denoted by* $s \equiv s'$*, if they agree on the values of the variables they share.*

3 ASODPOP

In the DCOP literature, there are two types of algorithms. The search based algorithms, of which ADOPT [6] is a good example, use the DFS tree to perform a top-down search

[1] Because we follow the OCP paradigm, nodes can have incomplete information of the domains.

procedure, while the dynamical programming based approaches, like DPOP, aggregate solution from a bottom up manner and do not search.

ODPOP is a DPOP like algorithm and works in a bottom up manner. In stead of agents setting themselves to a value and notifying their children, in ODPOP agents ask their children what value they should set themselves to. Children respond in a best first manner with such assignments. An agent continues to ask for assignments until it has enough information to be able to suggest an optimal assignment to its parent, i.e. until the agent is *valuation sufficient*. Bounds on the utilities are used to recognise valuation sufficiency. When the root agent is valuation sufficient, it notifies its children on its decision. The problem with ODPOP in situations where agents are bounded by time is two fold. First, an agent's bounds for a particular assignment are not instantiated until it has received information on this assignment from all its children. Thus the speed with which this agent can make suggestions to its parent is limited by the slowest child. Second, the root agent chooses an assignment only when he is absolutely sure about the optimality of this assignment. Hence, its descendants are not able to assign a value to themselves until the root node has.

We propose to address these problems by allowing nodes to respond to their parents using incomplete information, by allowing agents to make estimations for the values it has not received any information about yet and by allowing the root node to make a "temporary" decision before it is certain of the optimality of its assignment. These adjustments result in the ASODPOP algorithm introduced below.

The algorithm consists of three different phases. In the first phase a DFS tree is generated, see [8] for more information on the distributed DFS algorithm. The second and third phase, respectively the ASK/GOOD phase and the VALUE propagation phase run concurrently, where the former aggregates solutions in a bottom up manner and the latter distributes decisions top down. These two phases are discussed in greater detail.

3.1 ASK/GOOD phase

As in ODPOP, nodes ask their children what values are best via ASK messages. The children respond to this with a *good*. Such a good contains an assignment s to all the variables in the agents separator (so not for the agent itself), the combined utility v the agents in the sub tree rooted at the child can obtain when the assignment is chosen and a binary variable b. b is used to distinguish between *true goods* ($b = true$) and *false goods* ($b = false$), where a true good is based on complete information, while a false good is based on incomplete information.

In order to find the optimal value, these goods should be sent in a best first manner. In ODPOP this can be done by ensuring that the leaf nodes do this. However, in ASODPOP certain goods have a utility based on partial information and estimates, which cannot be used to determine a best first order. To still ensure optimality, the best first requirement is limited to *true* goods. A good is a *true* good if its utility is calculated from received true goods and if at the moment it is sent it is the best good not yet sent. The purpose of the *false* goods is merely to propagate partial information and estimates upwards.

Algorithm 1: ASODPOP Agent

`Receive(ASK)`

if $\exists s_{max}$ **then**

  if $b^i(s_{max})$ *and* $\forall s' \in Ass^i \setminus sent\_goods \quad E^i(s_{max}) \geq UB^i(s')$ **then**

    `Send(`P^i, `GOOD(`$s^-_{max}, E_{max}, true$`))`;

    sent_goods $\leftarrow$sent_goods $\cup\{s | s \in Ass^i, s \equiv s^-_{max}\}$

  else

    `Send(`P^i, `GOOD(`$s^-_{max}, E_{max}, false$`))`;

    sent ASK to children;

`Receive(`*child,* `GOOD(`s, V, b`))`

If s contains new information concerning the separator or a variable domain, update Sep^i and Ass^i.

for *all* $s' \in Ass^i$ *such that* $s' \equiv s$ **do**

  $E^i_{child}(s') \leftarrow V$;

  $b^i_{child}(s') \leftarrow b$;

if b **then**

  Adjust estimates E^i_{child} for all $s' \in Sep^i$ with $s' \not\equiv s$ such that $E^i_{child}(s') < V$

When the agents assignment changes due to updated information, sent a VALUE message to its children;

When an agent x_i receives a good $\langle s, v, b\rangle$, it first updates its separator. After that it sets $E^i_j(s') = v$ and $b^i_j(s')$ for every assignment $s' \in Ass^i$ that is compatible[2] with s. If x_i has not yet received any information on an assignment $s' \in Ass^i$ from child j, it sets $b^i_j(s') = false$ and it either sets $E^i_j(s')$ to 0, or makes an estimate for its value. These estimates can be made according to a-priori information concerning the problem, using a pre-processing step that sets bounds on the utilities or, if no additional information is available, the estimates can be random.

The total utility for a specific assignment s is then obtained by summing up the utilities obtained from x_i's children; $E^i(s) = \sum_j E^i_j(s)$ and $b^i(s) = \bigwedge_j b^i_j(s)$.

When a node receives an ASK message (see Algorithm 1), it first determines $s_{max} \in Ass^i$, which is the assignment that currently has the highest utility and has not yet been sent as a true good and then sends $\langle s^-_{max}, E^i(s_{max}), b\rangle$, where s^-_{max} is restricted to sep^i and $s^-_{max} \equiv s_{max}$, i.e. s^-_{max} is obtained by dropping agent x_i's assignment from s_{max}. If s_{max} is based on true goods ($b^i(s_{max})$ = true) and its value is at least as high as the upper bound for all the assignments that have not been sent yet, it is sent as a true good. If not, it is sent as a false good. The upper bound for an assignment is calculated as

$$UB^i_j(t) = \begin{cases} E^i_j(t) \text{ if } b^i_j(t) = true \\ E^{last}_j \text{ if } E^{last}_j \text{ exists} \\ \infty \quad \text{otherwise} \end{cases} \qquad UB^i(t) = \sum_j UB^i_j(t)$$

[2] An agents separator can differ from its children's separators

where E_j^{last} is the utility of the last true good it has received from child j.

3.2 VALUE propagation phase

In the VALUE propagation phase information about the assignments agents choose is sent down the tree. The phase starts by the root node setting itself to s_{max} and announcing this to its children via a VALUE message. When an agent receives a VALUE message, it looks in sent_goods what value it should set itself to and reports this to its children via a VALUE message. Each time an agent's assignment changes due to the reception of a new good, its children are notified via a VALUE message.

4 Experimental evaluation

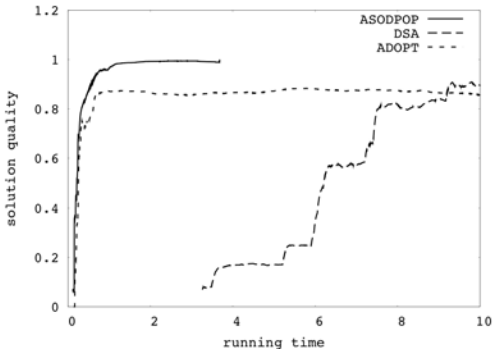

Fig. 2: Convergence performance on problems with 17 variables, averaged over 10 runs

ASODPOP is designed to operate in environments where the agents might have to make a decision concerning the assignment of their variables before the algorithm has ended. In such situations it is very important to have fast convergence to a (near) optimal solution. We are therefore interested in the convergence properties of ASODPOP compared to other asynchronous algorithms, in this case ADOPT [6] (with the DO2 pre-processing step [1]) and Asynchronous Distributed Local Search (DSA-C) [9]. We also looked at the number of messages that are sent and at the level of parallelism of the computation by measuring the number of non-concurrent constraint checks (NCCCs) [5].

In implementing ASODPOP we have made the simplifying assumption that all the agents do have full knowledge of their separator and the associated variable domains. So, the results on the amount of NCCCs must be seen as an upper bounds. The convergence and the number of messages used would have been the same.

We performed tests on meeting scheduling problems in an organisation with a hierarchical structure, using the PEAV model from [4]. Random meeting scheduling problems have been generated using 3, 10, 17 and 24 variables, 10 instances of each. Since we are dealing with optimisation problems, the tightness is of no concern. DCOP algorithms are meant to operate on low density problems, and hence the problems we generate have a low density as well. Finally, each variable has a cardinality of 5. The low number of 24 is due to the limitations of ADOPT. However, with ASODPOP we have been able to solve problems of up to a 100 variables in approximately 60 seconds. The estimates that are used in ASODPOP are generated randomly, i.e. no domain knowledge is used.

Due to space limitations, only the convergence results are shown. It is clear from Fig. 4 that ASODPOP converges must faster than ADOPT and reaches a better result than *DSA* in the same time. In terms of NCCCs and messages sent it also outperformed ADOPT by 2 orders of magnitude.

5 Conclusions

In this paper we introduced an asynchronous algorithm for Distributed Constraint Optimization, based on ODPOP. Where in ODPOP every agent waits until it has received enough information to determine its most preferred assignment, in the approach we have introduced the agents are allowed to make estimates concerning the assignments they do not yet have received enough information about. Such an algorithm can be very useful in situations where an agent might have to decide on the assignment of its variables before the algorithm has ended. In such a case fast convergence to a (near) optimal solution is very welcome.

The experiments performed show that ASODPOP exhibits such fast convergence. On top of that the actual optimum is found very fast, whereas both ADOPT and local search need more time to converge to a solution and the latter even fails to find the optimum. Future work should focus on finding more efficient implementations of ASODPOP. Furthermore, methods should be devised to get meaningful estimates for the assignments an agent does not have information about yet. This should improve the convergence results even more.

References

1. S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS '05*, pages 1041–1048, New York, NY, USA, 2005. ACM.
2. B. Faltings. *Distributed Constraint Programming*, pages 699–729. Foundations of Artificial Intelligence. Elsevier, 2006.
3. B. Faltings and S. Macho-Gonzalez. Open Constraint Programming. *Artificial Intelligence*, 161(1-2):181–208, January 2005.
4. R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *AAMAS '04*, pages 310–317, Washington, DC, USA, 2004. IEEE Computer Society.
5. A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing Performance of Distributed Constraints Processing Algorithms. In *DCR 2002*, 2002.
6. P. Modi, W. Shen, M. Tambe, and M. Yokoo. An Asynchronous Complete Method for Distributed Constraint Optimization. AAMAS'03, 2003.
7. A. Petcu and B. Faltings. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *IJCAI 05*, pages 266–271, Edinburgh, Scotland, Aug 2005.
8. A. Petcu and B. Faltings. O-DPOP: An algorithm for Open Distributed Constraint Optimization. In *AAAI-06*, pages 703–708, Boston, USA, July 2006.
9. W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks. *Artif. Intell.*, 161(1-2):55–87, 2005.