A Comparison of Approaches to Handling Complex Local Problems in DCOP

David A. Burke and Kenneth N. Brown
Centre for Telecommunications Value-chain Research
and Cork Constraint Computation Centre,
Department of Computer Science,
University College Cork, Ireland

Abstract. Many distributed constraint optimisation algorithms require each agent to have a single variable. For agents with multiple variables, there are two standard approaches: decomposition – for each variable in each local problem, create a unique agent to manage it; and compilation – compile the local problem down to a new variable whose domain is the set of all local solutions. We compare these two approaches with each other and with a modified compilation approach that uses dominance and interchangeabilities to reduce problem size and speed up search. Our preliminary results show: (i) the basic compilation is almost never competitive; (ii) the modified compilation gives significant improvements over the other methods as the size and complexity of each agent's internal problem grows, as long as the number of inter-agent constraints and the domain size of the variables remains small; (iii) the decomposition approach is more appropriate to use as the number of inter-agent constraints and the domain size of the variables increase, as long as the overall problem size is small.

1 INTRODUCTION

Many combinatorial problems are naturally distributed over a set of agents: e.g. coordinating activities in a sensor network [2], coordinating vehicle schedules in a transport logistics problem [6], or scheduling meetings among a number of participants [21]. Distributed Constraint Reasoning (DCR) considers algorithms explicitly designed to handle such problems, searching for globally acceptable solutions while minimising communication between agents. Most of the algorithms, however, assume that each agent controls only a single variable. This assumption is justified by two standard reformulations [23] by which any DCR problem with complex local problems (i.e. multiple variables in each agent) can be transformed to give exactly one variable per agent:

(i) Compilation: for each agent, define a variable whose domain is the set of solutions to the original local problem;
(ii) Decomposition: for each variable in each local problem, create a unique agent to manage it.

In distributed constraint satisfaction (DCSP), it is known that neither of these methods scales up well as the size of the local problems increase, because of the space and time requirements of the reformulation (compilation), or the extra communication overhead (decomposition). To address these issues, algorithms for handling multiple local variables in DCSP have been proposed [1, 22, 11, 14]. These algorithms are specific to DCSP, since they reason about violated constraints, and cannot be applied directly to distributed constraint optimisation problems (DCOP), which are concerned with costs. For DCOP, the two original reformulations remain the standard way of handling complex local problems [17].

In this paper we investigate both of these techniques applied to DCOP, together with a modified version of the compilation method that makes use of dominance and interchangeabilities to reduce search [4, 5]. We evaluate the different approaches using the ADOPT [17] algorithm applied to distributed graph-colouring problems of various sizes. We believe this is the first experimental comparison of these techniques for DCOP, and we present preliminary results from tests executed in a simulated distributed environment. Our findings show that the decomposition and improved compilation methods are useful for problems with different characteristics while the basic compilation approach performs poorly on all but the smallest of problems. We show that the modified compilation approach can significantly outperform the other techniques when the agents' internal problems are large and dense. However, it requires the number of inter-agent constraints and the domain size of variables to be small. On the other hand, we show that the decomposition approach is more suited to problems where the number of inter-agent constraints and the domain size of variables is increasing but performs badly when agents' internal problems grow.

In Section 2 we give descriptions for DCOP and the ADOPT algorithm. The techniques for handling complex local problems that we will evaluate are described in Section 3. We then present the experiments we performed and their results in Section 4. This is followed by a discussion, analysing the merits of the different techniques in Section 5. Finally, in Section 6, we briefly summarise our work.

2 BACKGROUND

A Distributed Constraint Optimisation Problem consists of a set of *agents*, $A=\{a_1, a_2, ..., a_n\}$, and for each agent a_i, a

set X_i=$\{x_{i1}, x_{i2}, \ldots, x_{im_i}\}$ of *variables* it controls, such that $\forall i \neq j\ X_i \cap X_j = \phi$. Each variable x_{ij} has a corresponding domain D_{ij}. $X = \bigcup X_i$ is the set of all variables in the problem. $C = \{c_1, c_2, \ldots, c_t\}$ is a set of *constraints*. Each c_k has a *scope* $s(c_k) \subseteq X$, and is a function $c_k : \prod_{ij:x_{ij} \in s(c_k)} D_{ij} \rightarrow I\!N$. The *agent scope*, $a(c_k)$, of c_k is the set of agents that c_k acts upon: $a(c_k) = \{a_i : X_i \cap s(c_k) \neq \phi\}$. 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 $\prod_j D_{ij}$. Let t be any assignment, and let Y be a set of variables, then $t_{\downarrow Y}$ is the projection of t over the variables in Y. The global objective function, F, assigns a cost to each global assignment: $F : \prod_{ij} D_{ij} \rightarrow I\!N :: g \mapsto \sum_k c_k(g_{\downarrow s(c_k)})$. An 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.

Most research has focused on DisCSP [23, 20, 10, 3, 24]. Recently, several algorithms for DCOP have been proposed [15, 18, 9], including ADOPT [17] – a complete algorithm that allows agents to work asynchronously. Agents are prioritised into a tree. During search, each agent repeatedly and asynchronously performs the following tasks:

1. receive incoming values from higher priority agents and add to *CurrentContext*, which is a record of higher priority neighbours' current variable assignments;
2. receive costs from children and store if they are valid for *CurrentContext* (the agent maintains separate costs for each child/subtree for each of its possible values);
3. calculate the cost of each of its possible values, summing the cost of subtrees and costs of constraints with agents of higher priority (i.e. calculate the cost of the subtree rooted at this particular agent);
4. choose the value that minimises the lower bound on the costs and send this value to all neighbours of lower priority;
5. send the minimum lower and minimum upper bounds of the costs calculated in (3) to parent agent including the context to which they apply.

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

3 HANDLING COMPLEX LOCAL PROBLEMS

3.1 Decomposition

To apply the *decomposition* method: for each variable x_{ij} in each agent a_i, create a new virtual agent ax_{ij} to manage that variable. Thus, the problem is reduced to having a single variable per virtual agent. Agent a_i is then required to manage the activies of a set of virtual agents AX_i=$\{ax_{i1}, ax_{i2}, \ldots, ax_{im_i}\}$. Implemented in ADOPT [16], the communication between internal variables is done synchronously while the external communication is asynchronous.

3.2 Basic compilation

To apply the *basic compilation* method: (i) for each agent a_i, create a new variable z_i, whose domain $D_i = \prod_j D_{ij}$ is the set of local assignments to the agent's internal problem; (ii) for each agent a_i, add a unary constraint function f_i, where $\forall l \in D_i,\ f_i(l) = \sum_{j:s(c_j) \subseteq X_i} c_j(l_{\downarrow s(c_j)})$ (i.e. the cost is the sum of the costs from all constraints which act on a_i only); (iii) for each set of agents $A_j = \{a_{j1}, a_{j2}, ..., a_{jp_j}\}$, let $R_j = \{c : a(c) = A_j\}$ be the set of constraints whose agent scope is A_j, and for each $R_j \neq \phi$, define a new constraint $C_j : D_{j1} \times D_{j2} \times \ldots \times D_{jp_j} \rightarrow I\!N :: l \mapsto \sum_{c \in R_j} c(l_{\downarrow s(c)})$, equal to the sum of the constraints in R_j (i.e. construct constraints between the agents' new variables, that are defined by referring back to the original variables in the problem). Once compiled, we can run the ADOPT algorithm using the new variables. It is worth noting that a significant advantage that the compilation approach has over decomposition, is that it allows each agents local problem to be solved using existing efficient centralised solvers.

3.3 Improved compilation

An *improved compilation* method is presented in [5]. It proposes two advances on the basic compilation method.

First, the size of the compiled domain is reduced by finding only one optimal local solution for each combination of external variables (i.e. those variables of an agent that are connected to another agent). Only external variables can have a direct impact on other agents, therefore, dominated or fully interchangeable solutions in the local problems can be discarded. Formally, for each agent a_i, let $p_i = \{x_{ij} : \forall c\ x_{ij} \in s(c) \rightarrow s(c) \subseteq X_i\}$ be its *private* variables – the subset of its variables which are not directly constrained by other agents' variables – and let $e_i = X_i \setminus p_i$ be its *external* variables – the variables that do have direct constraints with other agents. For each a_i, create a new variable z'_i with domain $D'_i = \prod_{j:x_{ij} \in e_i} D_{ij}$, and add a function f'_i, where $\forall l \in D'_i,\ f'_i(l) = \min\{f_i(t) : t \in D_i, t_{\downarrow e_i} = l\}$. That is, D'_i contains all assignments to the external variables, and their cost is the minimum cost obtained when they are extended to a full local assignment for a_i. The new constraints are defined exactly as in the basic compilation (for each $R_j \neq \phi$, $C'_j : D'_{j1} \times D'_{j2} \times \ldots \times D'_{jp_j} \rightarrow I\!N :: l \mapsto \sum_{c \in R_j} c(l_{\downarrow s(c)}))$, but will act on smaller sets of tuples.

Second, the concept of Sub-neighbourhood interchangeability is introduced and applied to distributed search using compiled values. For each agent a_i and for a set of agents S, let $h_i^s = \{x_{ij} : \exists c : x_{ij} \in s(c) \wedge a(c) \cap S \neq \phi\}$ be the set of original variables of a_i that are adjacent to original variables of the agents in S. Two compiled values, $l_a, l_b \in D_i$ are *sub-neighbourhood interchangeable* (SNI) with respect to the agents in S ($l_a \equiv_i^s l_b$), if both values represent identical assignments to the variables of h_i^s: $l_a \equiv_i^s l_b \leftrightarrow l_{a \downarrow h_i^s} = l_{b \downarrow h_i^s}$ [1]. In ADOPT, agents send their values to all neighbours lower in the priority tree, and receive costs only from direct children. Considering each subtree separately, let S be the set of lower

[1] SNI works by exploiting symmetries that are introduced by the compilation technique - note that these symmetries will occur regardless of the problem domain.

priority neighbours of a_i, lying in the subtree rooted by a_s. For each subtree, partition a_i's values into SNI sets, such that $\Phi_i^s(x)$ is a function returning the SNI set to which x belongs: $\forall l_a, l_b \in D_i, l_a \equiv_i^s l_b \leftrightarrow \Phi_i^s(l_a) = \Phi_i^s(l_b)$. Then, ADOPT can be modified such that if a_i receives a cost ε from a_s with a compatible context, the costs of all values interchangeable with the current value x are updated: $\forall l \in \Phi_i^s(x), cost(l, a_s) \leftarrow \varepsilon$.

4 EXPERIMENTS

4.1 Experimental setup

We compare the decomposition, basic compilation and the improved compilation method (with and without SNI sets) on random binary graph colouring problems of varying complexity. The experiments are run in a simulated distributed environment: we use one machine but each agent runs asynchronously on its own thread. Both the basic and improved compilation are generated using Ilog JSolver [12]. This use of a 'black box' solver during compilation makes comparisons with the decomposition approach difficult. JSolver does not report constraint checks which means we are unable to use the preferred Concurrent Constraint Checks metric [13]. Instead, we compare both the number of messages communicated by the agents and the time taken to solve the problems to optimality (including compilation time for compilation approaches) [2]. A time limit of 3,600 seconds is used for each trial and the algorithm is terminated if this is exceeded. In such a case, we treat the finishing time as being 3,600 seconds and the number of messages as being 100,000 – a default 'high' value, which is used as we do not know how many messages would be required for the algorithm to terminate, and the number of messages passed at the point of reaching the cutoff may be inaccurate. All results are averaged over 20 test instances.

The problems are characterised according to the 5-tuple $\langle a, e, v, i, d \rangle$, where a is the number of agents, e is the external link density (# inter-agent constraints $= ea$), v is the number of variables per agent, i is the internal link density (# intra-agent constraints $= iv$) and d is the domain size of each variable. Each agent's local problem is generated separately, and then external links are added by randomly choosing variables from different agents to connect. We break down our experiments into three categories: the small problems in test set 1 allow us to compare all approaches; the more difficult problems in test set 2 are used to provide a deeper investigation into the decomposition and improved compilation techniques; and finally, in test set 3, selected tests are performed using the decomposition and improved compilation approaches on problems that are suited to these techniques.

4.2 Test set 1

Our first test cases consider small graph-colouring problems, where the basic setting is $\langle 5, 1, 6, 1, 2 \rangle$. We then compare each of the approaches varying v, i, a, d and e in turn. We chose these settings specifically to allow the basic compilation method to complete in reasonable time. Figures 1 and 2 show the results in log scale. *DECOMP* is the decomposition approach, *COMP* is the basic compilation, *IMP1* uses the improved compilation without SNI sets and *IMP2* uses the improved compilation with SNI sets. Points in the graph that reach the cutoff are filled in in black.

Figure 1 shows the number of messages that are passed during execution. For the smallest problems we can see that the basic compilation sometimes has fewer messages than the decomposition approach. However, as the number of internal variables v, internal link density i and domain size d increase, we see that the number of messages of *COMP* increases rapidly. The improved compilation approaches pass less messages in all tests, although as the domain size and external link densities are increased this advantage lessens.

An alternative view of the results can be gained by examining the total cpu time (Figure 2). Here we can see that the basic compilation performs poorly compared to the other approaches under all settings, even in cases where the number of messages is fewer. This would indicate that it is the internal computation of the agent that is the cause for the poor times in these cases. We see that the improved compilation approaches outperform the decomposition approach across all tests except for when the domain size increases. In particular, as the number of internal variables and internal link density increase (Figure 2 (a) and (b)) centralised optimization techniques can efficiently handle extra complexity in each agents local problem, leaving the external phase of the search unimpacted. This results in signifigant performance gains for both of the improved compilation methods. *IMP1* and *IMP2* also prove efficient as the number of agents increases. *IMP1* and *IMP2* also tend to be faster than *DECOMP* as external link density increases, although at the maximum ($e = 2$) *DECOMP* has become faster than *IMP1*. Finally, increasing the domain size increases the number of local solutions required for each agent resulting in a slow down in performance for the compilation approaches. For the decomposition approach in a graph colouring problem, increasing the domain size only makes the problem easier.

4.3 Test set 2

In this section we extend our analysis of the decomposition and improved compilation techniques to more difficult problems. We again study graph-coluring problems but this time with a basic setting of domain size 3 and internal and external densities of size 2 ($\langle 5, 2, 6, 2, 3 \rangle$). In these experiments, we again vary v, i, a, d and e in turn. The time limit remains at 3,600 seconds but in this case if the time limit is exceeded we set the number of messages to be 300,000. We do this as for these problems the number of messages exchanged is frequently higher than the previous limit of 100,000.

The patterns we saw in the smaller problems are visible once again here, but with some differences. During these experiments, the time cutoff was met much more frequently. With a default internal density of 2, the decomposition approach struggles to solve many of the problems. We see that it only performs well when the internal density was less than 2 (Figures 3, 4 (b)) and when the domain size was increased (Figures 3, 4 (d)) - i.e. when the problems were quite loose.

[2] We are interested in the longest execution sequence in wall-clock time, the balance of cpu time across all agents, and the total cpu time, but the simulation framework we use doesn't give us a simple way of reporting the first two. Therefore, we record here only the total cpu time as an indicative measure, and we are currently investigating how to adapt the framework to extract the necessary information.

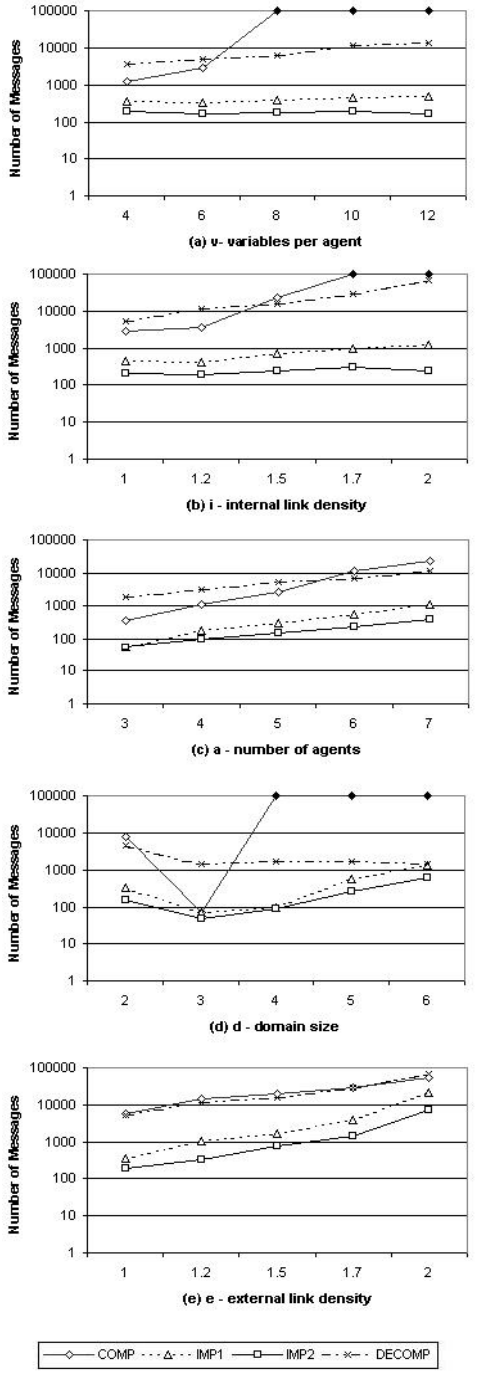

Figure 1. Test Set 1: $\langle 5, 1, 6, 1, 2 \rangle$. Number of messages (cutoff = 100,000).

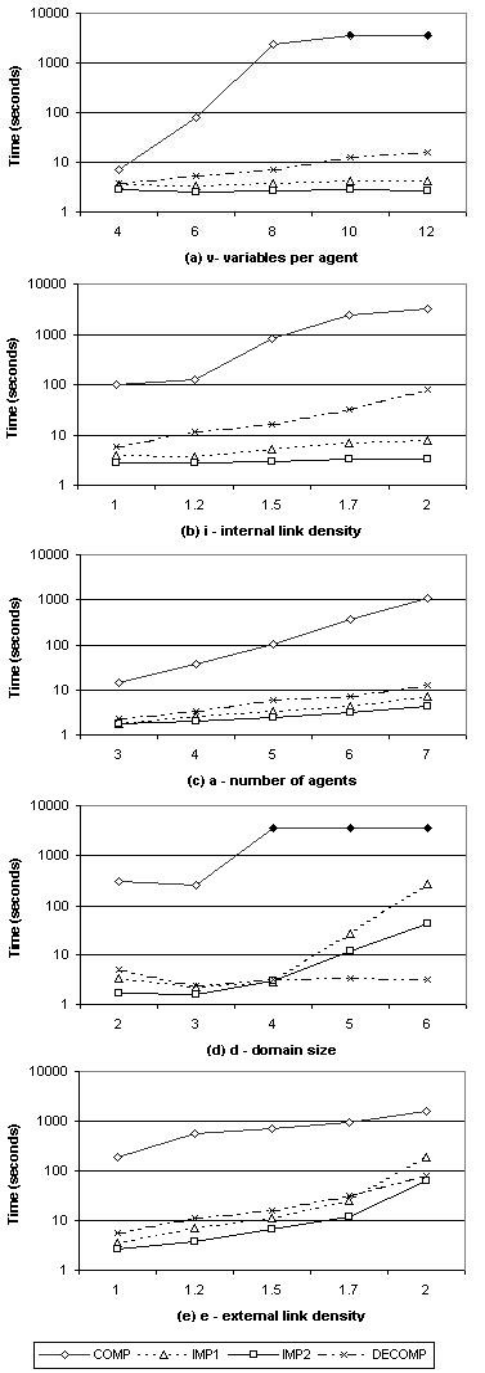

Figure 2. Test Set 1: $\langle 5, 1, 6, 1, 2 \rangle$. Time – seconds (cutoff = 3,600).

IMP1 and, in particular, *IMP2* cope better with these more difficult problems except for when the domain size increases. However, we notice that increased domain size does not result in a continuous worsening of performance. The increased internal computation required by *IMP1* and *IMP2* due to domain size increases is less of a factor in the context of these larger problems. Increasing the number of variables in each agent is shown again to have minimal effect on the improved compilation approaches. Increases in the other parameters including, somewhat surprisingly, internal link density, all result in decreasing performance, although still better than the decomposition approach.

4.4 Test set 3

Having identified the strengths and weaknesses of both the decomposition and improved compilation methods, we now choose problem settings suited to each approach and inves-

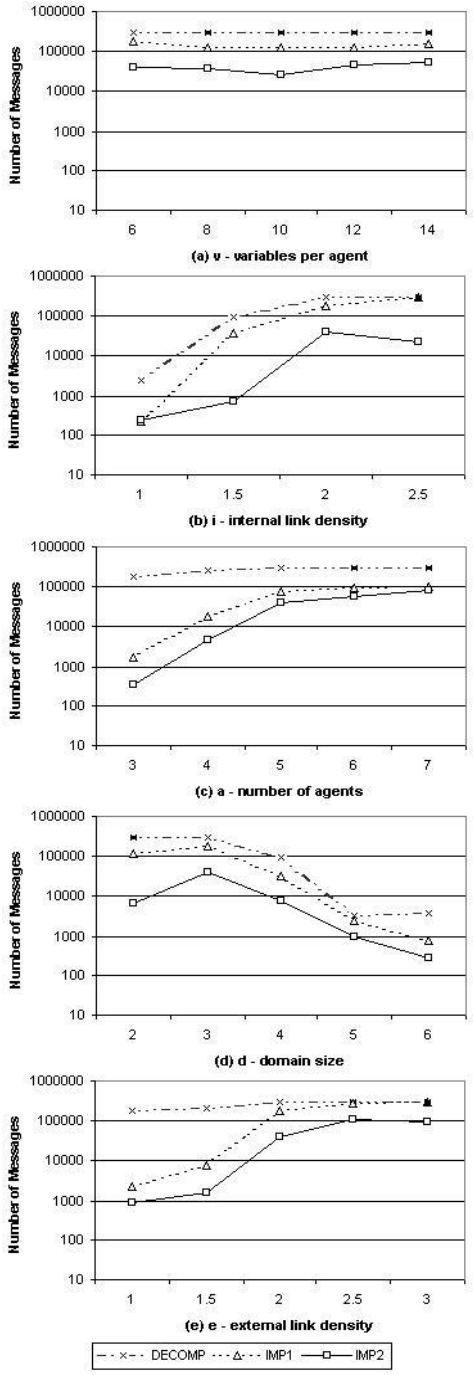

Figure 3. Test Set 2: $\langle 5, 2, 6, 2, 3 \rangle$. Number of messages (cutoff = 300,000).

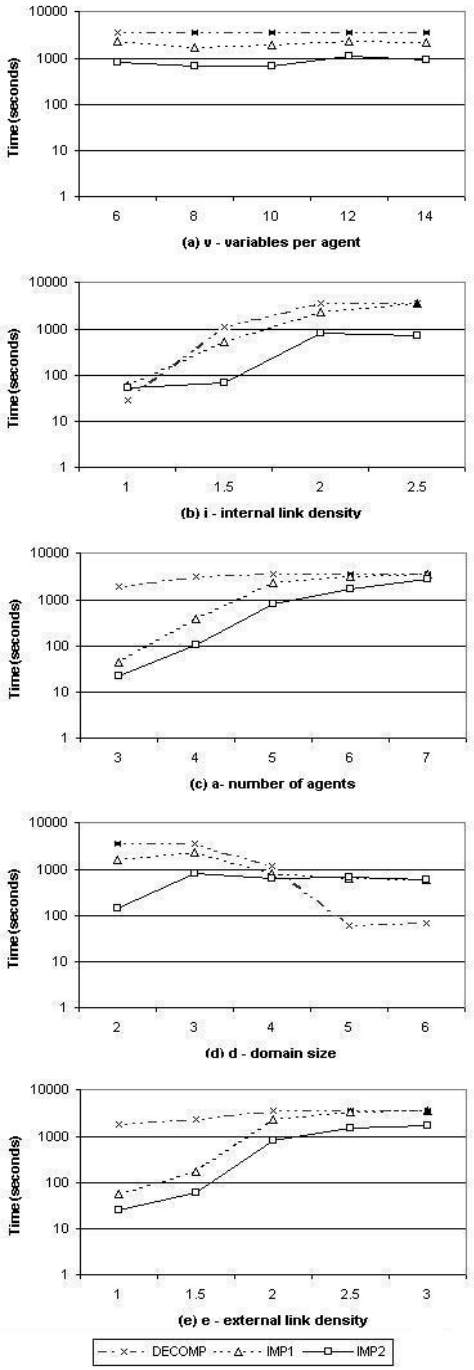

Figure 4. Test Set 2: $\langle 5, 2, 6, 2, 3 \rangle$. Time – seconds (cutoff = 3,600).

tigate further both the limitations and solving power of the techniques.

For the decomposition approach, we begin with settings of $\langle 5, 2, 6, 1, 3 \rangle$. We then test against combinations of external link density, e, from 2 to 6, and internal link density, i, from 1 to 2.5. Figures 5 and 6 shows that the decomposition approach can scale quite well as the external link density increases as long as the internal link density remains small. However, even a small increase in internal link density results in a rapid degradation of performance.

For the improved compilation approach with SNI sets, we use base settings of $\langle 5, 1, 10, 2, 3 \rangle$. We test against combinations of external link density, e, from 1 to 3, and internal link density, i, from 2 to 4. The results in Figures 7 and 8 demonstrate clearly that increasing the external link density has a negative effect while the internal link density has minimal effect, confirming our earlier findings. The effect of external link density on the results can be put down to two factors: (i) in-

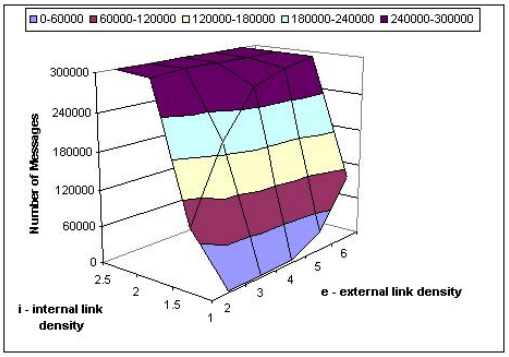

Figure 5. Number of messages for *DECOMP* varying the internal and external link densities: $\langle 5, e, 6, i, 3 \rangle$.

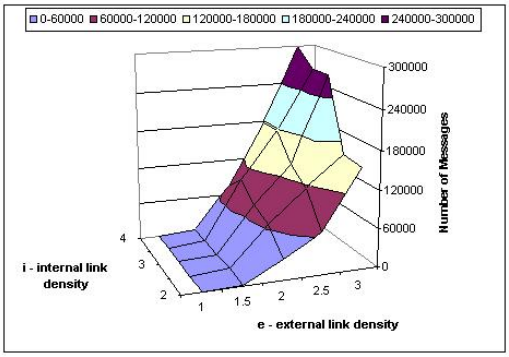

Figure 7. Number of messages for *IMP2* varying the internal and external link densities: $\langle 5, e, 10, i, 3 \rangle$.

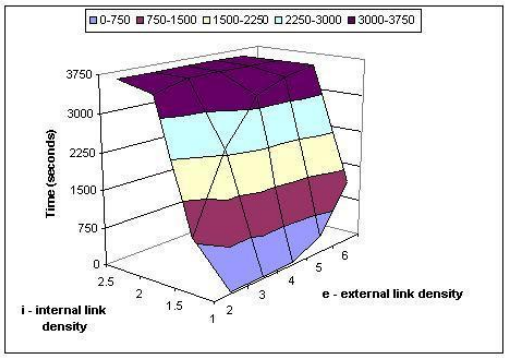

Figure 6. Execution time for *DECOMP* varying the internal and external link densities: $\langle 5, e, 6, i, 3 \rangle$.

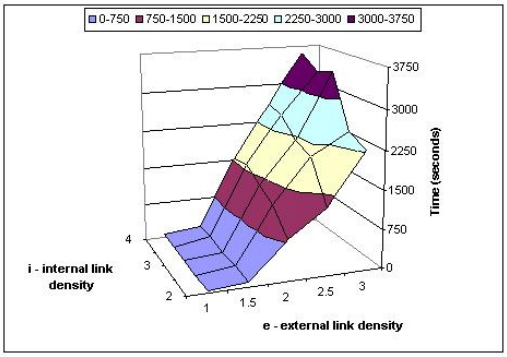

Figure 8. Execution time for *IMP2* varying the internal and external link densities: $\langle 5, e, 10, i, 3 \rangle$.

creasing external link density increases the overall complexity of the problem; and (ii) in random problems, increasing external link density increases the number of external variables that are present in each agent. This second point is important in that if all external links were concentrated on a small subset of an agent's variables, then the domain size of the compiled variables will be smaller and it is likely that performance would be better.

5 DISCUSSION

Both compilation and decomposition have their advantages and disadvantages. It may often be both desirable and natural for agents that have complex local problems to model these problems using centralised Constraint Programming (CP) [8]. This will allow the local solving process to benefit from many specialised techniques such as global constraints [19] and symmetry breaking [7]. The compilation method can use a centralised solver in this way to perform each agent's internal search, allowing problems where agents have large internal problems to be tackled. The drawback of the basic compilation method is that all local solutions have to be found before the distributed search can begin. In an optimisation problem, every set of assignments of values to variables is a valid solution. Therefore, for large internal problems, this process can be expensive and can result in very large domains of compiled values. The improved compilation technique tackles this problem and increases the range of DCOP problems that can be handled by just finding one solution for each combination of external variable. Unfortunately, it too can become expensive when the domain size and number of external variables of an agent becomes large. In the worst case, if all the agents' variables are external, the size of the problem using the improved compilation is equivalent to that of the basic compilation. Techniques for efficient handling of large domain sizes are required to offset this problem. Compilation also allows the use of SNI sets applied to compiled values, which we have seen can provide a significant speed up in the distributed search.

Using the decomposition approach agents' variables are kept in their original form and all the local solutions do not have to be pre-determined, however the benefits to be gained from using a centralised solver are lost. Our results have shown that this restriction can result in poor performance when the size and complexity of agents' internal problems grow. On the other hand, we have also seen that the decomposition approach can scale better as domain sizes and number of external variables increase. However, these benefits only remain if agents' internal problems remain small.

As each approach has its limitations, the scope for further research in this area is large. A number of approaches for handling multiple variables have been proposed for DCSP algorithms and, while these are not directly applicable to DCOP, they contain some useful concepts. Techniques similar to compilation have been used in [1, 11, 14] where centralised solvers

are used for solving each agents local problem, but local solutions are searched for on-demand as opposed to being precompiled. In [14], they also improve agent cooperation and reduce communication by trying to preserve the values of variables that are constrained with other agents – this is a concept that could also be applicable to DCOP algorithms. A decomposition based approach for handling multiple variables in DCSP is proposed in [22]. However, it involves dynamically re-prioritising variables which may be difficult to apply to DCOP as optimisation algorithms keep track of costs and bounds that are normally based on a static ordering of agents.

In this paper, we have evaluated agents with complex local problems using the ADOPT algorithm on distributed graph-colouring problems. As such, our findings are specific for this algorithm and problem domain. While we cannot be sure that these results can be generalised to other DCOP algorithms without a proper investigation, we feel it is likely that the core trends of the results are applicable.

6 CONCLUSION AND FUTURE WORK

DCOP algorithms generally require agents to have single variables. Two standard approaches to handling agents with complex problems are to either compile the local problem down to a single variable or to decompose the local problem by creating a virtual agent to represent each internal variable. We have compared both of these approaches with each other and with a third approach that improves on the compilation method by identifying interchangeable values that reduce both the size of the compiled problems and the search required to solve the compiled problems. We have evaluated the methods using the ADOPT algorithm on distributed graph colouring problems, and have identified the characteristics of problems that are suited for solving using the different approaches. We have shown that problems where agents have large, complex internal problems are best handled using the improved compilation technique, which outperforms the basic compilation technique in all cases and the decomposition technique in most cases. However, the usefulness of this technique decreases as the domain size of variables and the number of inter-agent constraints increases. The decomposition approach can successfully handle larger domain sizes, but requires the overall problem size to be reasonably small. We also show that the basic compilation technique is impractical for use in DCOP.

All experiments were run in a simulated distributed environment and it is future work to perform an experimental analysis on a properly distributed environment. We also hope to extend our analysis to consider different DCOP algorithms and problem domains.

ACKNOWLEDGEMENTS

We would like to thank the authors of the ADOPT algorithm, whose publicly available implementation [16] was used as a base upon which to do our work. We would also like to acknowledge the Boole Centre for Research in Informatics (BCRI) at UCC, whose computing resources were used to perform some of the experiments. This work is supported by Science Foundation Ireland under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain Research (CTVR).

REFERENCES

[1] A. Armstrong and E. Durfee, 'Dynamic prioritization of complex agents in distributed constraint satisfaction problems.', in *Proc. IJCAI*, pp. 620–625, (1997).

[2] R. Béjar, C. Domshlak, C. Fernàndez, C. Gomes, B. Krishnamachari, B. Selman, and M. Valls, 'Sensor networks and distributed csp: communication, computation and complexity.', *Artificial. Intelligence*, **161**(1-2), 117–147, (2005).

[3] C. Bessière, A. Maestre, I. Brito, and P. Meseguer, 'Asynchronous backtracking without adding links: a new member in the ABT family', *Artificial Intelligence*, **161**, 7–24, (2005).

[4] D.A. Burke and K.N. Brown, 'Applying interchangeability to complex local problems in distributed constraint reasoning', in *Proc. Workshop on Distributed Constraint Reasoning, AAMAS*, (2006).

[5] D.A. Burke and K.N. Brown, 'Efficient handling of complex local problems in distributed constraint optimization.', in *Proc. ECAI*, (2006). To appear.

[6] M. Calisti and N. Neagu, 'Constraint satisfaction techniques and software agents', in *Proc. Agents and Constraints Workshop, AIIA*, (2004).

[7] D. Cohen, P. Jeavons, C. Jefferson, K. E. Petrie, and B. M. Smith, 'Symmetry definitions for constraint satisfaction problems.', in *CP*, pp. 17–31, (2005).

[8] R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[9] A. Gershman, A. Meisels, and R. Zivan, 'Asynchronous forward-bounding for distributed constraints optimization', in *Proc. Workshop on Distributed and Speculative Constraint Processing, CP*, (2005).

[10] Y. Hamadi, 'Interleaved backtracking in distributed constraint networks.', *International Journal on Artificial Intelligence Tools*, **11**(2), 167–188, (2002).

[11] K. Hirayama and M. Yokoo, 'Local search for distributed sat with complex local problems', in *Proc. AAMAS*, pp. 1199–1206, (2002).

[12] ILOG. Jsolver, 2005. http://www.ilog.com/products/jsolver.

[13] A. Meisels I. Razgon E. Kaplansky and R. Zivan, 'Comparing performance of distributed constraints processing algorithms', in *Proc. Workshop on Distributed Constraint Reasoning, AAMAS*, pp. 86–93, (2002).

[14] A. Maestre and C. Bessière, 'Improving asynchronous backtracking for dealing with complex local problems.', in *Proc. ECAI*, pp. 206–210, (2004).

[15] R. Mailler and V. Lesser, 'Solving distributed constraint optimization problems using cooperative mediation', in *Proc. AAMAS*, pp. 438–445, (2004).

[16] P. Modi. Adopt algorithm homepage, 2005. http://www.cs.cmu.edu/~pmodi/adopt/.

[17] P. Modi, W. Shen, M. Tambe, and M. Yokoo, 'Adopt: Asynchronous distributed constraint optimization with quality guarantees', *Artificial Intelligence*, **161**(1–2), 149–180, (2005).

[18] A. Petcu and B. Faltings, 'A scalable method for multiagent constraint optimization.', in *Proc. IJCAI*, pp. 266–271, (2005).

[19] J-C. Regin, *Constraints and Integer Programming Combined*, chapter Global Constraints and Filtering Algorithms, Kluwer, 2003.

[20] M. Silaghi, D. Sam-Haroud, and B. Faltings, 'Asynchronous search with aggregations', in *Proc. AAAI/IAAI*, pp. 917–922, (2000).

[21] R. Wallace and E. Freuder, 'Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving.', *Artificial. Intelligence.*, **161**(1-2), 209–227, (2005).

[22] M. Yokoo and K. Hirayama, 'Distributed constraint satisfaction algorithm for complex local problems', in *Proc. ICMAS*, p. 372, (1998).

[23] M. Yokoo and K. Hirayama, 'Algorithms for distributed constraint satisfaction: A review', *Autonomous Agents and Multi-Agent Systems*, **3**(2), 185–207, (2000).

[24] R. Zivan and A. Meisels, 'Dynamic ordering for asynchronous backtracking on discsps.', in *Proc. CP*, pp. 32–46, (2005).