Artif Intell Rev (2007) 28:35–50
DOI 10.1007/s10462-008-9072-7

Using relaxations to improve search in distributed constraint optimisation

David A. Burke · Kenneth N. Brown

Published online: 13 September 2008

Abstract Densely connected distributed constraint optimisation problems (DisCOP) can be difficult to solve optimally, but finding good lower bounds on constraint costs can help to speed up search. We show how good lower bounds can be found by solving relaxed problems obtained by removing inter-agent constraints. We present modifications to the ADOPT DisCOP algorithm that allow an arbitrary number of relaxations to be performed prior to solving the original problem. We identify useful relaxations based on the solving structure used by ADOPT, and demonstrate that when these relaxations are incorporated as part of the search it can lead to significant performance improvements. In particular, where agents have significant local constraint costs, we achieve over an order of magnitude reduction in messages exchanged between agents. Finally, we identify cases where such relaxation techniques produce less consistent benefits.

Keywords Distributed constraint optimisation · Multi-agent systems

1 Introduction

Many combinatorial decision problems are naturally distributed over a set of agents: e.g. coordinating activities in a sensor network (Béjar et al. 2005), and scheduling meetings among a number of participants (Wallace and Freuder 2005). Distributed constraint reasoning (DCR) considers algorithms explicitly designed to handle such problems, searching for globally acceptable solutions while balancing communication load with processing time (Yokoo and Hirayama 2000). Many algorithms have been proposed that consider both satisfaction (DisCSP) and optimisation (DisCOP), including ADOPT (Modi et al. 2005).

D. A. Burke (✉) · K. N. Brown
Centre for Telecommunications Value-chain Research and Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Cork, Ireland
e-mail: david.burke@admeta.com

K. N. Brown
e-mail: k.brown@cs.ucc.ie

However, ADOPT's efficiency decreases as the size and density of the network of agents increase (Modi et al. 2005; Burke and Brown 2006). Search in ADOPT can be reduced if good lower bounds on costs are available. In this paper, we show how to generate effective lower bounds through problem relaxation. Relaxation changes a problem such that an optimal solution to the relaxed problem is a lower bound on the optimal solution to the original problem.

Relaxations have previously been applied to DisCSP, where they have been used to find solutions for over-constrained satisfaction problems (Yokoo 1993; Hirayama and Yokoo 2000; Wittenburg and Zhang 2003). In this paper, we investigate relaxation for DisCOP by removing selected inter-agent constraints. We present a relaxation framework, ADOPTRELAX, that allows ADOPT to be run in multiple phases, allowing one or more relaxed versions of the problem to be used when solving the original problem. Lower bound information gathered by the agents in one phase is used as input to the next, allowing portions of the search space to be pruned. While the concept of computing and re-using lower bounds dynamically during search has been explored in centralised constraint optimisation (Marinescu and Dechter 2005), this is the first investigation of such methods for DisCOP. The idea of using multiple levels of relaxations in AI planning was first introduced in Sacerdoti (1974), but this also has not been investigated in a distributed environment. We identify graph-based relaxations that are of particular use with the search structures used by ADOPT, and we show that incorporating these relaxations can significantly improve performance as the size and density of the network of agents increases. In particular, where agents have significant local constraint costs we show over an order of magnitude reduction in messages passed. We also identify one scenario where our general relaxations have fewer benefits due to poor local cost approximations. For this case we demonstrate how a problem specific relaxation can be implemented within our framework to provide greater performance improvements.

The remainder of the paper is structured as follows. In Sect. 2 we define the distributed constraint optimisation problem (DisCOP) and describe the ADOPT algorithm. We present our relaxation framework in Sect. 3 and then, in Sect. 4, propose a number of graph-based relaxations for use within this framework. We perform an experimental evaluation on our work in Sect. 5. Finally, in Sect. 6, we summarise our work.

2 Distributed constraint optimisation and ADOPT

A DisCOP consists of a set of *agents*, $A = \{a_1, a_2, \ldots, a_n\}$, and for each agent a_i, a set $X_i = \{x_{i1}, x_{i2}, \ldots, x_{im_i}\}$ of *variables* it controls, such that $\forall i \neq j\ X_i \cap X_j = \emptyset$. Each variable x_{ij} has a corresponding domain D_{ij} of values that it may be assigned. $X = \bigcup X_i$ is the set of all variables in the problem. $C = \{c_1, c_2, \ldots, c_t\}$ is a set of *constraints*, where each c_k acts on a subset of the variables $s(c_k) \subseteq X$, and associates a cost with each tuple of assignments to these variables $c_k : \prod_{ij:x_{ij} \in s(c_k)} D_{ij} \rightarrow \mathbb{N} \cup \{\infty\}$, where a cost of infinity indicates a forbidden tuple. The *agent scope*, $a(c_k)$, of c_k is the set of agents that c_k acts upon:[1] $a(c_k) = \{a_i : X_i \cap s(c_k) \neq \emptyset\}$. An agent a_i is a *neighbour* of an agent a_j if $\exists c_k : a_i, a_j \in a(c_k)$. A *global assignment*, g, is the selection of one value for each variable in the problem: $g \in \prod_{ij} D_{ij}$. A *local assignment*, l_i, to an agent a_i, is an element of $\prod_j D_{ij}$. For any assignment t and set of variables Y, let $t_{\downarrow Y}$ be the projection of t over the variables in Y. The global objective function, F, assigns a cost to each global assignment: $F : \prod_{ij} D_{ij} \rightarrow \mathbb{N} :: g \mapsto \sum_k c_k(g_{\downarrow s(c_k)})$. An

[1] In this study we restrict our attention to binary inter-agent constraints, i.e. constraints do not act on more than two agents.

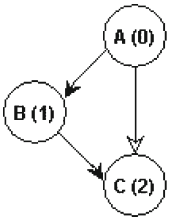
optimal solution is one which minimises F. The solution process, however, is restricted: each agent is responsible for the assignment of its own variables, and thus agents must communicate with each other, describing assignments and costs, in order to find a globally optimal solution.

ADOPT (Modi et al. 2005) is a complete DisCOP algorithm where agents execute asynchronously. Initially, the agents are prioritised into a depth-first search (DFS) tree, such that neighbouring agents appear on the same branch in the tree. Each agent a_i maintains a lower (LB_i) and upper (UB_i) bound on the cost of its subtree, which means that the lower and upper bounds of the root agent are bounds for the problem as a whole. Let H_i be the set of higher priority neighbours of a_i, and let L_i be the set of its children. During search, all agents act independently and asynchronously from each other. Each agent a_i executes in a loop, repeatedly performing a number of tasks:

1. `VALUE` messages, containing variable assignments, are received from higher priority agents and added to the current context CC_i, which is a record of higher priority neighbours' current assignments: $CC_i \in \prod_{j:a_j \in H_i} D_j$.
2. `COST` messages, containing lower and upper bounds, are received from children and stored if they are valid for the current context—for each subtree, rooted by an agent $a_j \in L_i$, a_i maintains a lower bound, $lb(l_i, a_j)$, and an upper bound $ub(l_i, a_j)$ for each of its assignments l_i. Each cost is valid for a specific context $CX(l_i, a_j) \in \prod_{k:a_k \in H_j} D_k$. Any previously stored cost with a context incompatible with the current context is reset to have lower/upper bounds of $0/\infty$.
3. A `THRESHOLD` message is received from the immediate parent of a_i—the threshold t_i is the best known lower bound for the subtree rooted by a_i.[2]
4. The local assignments with minimal lower and upper bound costs are calculated. Let C_{ij} be the constraint between x_i and x_j. The partial cost, $\delta(l)$, for an assignment of l_i to x_i is the sum of the agent's local cost $f_i(l_i)$, plus the costs of constraints between a_i and higher priority neighbours: $\delta(l_i) = f_i(l_i) + \sum_{j:a_j \in H_i} C_{ij}(l_i, CC_{i\downarrow x_j})$. The lower bound, $LB(l_i)$, for an assignment of l_i to x_i is the sum of $\delta(l_i)$ and the currently known lower bounds for all subtrees: $LB(l_i) = \delta(l_i) + \sum_{j:a_j \in L_i} lb(l_i, a_j)$. The upper bound, $UB(l_i)$, is the sum of $\delta(l_i)$ and the currently known upper bounds for all subtrees: $UB(l_i) = \delta(l_i) + \sum_{j:a_j \in L_i} ub(l_i, a_j)$. The minimum lower bound over all assignment possibilities, LB_i, is the lower bound for the agent a_i: $LB_i = \min_{l_i \in D_i} LB(l_i)$. Similarly, UB_i is the upper bound for the agent a_i: $UB_i = \min_{l_i \in D_i} UB(l_i)$.
5. The agent's current assignment, d_i, is updated and sent to all neighbours in L_i: if $t_i == UB_i$ then $d_i \leftarrow l_i$ that minimises $UB(l_i)$, else if $LB(d_i) > t_i$ then $d_i \leftarrow l_i$ that minimises $LB(l_i)$.
6. LB_i and UB_i are passed as costs to the parent of a_i, along with the context to which they apply, CC_i.

As the search progresses, the bounds are tightened in each agent until the lower and upper bounds are equal. If an agent detects this condition, and its parent has terminated, then an optimal solution is found and it may terminate.

[2] The threshold in ADOPT is used to reduce thrashing. During search agents discover lower bounds for different contexts. When an agent returns to a previously explored context, the search is guided by the fact that the agent knows it cannot find an assignment with a cost better than the threshold. For a detailed explanation of thresholds and ADOPT, please refer to (Modi et al. 2005).



Constraint A–B			Constraint B–C			Constraint A–C		
A	B	Cost	B	C	Cost	A	C	Cost
x	x	1	x	x	4	x	x	1
x	y	2	x	y	5	x	y	2
y	x	7	y	x	1	y	x	1
y	y	8	y	y	2	y	y	3

Fig. 1 Example DisCOP problem with 3 agents, each with a single variable. Black arrowheads indicate a direct parent child relationship in the priority tree. The number in brackets indicates the level of the agent in the priority tree

2.1 Search in ADOPT

In order to explain the benefit of using relaxations with ADOPT, we first describe the search process in more detail. Consider the simple optimisation problem in Fig. 1, where each agent has a single variable that can take the value x or y. Since ADOPT is an asynchronous search algorithm, there are many potential execution paths. We now consider one possible path. Initially, agents have no assignment or cost information regarding their neighbours. Let us assume that agent A selects the value x, and sends this assignment to all children. Agent B receives this and updates its context, $CC_B \leftarrow \{A = x\}$. B then selects the value that minimises its lower bound, considering the constraints with higher priority neighbours and the costs of subtrees. In this case, it selects the value x, which has a cost of $1 + 0 = 1$ (since B has not yet received cost information from agent C, the lower bound of subtrees is 0 and the upper bound is ∞ for all assignments). B then sends this assignment to its child, C, and sends its costs to its parent A including the context to which they apply, $\texttt{COST}[CX = \{A = x\}, lb = 1, ub = \infty]$. C has now received values from A and B, $CC_C \leftarrow \{A = x, B = x\}$. C selects its best value, x, and sends costs for this assignment to B, $\texttt{COST}[CX = \{A = x, B = x\}, lb = 5, ub = 5]$. On receiving this message, B sees that its lower bound for value y $(2 + 0)$ is now less than that for x $(1 + 5)$ and so changes its value to y. This assignment is sent to C and an updated cost message is sent to A, $\texttt{COST}[CX = \{A = x\}, lb = 2, ub = \infty]$.

However, while this search has been continuing, A has received the previous cost message from B (with $lb = 1$), and determines that its lower bound for the assignment $A = x$ is greater than for the assignment $A = y$ (lower bound = 0—initial default costs), and so switches its value and sends its new assignment to B. When B receives this, it updates its context, $CC_B \leftarrow \{A = y\}$, and detects that the costs it has stored are no longer compatible (as they are for the context $A = x$). These costs are reset to have a lower bound of 0.[3] B then searches through each of its values again, beginning by choosing the value x and sends the message $\texttt{COST}[CX = \{A = y\}, lb = 7, ub = \infty]$ to A. On receiving this, A sees that the lower bound for its assignment of y is now greater than that of x, and so switches assignment, $A = x$. When B receives this new assignment from A it again invalidates the cost information it has collected in the meantime, forcing it to rediscover the cost information for the context $A = x$. A also sends a threshold $t = 1$ to B, which is the best lower bound that is known by A for the assignment $A = x$, i.e. when x was previously assigned to A, the highest lower bound received in a cost message from B was 1. The threshold is used to reduce repeated search, as B only switches assignment when its current assignment has a lower bound greater

[3] To avoid exponential space requirements, only costs compatible with the current context are kept. Incompatible costs are reset to have lower/upper bounds of $0/\infty$.

than this threshold. However, in this example, the threshold is broken when C next sends a cost message, COST$[CX = \{A = x, B = x\}, lb = 5, ub = 5]$, and so no search is avoided. In general, the higher the threshold, the less context switching occurs.

From this example, we can see that there is a significant overhead involved with context switching in ADOPT. By reducing the context switching we can prune the search space. One technique that can be used to do this is to use informed lower bounds in each agent (Ali et al. 2005). We can see that if A just considers its constraint with B, then the minimum cost it will incur by choosing y is 7. If it uses this value, instead of 0 as its default lower bound for the assignment $A = y$, then it will never select y, thus reducing the context switching that occurs. Note that this informed bound was derived through examination of only a subset of the constraints of the agent. Informed lower bounds also provide better approximations for the threshold, which also leads to less context switching.

The benefits of using informed lower bounds in ADOPT has been previously demonstrated in Ali et al. (2005). In that paper, a dynamic programming preprocessing technique was used to produce lower bounds that were then used during a subsequent execution of ADOPT.[4] In the most efficient of their approaches, agents calculate lower bounds for each possible assignment of their direct parent considering both the bounds received from children and also the minimal costs that could be incurred in constraints with higher priority agents. While useful, this dynamic programming technique may not always be applicable because: (i) it assumes that each agent knows the domain of its parent; (ii) each agent produces bounds for *all* of its parent's possible assignments, while in fact the parent may have private constraints or constraints with other agents eliminating some of these assignments; (iii) when an agent has multiple variables this approach requires repeatedly solving its local problem for each possible parent assignment, which can be expensive for large local problems. In Sect. 4, we make use of the same concept of 'informed lower bounds', but do so within a new relaxation framework.

3 Relaxation framework for ADOPT

ADOPTRELAX (Algorithm 1) builds on the ADOPT algorithm to allow iterated searches on a number of problem relaxations that lead to the optimal solution of the original problem. In a similar style to Sacerdoti (1974), the search is split into n phases, i.e, $n-1$ relaxations, and a final search on the original problem. The construction of these relaxed problems is described in Sect. 4. The current phase is denoted by $relaxLevel$, whereby a value of $n-1$ is the most relaxed problem and 0 is the original problem. The first phase uses the most relaxed problem (line 2). Once a solution to the current problem has been found, the relaxation level is checked (5). If the solution is for the original problem the algorithm terminates as normal (9). If it is for a relaxed problem, then the current bounds will be saved (6). In the $save()$ procedure, each agent will store its current lower bounds for each subtree and each assignment, including the contexts to which these bounds apply (13).[5] Non-leaf agents then send a SAVE message informing children that a save is in progress, while leaf agents send a SAVE_ACK to inform their parent that they have completed saving. The leaf agents are now ready to begin the

[4] Ali et al. (2005) also mention that ADOPT could be run on a relaxed version of the problem to produce the bounds, but they do not investigate this issue in detail.

[5] By saving a single context-dependent set of bounds after each relaxation for each subtree and each assignment, we keep to the principles of the original ADOPT algorithm, which requires polynomial space. More information could be stored, potentially leading to greater improvements, but would also lead to greater memory requirements.

Algorithm 1: ADOPTRELAX (modifications and additions to ADOPT)

```
1  init()
2  relaxLevel ← n − 1; currentProblem ← phase[relaxLevel];
3  ADOPT();
4  checkTermination()
5  if relaxLevel > 0 then
6  |  if isRoot() or SAVE_MSG received from parent then save();
7  else
8  |  if isRoot() or TERMINATE_MSG received from parent then
9  |  |  terminate();
10 save()
11 forall a_j ∈ L_i forall l_i do
12 |  relaxLB(relaxLevel, l_i, a_j) ← lb(l_i, a_j);
13 |  relaxCX(relaxLevel, l_i, a_j) ← CX(l_i, a_j);
14 if !isLeaf() then send SAVE_MSG to children;
15 else
16 |  send SAVE_ACK_MSG to parent;
17 |  nextPhase();
18 receiveSaveAck()
19 if SAVE_ACK_MSG received from all children then
20 |  if !isRoot() then send SAVE_ACK_MSG to parent;
21 |  nextPhase();
22 nextPhase()
23 relaxLevel − −;
24 currentProblem ← phase[relaxLevel];
25 UB_i ← Inf;
26 if !isRoot() then LB_i ← 0; t_i ← 0;
27 ADOPT();
28 reset()
29 lb(l_i, a_j) ← 0; ub(l_i, a_j) ← ∞;
30 previousRelax ← relaxLevel + 1;
31 while previousRelax < n do
32 |  if relaxCX(previousRelax, l_i, a_j) compatible with CC_i then
33 |  |  lb(l_i, a_j) ← relaxLB(previousRelax, l_i, a_j);
34 |  |  CX(l_i, a_j) ← relaxCX(previousRelax, l_i, a_j);
35 |  |  break;
36 |  previousRelax ← previousRelax + 1;
```

next search phase (17), while non-leafs must wait until all of their children have saved their status (19). Once this has happened, they too can send a SAVE_ACK message to their parent, and are ready to begin their next search phase (21). In preparation for the next phase of the search, the relaxation level is decremented (23), which means that the algorithm will execute on the next relaxation, or, if there are no more relaxations, on the original problem. The upper bound for the agent is reset, and for all agents except the root, the lower bound and threshold are reset. Since all non-root agents have bounds that are dependent on the assignments of higher priority agents, they are not guaranteed to be lower bounds over all possible solutions. However, the bounds of the root/highest priority agent reflect the entire problem. Since the relaxed problem lower bound is guaranteed to be less than or equal to the lower bound of the original problem, the root can keep this bound.

The next phase of the search has two advantages over the initial search. First, the root has a lower bound that will be propagated down through the priority tree as thresholds to each agent, preventing some repeated search. Second, each agent has a lower bound for each subtree and each of its local assignments. When costs are reset, this lower bound can be used whenever the current context is compatible with the context of the stored lower bound (32), resulting in reduced context switching. When multiple relaxations are used, the saved lower bounds are considered in reverse order, and the first compatible bound is used (31).

Using a general DisCOP algorithm such as ADOPT in each search phase provides us with a general framework that allows us to compute lower bounds in a decentralised manner for arbitrary problem relaxations with different topologies. While it would also be possible to use other algorithms to solve the relaxed problems, it may be more difficult to exchange information between phases such that the information could still be used by ADOPT. Related work includes Ali et al. (2005), where lower bounds are produced for ADOPT using a dynamic programming approach. While their algorithms are efficient, the range of problem relaxations that can be considered is restricted in order to ensure polynomial time execution. Also, as mentioned in Sect. 2, this technique can perform unecessary work compared to a search based algorithm in some scenarios. In contrast to their approach, our general search based method can consider any problem relaxation, while also having the ability to perform multiple relaxations.

4 Relaxations in ADOPT

To use the relaxation framework we must first define problem relaxations. There are a number of different ways in which to relax distributed constraint problems, e.g. agents could be removed, constraints could be deleted or costs of tuples in constraints could be reduced or removed. Previous experimental analysis has shown the number of inter-agent constraints to be a key factor in determining the 'hardness' of distributed constraint problems (Hirayama et al. 2000; Modi et al. 2005; Burke and Brown 2006), so we will focus on removing inter-agent constraints. The next question is deciding which constraints to remove. We want to remove constraints to produce a relaxed problem that can be quickly solved, while still containing enough of the original problem to provide meaningful lower bounds. We will use our knowledge of the context switching behaviour and the priority tree structure to determine which constraints to remove. Figure 2a shows the priority tree of an example problem. The current context of each agent consists of assignments to all higher priority neighbours of the agent, plus higher priority non-neighbours that impact on the costs received by the agent. We can reduce the space of possible contexts in agents, and in turn the number of context switches that will occur, by removing constraints. We now make two important observations:

1. The costs stored by an agent may become incompatible if they are dependent on agents of higher priority. E.g. the costs that agent H stores for its child J have a context that contains the assignment to D (because J has a constraint with D), and so become incompatible if D changes its assignment.
2. The higher up in the search tree that a context switch occurs, the greater the potential impact, i.e, when agents change their assignment, it will lead to a new search involving all lower priority agents, and so a context switch in higher priority agents can be more expensive than in lower priority agents. E.g. a context switch in agent B may affect all agents $C - J$, while a context switch in D only affects agents H and J.

Based on these observations, we propose three relaxations to investigate:

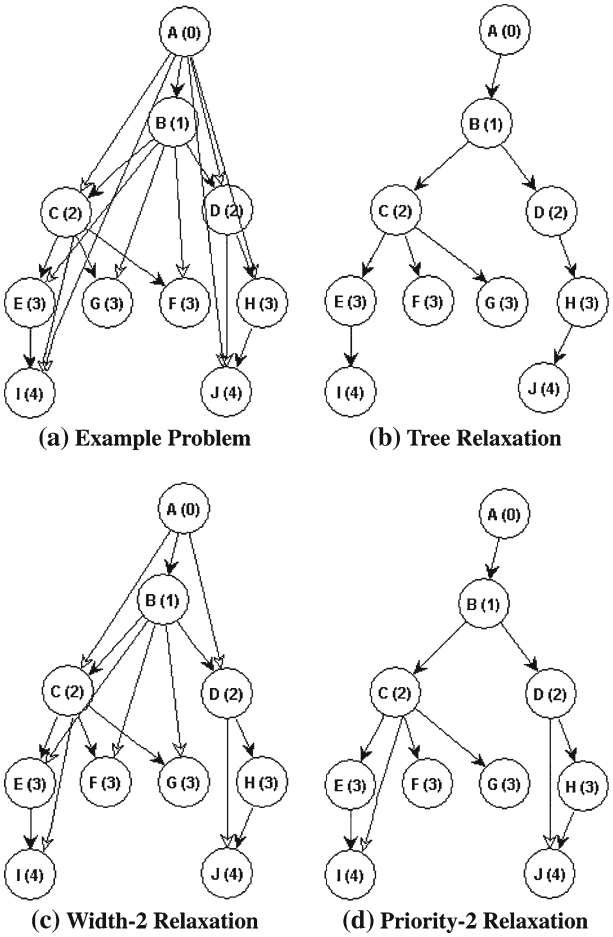


Fig. 2 (**a**) Example DisCOP agent graph. Arrows indicate constraints between agents, with black arrowheads indicating parent–child relationships within the priority tree. Number indicates level of agent in priority tree. (**b**) *TREE* relaxation removes all non-parent/child constraints. (**c**) *WIDTH-2* removes all constraints that span greater than 2 levels. (**d**) *PRIORITY-2* removes all non-parent/child constraints from top 2 levels

1. ***TREE*** : remove all non-parent/child constraints in the tree;
2. ***WIDTH-X*** : remove all constraints spanning more than X levels in the tree;
3. ***PRIORITY-X*** : remove all non-parent/child constraints from agents with priority less than X.

Taking into account the first observation, in the *TREE* relaxation, we remove all non-parent/child links, reducing the context space of each agent to be dependent on just one other agent—its immediate parent (Fig. 2b). In this relaxation, all costs received by an agent are independent of any higher priority agents, and so they are valid for all contexts and will never need to be reset. The *TREE* relaxation should make the problem much easier to solve, but if the network is densely connected it will remove many constraints, which means that the resulting bounds may not be good approximations. It may still be useful for loosely connected networks and also where agents have complex local problems. By only removing inter-agent

constraints, each agent's internal problem is still considered in full, and so local costs still contribute to the relaxed bounds.

If we want to retain more constraints, we can generalise the *TREE* relaxation. *WIDTH-X* reduces the context space of each agent to be dependent on at most X agents (Fig. 2c). This is done by removing all constraints that span greater than X levels in the tree, thus reducing the *width* (Dechter 2003) of the given graph ordering to be at most X. In fact, $TREE = WIDTH\text{-}1$. This relaxation allows us to trade off solving the relaxed problem quickly (low values for X) against getting a good lower bound (high values for X). It may be that different values of X may be of use for different density networks. It should be noted that in *TREE* the lower bounds found in the relaxation are compatible with all contexts, while in *WIDTH-2* this is not the case. In Fig. 2c, for example, the lower bounds of agent H for its child J are dependent on the assignment of D. This means that the final bounds stored by H will be useful when solving the original problem only when D has an assignment compatible with the stored context.

Our next relaxation considers the second observation we made previously. That is, we would like to reduce context switches in agents higher up in the search tree. The *PRIORITY-X* relaxation is thus biased towards removing constraints that appear higher up in the tree. *PRIORITY-X* removes all non-parent/child constraints from agents with a priority less than X (Fig. 2 (d)). This may allow fewer constraints to be removed while achieving greater search savings.

Each of these relaxations provide lower bounds that can be used to prune the search space in subsequent search phases. Multiple relaxations can be used in a single execution of the algorithm, with the bounds from each phase feeding into the subsequent phase, e.g. *TREE* could be followed by *PRIORITY-2* before the original problem is solved. Finally, note that these relaxations can be performed in a distributed manner. The priority tree can be created using a decentralized algorithm (Chechetka and Sycara 2005). Then, using only knowledge of their own priority and the priority of their neighbours, agents can remove the necessary inter-agent constraints.

5 Experiments

We compare the original ADOPT with ADOPTRELAX on three problem domains: random DisCOP; meeting scheduling; and minimum energy broadcast for wireless networks. ADOPTRELAX is run using each of *TREE*, *WIDTH-2* and *PRIORITY-2* relaxations individually as part of a two-phase search, and also using the combination *TREE -PRIORITY-2* as part of a three-phase incremental search. The experiments are run in a simulated distributed environment: we use one machine but each agent runs asynchronously on its own thread. In problems where agents have multiple variables, a centralised solver is used to make local assignments. To compare performance, we recorded the number of messages communicated by the agents, and also the number of non-concurrent constraint checks (NCCC) (Meisels et al. 2002). The results of both metrics were comparable, so we now only display graphs for the number of messages. All results are averaged over 20 test instances.

5.1 Random distributed constraint optimisation problems

In the random problems, we use 10 agents, each with a single variable of domain size 5. For each constraint, 90% (the tightness) of tuples have a non-zero cost chosen uniformly

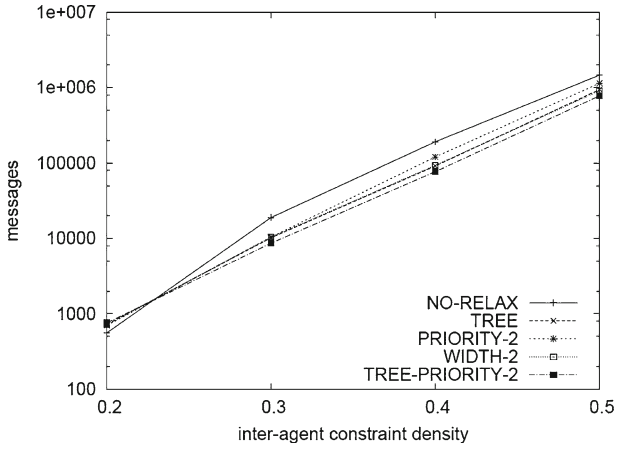


Fig. 3 Random DisCOPs varying inter-agent constraint density: 10 agents, each with one variable of domain size 5; tightness = 0.9; costs from 1 to 3

from the set {1, 2, 3}. The inter-agent constraint density is varied between 0.2 and 0.5.[6] A characteristic of these problems is that all costs are on inter-agent constraints, i.e. there are no local agent costs.

Figure 3 (log scale) shows that the relaxations give an improvement over the standard ADOPT as the density increases. For a density of 0.2 (9 constraints between 10 agents) our proposed relaxations do not remove any constraints, and so using them incurs an overhead of executing multiple phases of search without any benefits. However, for larger densities the relaxations come into use. *PRIORITY-2* finds high lower bounds and for less dense problems these bounds are found quickly. For denser problems it can take longer, and so the benefit from relaxation only accrues late in the search, hence worse performance for higher densities. *TREE* always finds bounds quickly, although for denser problems, these bounds will be further from the actual solution. *TREE* slightly outperforms *WIDTH-2* up to a density of 0.4 but *WIDTH-2* is better for 0.5. By combining two relaxations, as in *TREE -PRIORITY-2*, there is an increased overhead, i.e. an extra search phase. However, as the density increases, this overhead becomes less important and *TREE -PRIORITY-2* outperforms the other relaxations, giving almost 50% improvement over ADOPT for density 0.5.

5.2 Meeting scheduling

We generate meeting scheduling problems following a model used by Petcu and Faltings (2005) and others. For each meeting each agent is involved in, it owns a variable with 8 values (meeting starting times). Variables in different agents that represent the same meeting are linked with equality constraints. Agents also have personal tasks (single variables with 8 values). Variables in the same agent are linked with inequality constraints (the agent can not have two meetings/tasks at the same time). Agents have preferences, represented as costs, for the values they would like to assign to each meeting/task. Note that in these problems the inter-agent constraints are hard constraints, which will have a cost of 0 when

[6] Increasing the number of inter-agent constraints is expensive (Hirayama et al. 2000). Most DisCOP algorithms are tested on problems with densities no greater than 0.4, e.g. (Modi et al. 2005; Petcu and Faltings 2005; Gershman et al. 2006).

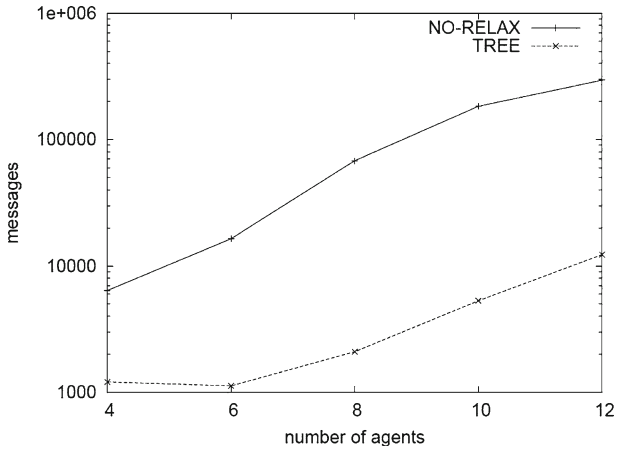


Fig. 4 Meeting scheduling problems: number of meetings = number of agents; 2 attendees per meeting; 2 personal tasks per agent; maximum of 4 meetings per agent

satisfied. Therefore, the costs incurred in solutions to the problem are local costs, i.e. the preferences of the agents. Removing inter-agent constraints allows the agents to choose more preferable local assignments, but in both the relaxed and original problems local costs will be incurred.

In our first experiment, we set the number of meetings equal to the number of agents. This setting means that all agent graphs will be a tree plus one additional constraint. Relaxing this constraint using the *TREE* relaxation gives remarkable results (Fig. 4), saving over an order of magnitude reduction in messages. The key reason for this is that the agents have significant local costs and since the problem relaxations consider the local problems in full, strong lower bound approximations can be found, allowing greater pruning of the search space.

In Fig. 5 we show results for increasing the number of agents with the inter-agent constraint density fixed to 0.3. All relaxations apart from *WIDTH-2* show over an order of magnitude improvement for 8 agents, and ADOPT hits an imposed cutoff of 10^6 messages for all instances greater than 8 agents. *PRIORITY-2* and *TREE* are successful for lower numbers of agents, but *TREE -PRIORITY-2* is the best once the problems are increased to contain 10 agents, similar to the single variable problems with a similar link density and number of agents. *WIDTH-2* becomes more competitive when there are more agents and more opportunities to remove constraints.

5.3 Minimum energy broadcast problem

In our final experiment we examine a new problem for DisCOP. An ad hoc network is a collection of wireless devices that form a network without any centralised infrastructure. When the devices are deployed they must first configure themselves to form a correctly functioning network. One configuration task when operating in a battery limited environment is the minimum energy broadcast (MEB) problem (Papadimitriou and Georgiadis 2006). The aim is to configure the power level in each device such that if a specified source device broadcasts a message it will reach every other device either directly or by being retransmitted by an intermediate device (a broadcast tree is formed). The desired configuration is that which minimises the total energy required by all devices, thus increasing the lifetime of the network.

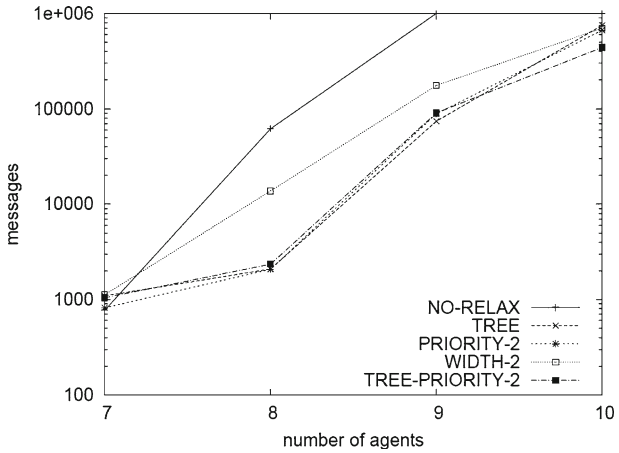


Fig. 5 Results of meeting scheduling problems: inter-agent link/meeting density = 0.3; 2 attendees per meeting; 2 personal tasks per agent; maximum of 4 meetings per agent

To formulate the problem as a DisCOP, we have an agent, a_i, representing each device in the network. The neighbours of a_i include all agents that a_i can communicate with when broadcasting at its maximum power level. For each neighbour a_j, a_i has a public variable, taking one of 3 values, indicating the *relationship* between the two devices in the current solution (broadcast tree): $0 \Rightarrow$ not connected; $1 \Rightarrow a_i$ is parent of a_j; $2 \Rightarrow a_i$ is child of a_j. An inter-agent constraint between each pair of neighbours ensures that the corresponding variables in neighbouring nodes match up correctly, i.e. both are 0, or else one is 1 and the other is 2. To construct a tree, each agent is constrained to have exactly one parent in the broadcast tree. The exception to this is the source device, which is not allowed any parents. The agents also have a private variable corresponding to each of these public variables set to be the energy cost incurred due to the setting of that public variable, i.e. if the public variable is 1 then the private variable is assigned the energy cost for broadcasting to that neighbour, otherwise it is assigned 0. A private constraint over all of these 'energy cost' variables states the total cost for a_i to broadcast to all of its children is the maximum of these costs. Each agent also has a *hop-count* variable, indicating how many hops that device is from the source device. A second inter-agent constraint between neighbouring agents ensures that the hop-count of a child in the broadcast tree is one greater than its parent, thus preventing cycles. Problems were generated by placing between 4 and 12 devices randomly in a $150 \times 150\,\mathrm{m}^2$ area. Devices can broadcast to a maximum of 30 m and a standard radio propagation model is used. A check was performed on each problem instance to ensure that a broadcast tree was feasible, i.e. problem instances were rejected if any device did not have another device within range or if all devices could not form a single network when broadcasting at their maximum power.

The results in Fig. 6 (for clarity, only the best relaxations are shown) are less impressive than for the previous problem domains. Of our graph-based relaxations, *TREE -PRIORITY-2* again provides the best results. The greatest improvement is seen on problems with 12 devices, where *TREE -PRIORITY-2* outperformed ADOPT on 70% of problems, with up to 74% improvement on some instances. But the average improvement ($\sim$11%) is marginal, and on 30% of problems ADOPT is better.

The question now is why the relaxation techniques that worked well for other examples only work sporadically in this problem, and, in particular, whether there are features of

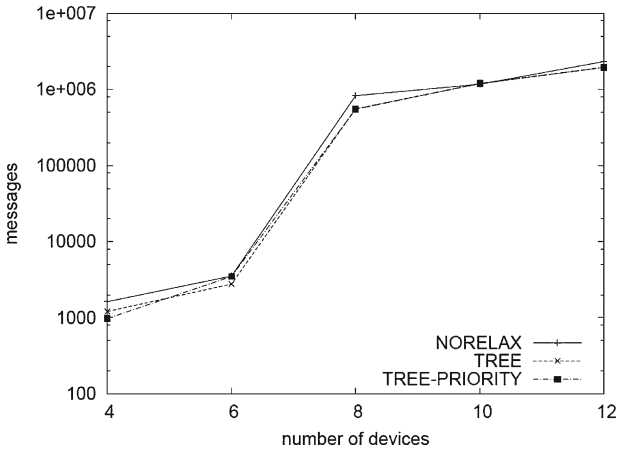


Fig. 6 Results of minimum energy broadcast problems: uniform distribution of devices over area of 150 × 150 m; devices have maximum of 30 m broadcast radius

individual problem instances that work against the relaxations. One of the main differences between the MEB problem and the meeting scheduling problem is in the cost structure. For meeting scheduling, each agent can incur local costs (based on its preferences for meeting times) regardless of what inter-agent constraints it has; for the MEB problem, an agent only incurs a cost when it acts as a parent to one or more of its neighbours. An agent must be connected to the rest of the network, and it must have at least one parent, and thus it is the presence of the inter-agent constraints on all of an agent's relationship variables that forces one of its neighbours to include a cost. So if we remove an inter-agent constraint, we remove the link between a pair of relationship variables. Each agent can then claim that its neighbour is the parent. The agent then believes it is connected to the network, and so may set all its other relationship variables to indicate no connection to any other neighbour. It then does not act as a parent, and so incurs no cost. If multiple agents are able to do this in a relaxed problem, the lower bounds that are discovered will be poor, and no search savings will be made. Note, however, that an agent may not be able to disconnect itself in this way, depending on the remaining constraints with its neighbours. As a rough estimate, the more agents that have constraints removed from them, the more likely it is that the relaxation will return poor bounds. To test this, we examined the instances with 12 devices, and measured the percentage improvement in messages sent compared to the percentage of agents that had constraints removed by each relaxation. In Fig. 7, we see that there is a clear difference in performance, with the best results coming when 25% of agents are affected, and very little improvement when 50% or more of agents are affected. Thus for certain cost structures, simple heuristics based on graph topology are not always appropriate.

To address this problem and demonstrate the flexibility of our relaxation framework, we introduce a problem-specific relaxation—*MEB-RELAX* —which does not remove inter-agent constraints linking the relationship variables, with the intention of forcing most agents to incur a local cost and thus produce better lower bounds. Instead, we simply remove all constraints on the hop-count variables (Fig. 8c), which allows cycles to be formed in the broadcast 'tree' of the relaxed problem. We ran this relaxation on its own and with a modified version of the *TREE* relaxation, that also had hop-count constraints removed (Fig. 8b).

The results are shown in Fig. 9. Our custom *MEB-RELAX* relaxation provided only minor improvements on the other relaxations when used on its own, but when combined with *TREE* ,

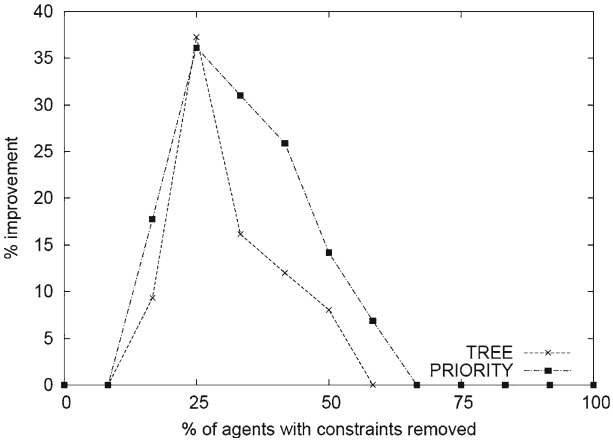


Fig. 7 Minimum energy broadcast problems—12 devices. Impact of relaxations based on the percentage of agents with constraints removed

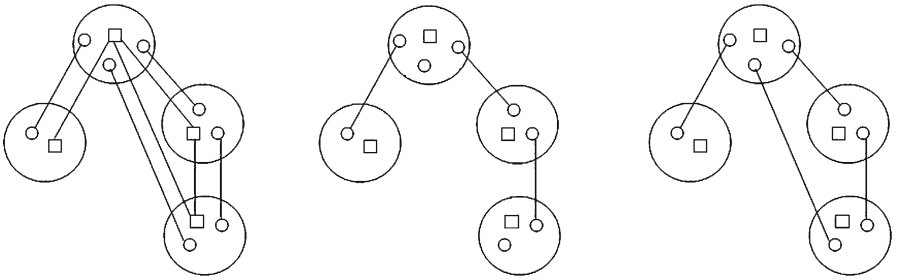


Fig. 8 (**a**) Example MEB problem showing public variables and constraints. Circles indicate relationship variables; squares indicate hop-count variables. (**b**) *TREE* relaxation for MEB problem removes: all hop-count constraints; and relationship constraints between non-parent/child agents. (**c**) *MEB-RELAX* relaxation removes all hop-count constraints, but no relationship constraints

significant gains were made. Thus, we can see that relaxation heuristics that take account of the structure of the objective function can be powerful, and may provide an interesting direction for future work.

6 Conclusions and future work

We have proposed ADOPTRELAX, a novel relaxation framework that is an extension of the ADOPT DisCOP algorithm. ADOPTRELAX allows an arbitrary number of problem relaxations to be solved prior to solving the original problem. These relaxations produce lower bounds that allow portions of the search space to be pruned. We have proposed a number of graph-based relaxations which remove inter-agent constraints. We have shown, through experimental analysis on random DisCOPs and meeting scheduling that ADOPTRELAX can offer an order of magnitude speed-up, particularly where agents have significant local costs. We also identified one scenario, illustrated by a wireless network configuration problem, where graph-based relaxation has fewer benefits due to poor local cost approximations. For this

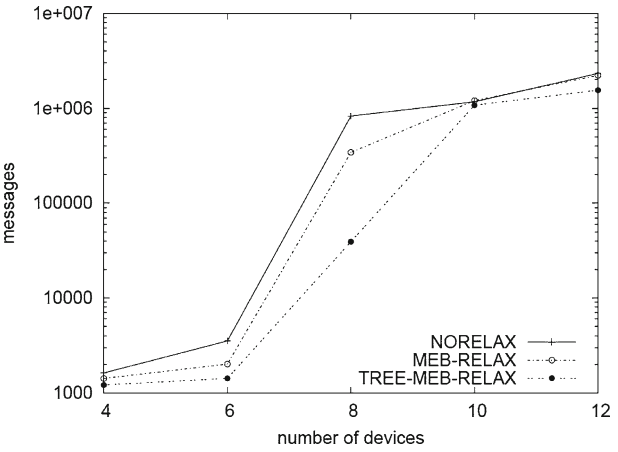


Fig. 9 Minimum energy broadcast problems. Specialised relaxation for MEB problem gives significant improvements

scenario, we demonstrated how a problem specific relaxation can easily be implemented within our framework to provide greater performance improvements.

Future work will investigate improved relaxation heuristics for these and other scenarios. We will further consider how to make more informed decisions on what constraints to remove from the graph, such that enough of the problem structure is maintained in order to achieve good lower bound approximations. For example, in the MEB problem, the analysis indicated that spreading the relaxations across the graph produces poor results. Instead, we will investigate heuristics which concentrate the relaxations on a few agents. In addition, we will examine relaxations where the constraints are modified rather than removed.

Acknowledgements This work is supported by Science Foundation Ireland under Grant No. 03/CE3/I405.

References

Ali S, Koenig S, Tambe M (2005) Preprocessing techniques for Accelerating the DCOP Algorithm ADOPT. In: Dignum F et al (eds) Proceedings of 4th international joint conference on autonomous agents and multiagent systems (AAMAS 2005), Utrecht, The Netherlands, July 25–29, 2005, pp 1041–1048

Béjar R, Domshlak C, Fernàndez C, Gomes C, Krishnamachari B, Selman B, Valls M (2005) Sensor networks and distributed CSP: communication, computation and complexity. Artif Intell 161(1–2):117–147

Burke DA, Brown KN (2006) Efficient handling of complex local problems in distributed constraint optimization. In: Brewka G et al (eds) Proceedings of 17th European conference on artifical intelligence (ECAI 2006), Riva del Garda, Italy, August 29–September 1, 2006, pp 701–702

Chechetka A, Sycara K (2005) A decentralized variable ordering method for distributed constraint optimization. In: Dignum F et al (eds) Proceedings of 4th international joint conference on autonomous agents and multiagent systems (AAMAS 2005), Utrecht, The Netherlands, July 25–29, 2005, pp 1307–1308

Dechter R (2003) Constraint processing. Morgan Kaufmann Publishers Inc., San Francisco, CA

Gershman A, Meisels A, Zivan R (2006) Asynchronous forward-bounding for distributed constraint optimization, In: Brewka G et al (eds) Proceedings of 17th European conference on artifical intelligence (ECAI 2006), Riva del Garda, Italy, August 29–September 1, 2006, pp 103–107

Hirayama K, Yokoo M (2000) An approach to over-constrained distributed constraint satisfaction problems: distributed hierarchical constraint satisfaction. In: Proceedings of 4th international conference on multiagent systems (ICMAS 2000), Boston, MA, USA, July 10–12, 2000, pp 135–142

Hirayama K, Yokoo M, Sycara K (2000) The phase transition in distributed constraint satisfaction problems: first results. In: Dechter R (eds) Proceedings of the 6th international conference on principles and practice of constraint programming (CP 2000), Singapore, September 18–21, 2000, pp 515–519

Marinescu R, Dechter R (2005) AND/OR branch-and-bound for graphical models. In: Kaelbling L, Saffiotti A (eds) Proceedings of the 19th international joint conference on artificial intelligence (IJCAI 2005), Edinburgh, Scotland, July 30–August 5, 2005, pp 224–229

Meisels A, Razgon I, Kaplansky E, Zivan R (2002) Comparing performance of distributed constraints processing algorithms. In: Proceedings of the 3rd international workshop on distributed constraint reasoning (DCR), Bologna, Italy, July 16, 2002, pp 86–93

Modi P, Shen W, Tambe M, Yokoo M (2005) ADOPT: asynchronous distributed constraint optimization with quality guarantees. Artif Intell 161(1–2):149–180

Papadimitriou I, Georgiadis L (2006) Minimum-energy broadcasting in multi-hop wireless networks using a single broadcast tree. Mob Netw Appl 11(3):361–375

Petcu A, Faltings B (2005) A scalable method for multiagent, constraint optimization. In: Kaelbling L, Saffiotti A (eds) Proceedings of the 19th international joint conference on artificial intelligence (IJCAI 2005), Edinburgh, Scotland, July 30–August 5, 2005, pp 266–271

Sacerdoti ED (1974) Planning in a hierarchy of abstraction spaces. Artif Intell 5(2):115–135

Wallace R, Freuder E (2005) Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. Artif Intell 161(1–2):209–227

Wittenburg L, Zhang W (2003) Distributed breakout algorithm for distributed constraint optimization problems—DBArelax. In: Proceedings of the 2nd international joint conference on autonomous agents and multiagent systems (AAMAS 2002), Melbourne, Victoria, Australia, July 14–18, 2003, pp 1158–1159

Yokoo M (1993) Constraint relaxation in distributed constraint satisfaction problems. In: Proceedings of the 5th international conference on tools with artificial intelligence (ICTAI 1993), Boston, MA, USA, November 8–11, 1993, pp 56–63

Yokoo M, Hirayama K (2000) Algorithms for distributed constraint satisfaction: a review. Auton Agents Multi-Agent Syst 3(2):185–207