

Coordinating Agent Plans Through Distributed Constraint Optimization

Brammert Ottens and Boi Faltings

{brammert.ottens, boi.faltings}@epfl.ch
Artificial Intelligence Laboratory (LIA)
EPFL, Switzerland

Abstract

In this paper we show how the coordination of agent plans can be performed using Distributed Constraint Optimisation (DCOP) techniques. In particular, we show how a Truck Task Coordination problem can be modelled as a DCOP. We introduce a complete asynchronous DCOP algorithm, Asynchronous Open DPOP (ASODPOP), based on the DPOP algorithm that exhibits fast convergence to the optimal solution compared with both ADOPT and Distributed Stochastic Search (DSA). Fast convergence is useful when agents are time bounded and are thus unable to wait for an optimal solution.

Introduction

In every situation where multiple agents have to decide on a set of actions to perform, coordination is of the utmost importance. Not only do agents need to communicate with each other to obtain a feasible plan, they have to coordinate to obtain the best plan possible. The way such problems are modelled has a big influence on the efficiency with which they can be solved. If, for example, one only considers the joint actions of all the agents, the problem very quickly becomes intractable as the number of agents in a problem rises. Instead, one should make use of the fact that in most coordination problems agents decisions are directly influenced by only a small number of other agents. Furthermore, distributing the search for a solution over the set of agents allows one to make use of the parallelism available in most distributed problems.

One way to reap the benefits of both the locality of the interaction and the inherent parallelism is to model such problems as Distributed Constraint Optimisation Problems (DCOP) (Yokoo et al. 1992). In a DCOP every agent owns a set of variables it can set, while the reward of a particular agent depends not only on its own variables but also on the variables of certain other agents. The goal of all the agents is to maximise the combined reward of all the agents together.

In applying DCOP techniques to multi-agent planning, interaction can be modelled by variables that are constrained to take compatible values. One instantiation of such a multi-agent planning problem is the Truck Task Coordination (TTC) problem. In a TTC problem one has a set of

trucks, dispersed over an area, and a set of packets that needs to be picked up and delivered. Each truck needs to create its own plan but also needs to coordinate with other trucks in order to make sure that the global plan is both feasible and is of a certain quality. Each truck operates in a specific region that potentially overlaps with other regions. Each packet in such an overlapping area can be picked up by any of the trucks that cover it, and defines a coordination variable between different agents, making this problem very suited to be solved by using DCOP methods. Note that the agents only coordinate over which packet is picked up by whom. All the agents are free to plan the pick up and delivery sequence for the allocated packages as they see fit.

The goal of this paper is to investigate the usefulness of a particular DCOP algorithm, ASODPOP, when solving agent coordination problems.

Agent Coordination

When all the agents are cooperative agents, they are interested in coordinating their decisions so as to maximise the global reward of all the agents. In order to find these optimal decisions, the agents have to communicate about their preferences. However, agents are usually bounded by certain constraints on communication bandwidth, memory use but also on the time available to solve the problem.

One way of modelling such coordination problems is, to model them as a Distributed Constraint Optimisation Problem (DCOP). In a DCOP agents have to assign values to their variables, where their rewards depend on the assignments other agents make. These rewards are coded as constraints over combinations of values, and the agents solve the problem via message passing

The types of decisions agents can coordinate over can range from interpreting sensor data to attending meetings to coordination pick up and deliveries by several trucks. In this paper we shall focus on the latter, but keep in mind that the methods used can be applied on a much wider range of problems.

Solving a DCOP

During the past decade major progress has been made in solving DCOPs, where ADOPT (Modi et al. 2003) was the first algorithm that was able to optimally solve problems in a

distributed fashion. ADOPT operates by first prioritising the agents using a Depth First Search (DFS) tree. A DFS tree is a spanning tree of the constraint graph (or coordination graph) where all the branches are independent, i.e. neighbours in the constraint graph are in an ancestor-descendant relation in the DFS tree. It then performs a distributed depth first search by allowing the agents to set their variables in a top down manner. The disadvantage of this method is that the number of messages is exponential in the depth of the DFS tree and that it is not able to handle large domains.

Another approach to solving a DCOP is taken by DPOP (Petcu and Faltings 2005). It also operates on a DFS tree, but where ADOPT performs a top down search, DPOP aggregates solutions in a bottom up manner to the root agent and does not perform any search. The number of messages that is sent is linear in the number of agents, but the size of the messages is exponential in the induced width of the DFS tree¹, which is never greater and usually much smaller than the depth. Furthermore, large domains still pose a problem.

To tackle the deficiencies of DPOP, the ODPOP (Petcu and Faltings 2006) algorithm has been developed. Just as DPOP, ODPOP aggregates solutions in a bottom up manner, but with the difference that solutions are sent upwards in a best first manner, one at a time. The idea behind this approach is that, in general, agents do not need to have the full picture of their local problem to be able to decide on an optimal solution. ODPOP uses only a fraction of the messages used by ADOPT, it does not necessarily run into problems when the domains become large and the size of the messages grows only linearly in the induced width of the tree. The only disadvantage of ODPOP is that it is still a synchronous algorithm. It can receive messages in an asynchronous manner, but it only considers sending up a solution when it has received information on this solution from all its children. As a result agents higher up in the hierarchy have to wait for all their descendants before being able to make any decisions. When agents have time constraints on how long they can wait for an optimal answer, this can seriously degrade the performance of the algorithm. One would therefore want to have an algorithm that is able to aggregate partial information and would be able to base its decision on this partial information.

ASODPOP (Ottens and Faltings 2008) is an extension of ODPOP that gets rid of this last disadvantage. It does this by allowing partial information to be propagated upwards. Furthermore, when the problem allows it, agents can combine the partial information with estimates over the missing information and in this way speed up the process of finding a solution. Note that finding the optimal solution and proving optimality are two different steps in this approach.

ASODPOP

Just as most DCOP algorithms, ASODPOP prioritises agents using a Depth First Search (DFS) tree.

Definition 1 (DFS tree) Given a graph $G = \langle V, E \rangle$, a DFS tree on G is a directed spanning tree $G' = \langle V, E' \rangle$ where $E' \subseteq$

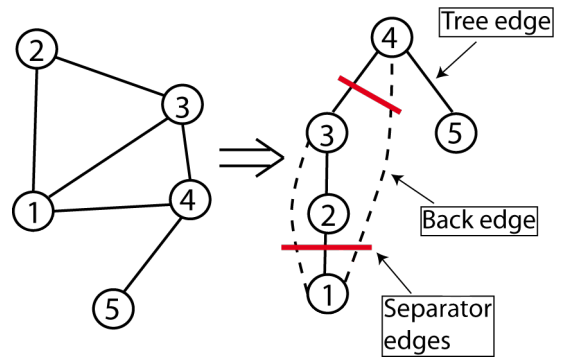


Figure 1: From a graph to a DFS tree

E such that all the branches of the tree are disconnected. That is, if $(a, b) \in E$ but $(a, b) \notin E'$, then a is an ancestor of b .

Figure 1 contains an example of a DFS tree. Edges shown as a solid line are *tree edges*, while edges shown as a dashed line are *back edges*. Each agent i has a separator sep^i that contains a minimal set of agents that need to be removed to completely separate the sub tree rooted at i from the rest of the tree. The *separator edges* are the edges that connect an agent with the agents in its separator. For example, the separator of agent 1 consists of agent 2, 3 and 4, while the separator of agent 5 contains only 4. The *induced width* of a tree is the size of the largest separator. This makes the tree of Figure 1 a tree of width 3.

To simplify the discussion, from here on we assume that all the agents own exactly one variable and that all the constraints are valued binary constraints². So, every agent i owns a variable x_i and $f(x_i, x_k)$ denotes a valued constraint over x_i and x_k , where the value denotes the utility for the particular combination of values. The goal is to find an assignment s such that

$$s = \operatorname{argmax}_{\{x_1, \dots, x_n\}} \sum_i \sum_k f(x_i, x_k) \quad (1)$$

When deciding upon an assignment, an agent will only have access to the variables in its separator. It has no information on what its descendants do. For example, when looking at the DFS tree in Figure 1, agent 2 will know the decision of both agent 3 and agent 4, but not the decision of agent 1. In order to be able to make the optimal decision given the decisions of 3 and 4, it has to know the influence of its decision on agent 1. This influence is measured in the utility that agent 1 can obtain when a certain decision is made, and agent 2 then chooses the assignment that maximises this utility.

To make this more formal, let i be an agent and let Ass^i be a set of assignments such that

$$Ass^i = \{x_{j_1} = v_{j_1}, \dots, x_{j_m} = v_{j_m} \mid x_{j_k} \in sep^i \cup \{x_i\}, v_{j_k} \in D_{j_k}, j_1 \neq \dots \neq j_m\} \quad (2)$$

²A valued binary constraint over two variables x and y gives a value to all the combinations of values of these variables.

¹A formal definition of a DFS tree is given in the next section

where D_{j_k} is the domain of x_{j_k} .

Definition 2 (Compatibility) Given two agents i and k , and two assignment $s \in Ass^i$ and $t \in Ass^k$. s and t are compatible, denoted by $s \equiv t$, if s and t agree over the assignments of their shared variables

In order to make an optimal decision, an agent i needs to know, for each $s \in Ass^i$, the maximal utility the tree rooted at i can obtain if s is used. We can assume that the agent is aware of its own private utility $own^i(s)$ for each $s \in Ass^i$. To stick with our example, this means that agent 2 needs to know the utility agent 1 can obtain for all the value combinations of x_2 , x_3 and x_4 .

Let $E^i(s)$ be the utility the tree rooted at i can obtain within the subtree when s is used. This utility is based on both the agents own utility and the utility of its children

$$E^i(s) = own^i(s) + \sum_c E_c^i(s) \quad (3)$$

where $E_c^i(s)$ is the utility child c can obtain when assignment s is used. Since an agent knows its own utility and is notified by its ancestors about their assignments, the only thing that is left to do is to determine the values for $E^i(s)$.

In DPOP, an agent's children aggregate the information concerning all their assignments in one message and send this upwards. ODPOP is based on the observation that in general an agent does not need information about all the utilities (Faltings and Macho-Gonzalez 2005), or even complete domain knowledge about all the variables in its separator. Instead only the best assignments will be part of the optimal combination. Thus, agents sequentially send utilities for assignments upwards in a best-first order and stop when the optimal solution is found. ODPOP significantly reduces the amount of information that needs to be exchanged to find the optimal solution, and is the best known algorithm in this respect. The problem with ODPOP now is that its agents only consider sending upwards an assignment if they have received information about this assignment from all their children. The difference between ODPOP and ASODPOP now is that agents combine partial information with estimates to propagate information sooner.

Since the root agent has no parents, its goal is simply to obtain enough information about its assignments to make an optimal choice. To do this, it sends ASK messages to its children. With an ASK message, an agent asks its children for new information concerning its assignments. When an agent receives an ASK message, it responds with a good $g = \langle s, u, b \rangle$, where s is an assignment, u is a utility and b a boolean variable. When $b = true$, g is a *true good*, while if $b = false$ g is a *false good*. The difference between true and false goods is that the false goods are used to aggregate partial information, while the true goods are based on complete information. Furthermore, the algorithm is designed in such a way that true goods are always sent in a best first manner.

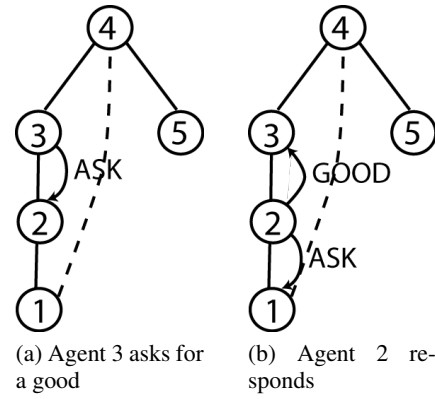


Figure 2: ASK/GOOD phase in ASODPOP

Example

Before we describe the algorithm in more detail, we first give a simple example of how the algorithm works. Consider the tree given in Figure 2 and let Table 1b represent the constraint between agent 2 and agent 3. Agent 1 is a leaf agent, and thus has complete information about the utilities for all possible value combinations in his separator. This means that he will always respond to an ASK message with a true good, i.e. a good with a utility based on complete information. We also assume that agent 1 responds to an ASK message in a best first manner with respect to the utility of the different goods.

We start when agent 3 sends an ASK message to agent 2, and assume that this is the first ASK message agent 2 receives. Since agent 2 has not received any information from agent 1, he is not aware of the fact that agent 4 is in his separator and hence all he knows is the information displayed in Table 1b.

Upon reception of the ASK message, agent 2 first determines the assignment, based on his current knowledge, that has the highest utility. In this case that is $s_1 = \{x_2 = a, x_3 = a\}$, which has a utility of 5. This utility is based on incomplete information (agent 2 does not know the utility of agent 1 for this assignment) and agent 2 thus responds to the ASK message by sending a false good containing the assignment s_1 to agent 3, as depicted in Figure 2.

Because agent 2's best assignment is based on incomplete information, he also sends an ASK message to agent 1. As a response to its ASK message, agent 2 receives the good $\langle \{x_2 = b, x_4 = t\}, 4, true \rangle$ from agent 1 (see Table 1a). Suppose it again receives an ASK message from agent 3. With the new information, this time $s_2 = \{x_2 = b, x_3 = d, x_4 = t\}$, with utility 7, is the assignment with the highest utility. Furthermore, it is based on complete information (true goods). However, because we assume that agent 1 responds with true goods in a best first manner, we know that the real utility for assignment $\{x_2 = b, x_3 = a, x_4 = t\}$ has an upper bound of 9. Hence, when sending assignment s_2 , agent 2 is not sure whether it is the next best assignment, and thus sends a false good. After two more ASK messages from agent 3 and goods from agent 1, the assignment with the highest

goods sent to parent	goods received
$\langle \{x_2 = a, x_3 = a\}, false, 5 \rangle$	$\langle \{x_2 = b, x_4 = t\}, 4, true \rangle$
$\langle \{x_2 = b, x_3 = d, x_4 = t\}, false, 7 \rangle$	$\langle \{x_2 = b, x_4 = r\}, 3, true \rangle$
$\langle \{x_2 = b, x_3 = d, x_4 = t\}, false, 7 \rangle$	$\langle \{x_2 = a, x_4 = t\}, 3, true \rangle$
$\langle \{x_2 = a, x_3 = a, x_4 = t\}, true, 8 \rangle$	\vdots
\vdots	\vdots

(a) The sequence of goods agent 2 receives from agent 1 (right) and the sequence of goods agent 2 sends to agent 3 (left)

$x_2 \setminus x_3$	a	b	c	d	e
a	5	5	1	5	1
b	2	0	1	3	2
c	1	2	3	1	4

(b) Valued constraint between agent 2 and agent 3

Table 1: Example

utility is $s_3 = \{x_2 = a, x_3 = a, x_4 = t\}$, which has utility 8. Furthermore, the upper bounds on the utilities of all other assignments is at least 8. This means that when a new ASK message is received, agent 2 can respond by sending a true good containing s_3 .

The example shows that agents are always able to respond to an ASK message with incomplete information, by sending a false good. Only when the agent is sure that his current best assignment will remain his current best assignment, he sends a true good.

The Algorithm

The goal of the algorithm (shown in Algorithm 1) is for each agent to aggregate enough information to make an optimal decision. In the example, we have seen that every agent does this by sending ASK messages. An agent responds to an ASK message by sending the assignment with the highest utility to its parent. By assuming that its children respond to ASK messages in a best first manner, it is able to maintain upper bounds on the utilities of all the assignments. The root agent is able to use this upper bound to determine when it has found the optimal assignment, while other agents use this upper bound to recognise when they have found the next best assignment. Before the algorithm is explained in more detail, some additional notation is needed.

Remember that an agent i stores the utility of a good obtained by child c in $E_c^i(s)$. To be able to distinguish between values based on true goods and values based on false goods, we use a binary variable $b_c^i(s)$ that is true if $E_c^i(s)$ is based on a true good and false otherwise, while $b^i(s) = \bigwedge_c b_c^i(s)$. Note that this means that $b_c^i(s)$ is also false when $E_c^i(s)$ is undefined, i.e. child c has not sent any good with an assignment compatible with s .

Furthermore, in order for an agent to determine its next best assignment, it needs to remember which "optimal" assignments it has already reported to its parent. To this end, let $sentGoods^i$ contain all the true goods that agent i has sent to its parent.

Given this set of sent assignment, let $s_{max} \in Ass^i \setminus sentGoods^i$ be the unreported assignment that currently has the highest utility

$$s_{max} = \operatorname{argmax}_{s \in Ass^i \setminus sentGoods^i} E^i(s) \quad (4)$$

In the example, an agent never reports its own assignment. Hence, given an assignment $s \in Ass^i$, let $[s]^i \equiv s$ be the

Algorithm 1: ASODPOP Agent

Receive(*parent*, ASK)

```

if  $\exists s_{max}$  then
  if valuation sufficient then
    Send( $P^i$ , GOOD( $[s_{max}]^i$ ,  $E^i(s_{max})$ , true));
    sent_goods  $\leftarrow$  sent_goods
     $\cup \{s \mid s \in Ass^i, s \equiv [s_{max}]^i\}$ ;
  else
    Send( $P^i$ , GOOD( $[s_{max}]^i$ ,  $E^i(s_{max})$ , false));
    send 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  $t \in Ass^i$  such that  $s' \equiv s$  do
   $E_{child}^i(t) \leftarrow V$ ;
   $b_{child}^i(t) \leftarrow b$ ;
if  $b$  then
  Adjust estimates  $E_{child}^i$  for all  $t \in Sep^i$  with
 $t \equiv s$  such that  $E_{child}^i(t) < V$ ;
   $E_{child}^{last} \leftarrow V$ ;

```

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

Receive(*parent*, VALUE(*context*))

```

Select the assignment  $s \in Ass^i$  that is compatible
with context and maximises  $E^i(s)$ ;
currentAssignment  $\leftarrow$  assignment of  $x_i$  in  $s$ ;
for all children  $c$  of  $i$  do
   $t \leftarrow$  assignment for  $sep^c$  given  $s$ ;
  send VALUE( $t$ ) to  $c$ ;

```

assignment equal to s , while not containing an assignment to x_i . For example, if $s = \{x_1 = a, x_3 = b\}$ then $[s]^1 = \{x_3 = b\}$.

As mentioned in the example, the last true good that has been received from a particular child can be seen as an upper bound on utilities of the assignments yet to be reported. Let E_c^{last} be the utility of the last true good to be received from child c , then let the upper bound of the utility a child c can obtain given assignment s be defined as

$$UB_c^i(s) = \begin{cases} E_c^i(s) & \text{if } b_c^i(s) = true \\ E_c^{last} & \text{if } E_c^{last} \text{ exists} \\ \infty & \text{otherwise} \end{cases} \quad (5)$$

$UB^i(s) = \sum_c UB_c^i(s)$ now defines to total upper bound on an assignment s .

Using these upper bounds, an agent is able to determine when it has received enough information in order to decide whether s_{max} is its next best assignment. That is, when an agent can determine when any additional goods received from its children will not change s_{max} .

Definition 3 Given an agent i and a set $S \subset Ass^i$. An assignment $s \in Ass^i$ is dominant conditional on the subset S , when $\forall s \in Ass^i \setminus S, E^i(s) \geq UB^i(t)$.

That is, when s is dominant conditional on the set S , s is the next best choice from Ass^i after the assignments in S .

Definition 4 (Valuation sufficient) An agent i is valuation sufficient if $b^i(s_{max}) = true$ and s_{max} is dominant conditional on $sentGoods^i$.

So when the utility of s_{max} is based on complete information and no other assignment will ever have a higher utility, agent i is valuation sufficient.

Initialisation

When the algorithm is initiated, an agent i is initialised with complete knowledge of the domains of the variables it has a constraint with. For example, when looking at the problem depicted in Figure 1, agent 2 knows the domains of both x_1 and x_3 . It does not, however, have any knowledge of x_4 . Furthermore, all the upper bounds are initialised to ∞

ASK/GOOD Phase

The phase in which all the agents aggregate information is called the ASK/GOOD phase. An important assumption is that all the children of an agent report true goods in a best first manner.

When a leaf agent i receives an ASK message, it first determines s_{max} . Since a leaf agent has no children, $E^i(s_{max}) = own^i(s_{max})$, and is thus exactly the utility agent i can obtain when s_{max} is used. It is not hard to see that this means that a leaf agent is always valuation sufficient. It thus responds with a good $\langle [s_{max}]^i, E^i(s_{max}), true \rangle$ and stores $[s_{max}]^i$ in $sentGoods^i$.

When a non leaf agent i receives an ASK messages, again s_{max} is calculated. Remember that we assumed that all children send true goods in a best first manner, and that these true goods are based on complete information. Thus, if i is valuation sufficient, we have that s_{max} is the agent's next best good. It thus responds with $\langle [s_{max}]^i, E^i(s_{max}), true \rangle$. In all

other cases, it responds with $\langle [s_{max}]^i, E^i(s_{max}), false \rangle$. Furthermore, it sends an ASK message to its children to ask for more information.

It remains to be discussed what happens when an agent i receives a GOOD message from child c , containing a good $\langle s, u, b \rangle$. Since an agent is not initialised with complete information on the domains of the variables in its separator, it could be that it was not aware of the assignment s . In this case it first updates the information on its separator. When the separator is up to date, for every $t \in Ass^i$ such that $t \equiv s$ it sets $E_c^i(t) = u$ and $b_c^i(t) = b$.

At every stage of the algorithm, an agent i has assignments s for which $E_c^i(s)$ is not yet defined, i.e. it has not received any information on s from c . This means that the current utility for s is based on only part of the tree rooted at i . It can now decide to leave it be, and only work with the partial information. However, when the problem at hand allows it, it can decide to make an estimate of this unknown value. This will not speed up the process of proving an optimal solution has been found. However, it could lead to a speed up in the convergence to the optimal value.

VALUE Propagation Phase

We have discussed how an agent can collect information about its sub tree, but we have not yet elaborated on how an agent receives information on the assignments used by the agents in its separator. Remember that agents might be operating under certain time constraints. It is therefore not always possible to wait until the algorithm has ended, which means that agents continuously need to update their current best assignment and propagate this to their children.

Lets start with the root agent. It always sets itself to s_{max} and whenever this changes, it notifies its children using a VALUE message. The other agents always set their variables to the values that maximise the known utilities given the assignments in their separator. Thus their assignment can change either when they receive a new VALUE message from their parent or when they receive a good from one of their children. Each time an agent changes its assignment, it sends a VALUE message to all its children to notify them. A VALUE message to child c contains an assignment to all the variables in sep^c that his parent is aware of.

The algorithm terminates when the root agent is valuation sufficient, and the optimal assignments have been propagated using VALUE messages.

Completeness and Termination

The algorithm described above only terminates when valuation sufficiency has been reached by the root agent. The following theorem states that ASODPOP using bounds always terminates, i.e. reaches valuation sufficiency when the domains are finite.

Theorem 1 (Termination) When the variable domains are finite, ASODPOP using bounds always terminates.

Proofsketch

All the leaf agents are initialised with complete knowledge on all their constraints, and are thus fully informed over their part of the problem. By definition, the root agent will

continue to send ASK messages until it is valuation sufficient. Lets assume that the root agent never reaches valuation sufficiency. Using induction it is not hard to show that the ASK messages send by the root agent pull information from the leaf agents up into the tree, and since the domains are all finite, at some point the root note must have complete knowledge of all the utilities, and thus all the upper bounds are equal to the actual value. Now since there must be at least one assignment that has a maximal utility, the root agent must become valuation sufficient at some point. \square

The algorithm is designed with the assumption in mind that all the true goods are sent in a best first order. The following proposition shows that if this assumption holds, an agent has found its next best good when it is valuation sufficient.

Proposition 1 *Given an agent i , if all its children report true goods in a best first order and it is valuation sufficient, then no assignment $s \neq s_{max}$ not in $sentGoods$ will be able to obtain a valuation $E^i(s)$ greater than $E^i(s_{max})$, i.e. s_{max} is the next best assignment.*

Proofsketch Suppose that agent i is valuation sufficient, and that there is some assignment s such that $s \notin sentGoods$ and $s \neq s_{max}$. We can then discern two different cases.

In the first case we assume that $s \in Ass^i$. Now let $E^i_{optimal}(s)$ be the optimal utility of the subtree rooted in i when assignment s is part of the global assignment and assume that $E^i_{optimal}(s) > E^i(s_{max})$. Assuming that every child always sends it true goods in a best first order, it is always the case that $UB^i(s) > E^i(s_{max})$. But since i is valuation sufficient this cannot be the case, hence $E^i(s_{max}) \geq E^i_{optimal}(s)$.

In the second case assume that $s \notin Ass^i$. This means that agent i does not know of the existence of s . Despite this, it can say something concerning the upper bound of s . Since all agent i 's children respond with true goods in a best first manner, it can assume that the value $E^i_c(s)$ is smaller than $E^i_{c^{last}}$, which means that it cannot be the case that $UB^i(s)$ is bigger than any already calculated upper bound. Hence, by assumption, $E^i_{optimal}(s)$ can never be greater than $E^i(s_{max})$, making s_{max} the next best assignment \square

Remember from the description of the algorithm that the leaf agents always sent their goods in a best first order. With the help of Proposition 1, it is not hard to show that the algorithm is complete.

Theorem 2 (Completeness) *Given that the leaf agents send their true goods in best first order, ASODPOP is able to find the optimal assignment.*

Proofsketch The algorithm terminates when the root agent is valuation sufficient. Theorem 1 shows that the root agent will always become valuation sufficient. Furthermore, Proposition 1 shows that this means that upon termination the root agent has found the assignment that allows the tree to obtain the highest utility. It is not hard to show, using induction, that the VALUE propagation phase ensures that the agents set their values to the optimal value. \square

Truck Task Coordination

Being able to efficiently distribute goods using a set of trucks has large practical values. In today's, ever more globalising world goods are becoming more and more mobile, and coordinating the movement of these goods is becoming increasingly more complicated. It is therefore important to have methods that can efficiently plan and make use of the locality present in most problems. In order to benchmark different approaches a proper model is needed. To that end we introduce a model for the TTC problem

The basic constituents of the model are

- a set of cities
- a set of roads between the cities
- a set of agents that represent the trucks
- a set of packets to be picked up and delivered

Together, the cities and roads form a map on which the trucks and packets are dispersed. The types of problems we are looking at are inspired by the problems parcel delivery services like DHL and TNT face. Drivers stay in a certain area, which means that the movements of each truck are restricted. Therefore, it is reasonable to assume that the trucks are restricted to certain regions on the map when picking up packages. These regions, however, can overlap. As a result, certain packages can be picked up by different trucks, and the agents must coordinate who picks up which packet. From now on, when we talk about a truck we mean the agent that represents the truck. In modelling this problem, one must make use of the fact that only certain regions overlap with each other, and thus trucks do not have to coordinate with every other truck over every other package.

Besides making sure that only one truck picks up a certain package, the individual trucks also have to take the cost of picking up and delivering a packet into account. The cost consists of the distance travelled, and thus the gas consumed. Note that this planning problem is a problem that is local to each of the trucks. A truck is only interested in which packets another truck delivers and the cost associated with it. It is not interested in the exact path the truck takes. In this problem, the planning and coordination are thus nicely separated.

Model

The model we use is based on (Bettex 2008). Let T be the set of trucks. Due to the overlapping areas, packets can be divided into two different types of packets. There are the packets that only a particular truck can pick up, and there are packets that several trucks can pick up. Given a truck $t \in T$, let O_t be the set of packages that only truck t can pick up and let S_t be the set of packages that truck t shares with other trucks.

Each truck $t \in T$ owns one variable $x_t \in \mathcal{P}(S_t)$ that consists of the set of packets it will pick up, where x_t can only contain packets that other trucks can also pickup. The cost of an instantiation of x_t is determined by the route taken by truck t when delivering its allocated packages. These costs are computed off line, and are represented as a unary constraint f_1 over x_t .

We also want to make sure that each packet is delivered, i.e. each packet is selected by at least one truck. Furthermore, a packet cannot be delivered by more than one truck, hence, each packet is to be selected by at most one truck. In the following we introduce two different ways of modelling this. The first does not introduce any additional variables but does require k -ary constraints, while the second uses only binary constraints but does need an additional variable.

Model 1 When two trucks t and t' have overlapping regions, they must make sure that they do not try to pick up the same packet. In other words, they must make sure that $x_t \cap x_{t'} = \emptyset$. This can be done by defining a constraint $f_2 : \mathcal{P}(D_t) \times \mathcal{P}(D_{t'}) \mapsto \mathbb{R}$, where $f_2(a, b) = 0$ if $a \cap b = \emptyset$ and $f_2(a, b) = \infty$ otherwise.

In order to make sure that every packet is delivered, a third constraint must be used. Say that k trucks can pick up a certain packet p . Then a k -ary constraint $f_3 : \mathcal{P}(D_{t_1}) \times \dots \times \mathcal{P}(D_{t_k}) \mapsto \mathbb{R}$ is needed between the trucks, such that $f_3(x_1, \dots, x_k) = 0$ if $p \in \bigcup_{i=1}^k x_i$ and $f_3(x_1, \dots, x_k) = \infty$ otherwise.

We now want to solve the following problem

$$x = \operatorname{argmin}_{x_1, \dots, x_n} \left(\sum_t f_1(x_t) + \sum_{\text{regions } t \text{ and } r \text{ overlap}} f_2(x_t, x_r) + \sum_{\text{regions } t_1, \dots, t_k \text{ overlap}} f_3(x_{t_1}, \dots, x_{t_k}) \right) \quad (6)$$

Model 2 In order to circumvent the addition of a k -ary constraint an extra variable y_p can be added for each packet p , where the domain of y_p consists of all the trucks that can pick it up. For example, if trucks 1, 5 and 7 can pick up packet p_1 , then $y_{p_1} \in \{1, 5, 7\}$. One now creates a constraint $f_4^{t,p}$ between a packet p and each truck t that can pick it up, that enforces that $y_p = t$ if and only if $t \in x_t$. Hence $f_4^{t,p}(x_t, y_p) = 0$ if $y_p = t$ and $p \in x_t$ and ∞ otherwise.

$$x = \operatorname{argmin}_{x_1, \dots, x_n} \left(\sum_t f_1(x_t) + \sum_{p \text{ is in the region of truck } t} f_4^{t,p}(x_t, y_p) \right) \quad (7)$$

Problem Generator

A problem is generated in the following manner. First, a grid of a predefined size is created, and on such a grid the

cities are randomly placed. Between the cities a network of roads is grown by first selecting one city in the map, and then iteratively adding the closest city to the graph.

In the next step, each truck is assigned a specific city in which it is to begin its days work and for each truck the areas are generated in such a way that a specified number over overlapping points is present. Next, the packets are dispersed over the network in such a way as to make the resulting problem connected. Finally, for each truck the cost of every combination of packets it can accept is calculated using a local search method with restart, where the path used to restart is an adaptation of the path the previous cycle has ended with.

The generation of f_1 is straightforward. However, due to limitations of the implementations of the algorithms at hand we were not able to directly implement constraint f_3 . The reason is that the present version of ASODPOP does not handle k -ary constraints. Hence, we have to use model 2.

Experimental Evaluation

We are interested in the performance of ASODPOP on the TTC coordination problem compared to other DCOP approaches. To that end, we compared our approach with both ADOPT (with the DO2 pre processing step (Ali, Koenig, and Tambe 2005)) and Asynchronous Distributed Local Search (DSA-C) (Zhang et al. 2005). The former is a complete solver while the latter is a stochastic, and thus incomplete solver. Since ASODPOP is designed to operate under time constraints, we are most interested in the convergence speed of the three different approaches. However, due to the fact that we simulate our runs on a single machine, the total runtime of an algorithm does not necessarily corresponds to the real behaviour. Therefore, we also look at the number of Non-Concurrent Constraint Checks (Meisels et al. 2002) to get a feeling of the level of parallelism present in the algorithms, where we take the look-up of the value of a constraint for a particular instance of the variables as a constraint check. Furthermore, we also look at the total number of message used by the different algorithms. Since both ADOPT and ASODPOP use messages to either send a single assignment plus the costs/utility associated with this assignment, or a value message, the size of the messages is constant and it thus suffices to compare the number of messages.

The implementation of ASODPOP that we used does assume full knowledge on the agents separator and uses random estimates to complement that partial information. Furthermore, one might note that where ADOPT is designed to minimise cost, ASODPOP maximises utility. The solution to this is simply to look at costs as having negative utility.

Experimental Setup

All three algorithms are implemented in the FRODO platform (Petcu 2006) and the experiments have been performed on a 2 Ghz Intel Core Duo MacBook with 1 GB of ram, running Leopard. The problems were generated on a map with 50 cities and 15 trucks and we ran two different experiments, varying two different parameters

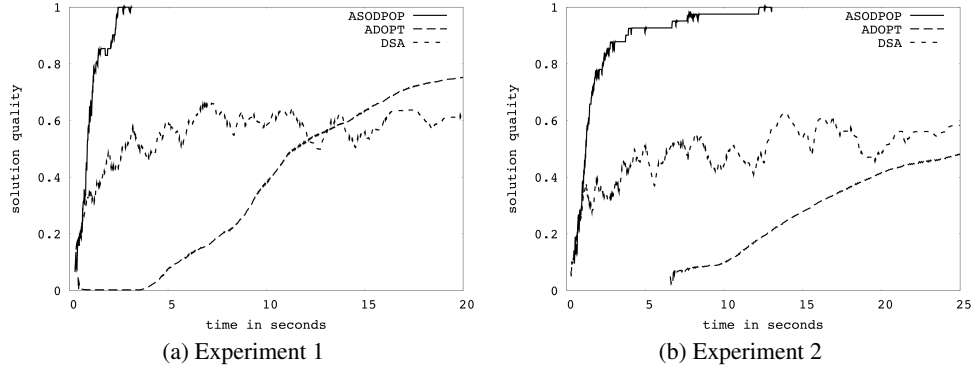


Figure 3: Converge speeds

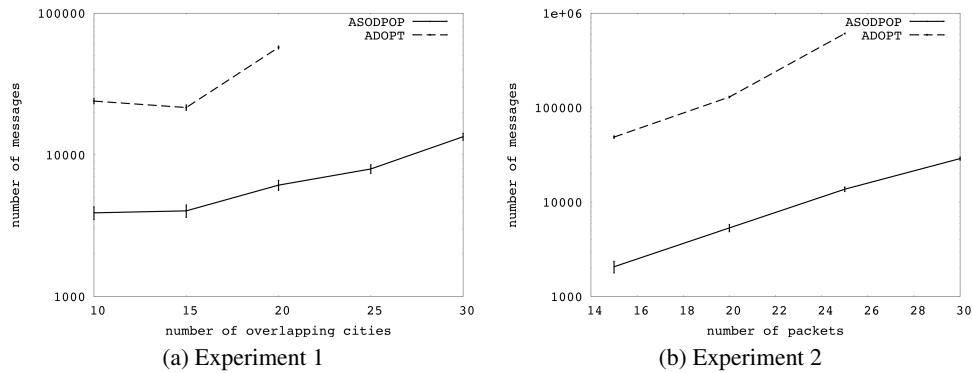


Figure 4: : the number of messages sent

Experiment 1: In this experiment we fixed the number of packets to 20 and varied the number of overlapping cities between 10 and 30 with steps of 5.

Experiment 2: In the second testset we fixed the number of overlapping cities to 20 and let the number of packets range between 15 and 30 with steps of 5.

For each combination of parameters, 40 different instances were generated. Since both the messages of ASODPOP and ADOPT are constant in size, we can directly compare the number of messages. For all the assignments with partial information, random estimates have been used to fill in the gaps.

Results

Figure 3 shows the convergence results on one type of problem in experiment 1 and one in experiment 2, averaged over 40 runs. Due to space limitations we cannot show the results for all the instances, but they all show a similar picture. ASODPOP converges much faster than both ADOPT and DSA, where the latter does not even come close to the solution. The gap for ADOPT in the right graph is caused by the fact that we simulate a distributed algorithm on a single machine. It does, however, show that the initialisation of ADOPT turns out to be quite expensive.

Figure 4 show the messages used by both ADOPT and ASODPOP to find the optimal solution. It shows that ASODPOP is 2 orders of magnitude more efficient in terms of the number of messages send. Because we simulate everything on a single machine, we also looked at the number of non-concurrent constraint checks to measure the level of parallelism in both ADOPT and ASODPOP. The results are shown in Figure 5, where ADOPT is slightly better when sending a message is free (instantaneous delivery and no computation). When there is a cost of sending a message however, ASODPOP performs better.

Discussion

In this paper we showed that DCOP techniques can directly be used for coordinating agent plans. It must be noted that DCOP techniques are only useful when the problem at hand is loosely coupled. In the TTC problem, for example, this amounts to small overlapping regions.

The algorithm used for coordination, ASODPOP, is an adaptation of the well known DPOP algorithm. The difference between ADOPT and ASODPOP is as follows. In ADOPT, agents choose values. Based on these values their children choose values and send costs upwards. Their parents then change their values based on the costs, and so on

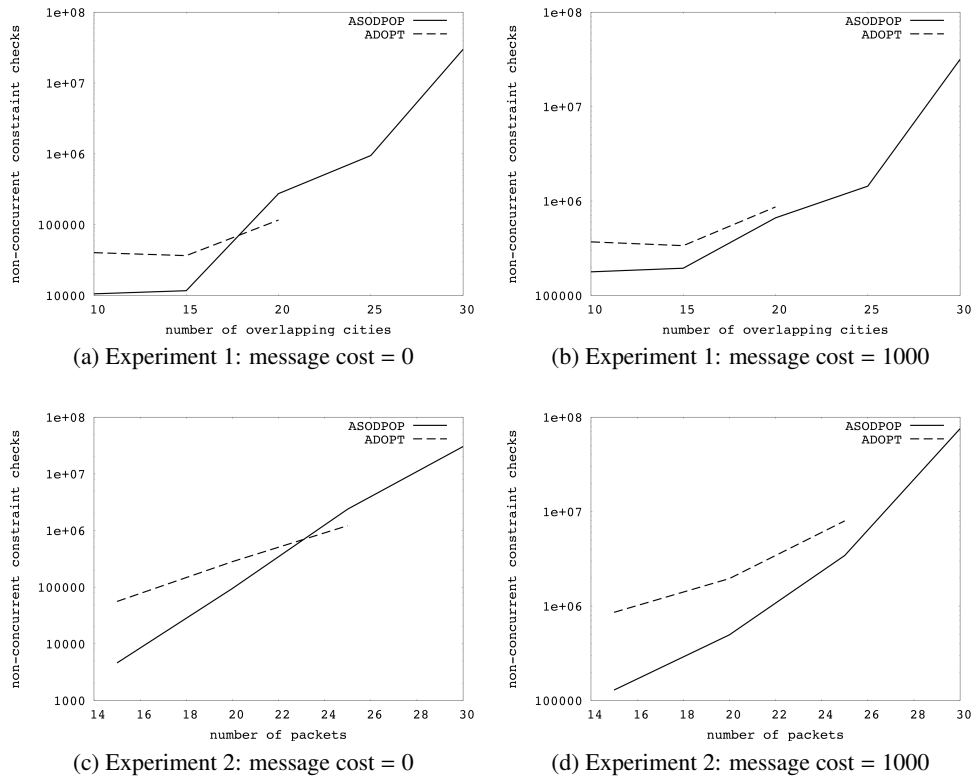


Figure 5: The number of messages sent

and so fort. Such a loop is not present in ASODPOP. On the contrary, in ASODPOP the costs (or utilities) that are send upwards are not influenced by the values send downward, i.e. the best first-order is only influenced by descendants of an agent and **not** by its parents.

The experiments showed that ASODPOP performs much better in terms of both the number of messages that are needed, but also in terms of speed of convergence, then ADOPT and DSA. This last property is useful when agents are under time constraints, i.e. they are not able to wait for the algorithm to find the optimal solution. In such a case, the faster the convergence, the better the result is when the algorithm is stopped prematurely.

Future Work

For future work we are planning to investigate the influence of the quality of the estimates on the convergence of ASODPOP. We also want to extend the TTC model, for example by letting an agent have a different variable for each overlapping area. We intend to replace the hard constraint that each packet should be delivered by a cost of not delivering a packet and it would be interesting to find ways of not having to pre compute all the costs off line.

Acknowledgements

Our thanks go to the anonymous reviewers, who's comments have been valuable in improving the final version of this

paper. We also would like to thank the participants of the DCR'08 workshop for their helpful comments on preliminary versions of ASODPOP.

References

- Ali, S.; Koenig, S.; and Tambe, M. 2005. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS '05*, 1041–1048. New York, NY, USA: ACM.
- Bettex, M. 2008. Truck-task scheduling using dpop. Semester project at the Artificial Intelligence Laboratory (LIA), EPFL (for a copy of the work, please mail the author).
- Faltings, B., and Macho-Gonzalez, S. 2005. Open Constraint Programming. *Artificial Intelligence* 161(1-2):181–208.
- Meisels, A.; Kaplansky, E.; Razgon, I.; and Zivan, R. 2002. Comparing Performance of Distributed Constraints Processing Algorithms. In *DCR 2002*.
- Modi, P.; Shen, W.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. *AAMAS'03*.
- Ottens, B., and Faltings, B. 2008. Asynchronous open dpop. In *Proceedings of the 10th International Workshop on Distributed Constraint Reasoning (DCR'08)*.
- Petcu, A., and Faltings, B. 2005. DPOP: A Scalable

Method for Multiagent Constraint Optimization. In *IJCAI 05*, 266–271.

Petcu, A., and Faltings, B. 2006. O-DPOP: An algorithm for Open/Distributed Constraint Optimization. In *AAAI-06*, 703–708.

Petcu, A. 2006. FRODO: A Framework for Open/Distributed constraint Optimization. Technical Report No. 2006/001 2006/001, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland). <http://liawww.epfl.ch/frodo/>.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, 614–621.

Zhang, W.; Wang, G.; Xing, Z.; and Wittenburg, L. 2005. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.* 161(1-2):55–87.