14-11-2005

Practical application of support-based distributed search

Peter Harvey
pah06@uow.edu.au

C. F. Chang
University of Wollongong, cfc@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

Practical application of support-based distributed search

Abstract

Algorithms for Distributed Constraint Satisfaction Problems have tended to mirror existing non-distributed global-search or local-search algorithms. Unfortunately, existing distributed global-search algorithms derive from classical backtracking search methods and require a total ordering over variables for completeness. Distributed variants of local-search algorithms (such as distributed breakout) inherit the incompleteness properties of their predecessors. A meeting scheduling problem translates to a DisCSP where a global ordering is difficult to maintain and creates undesirable behaviours. We present a practical demonstration of an algorithm in which a global ordering is not required, while avoiding the problems of local-search algorithms.

Disciplines

Physical Sciences and Mathematics

Publication Details

This article was originally published as: Harvey, P., Chang, C. F. & Ghose, A., Practical application of support-based distributed search, 17th IEEE International Conference on Tools with Artificial Intelligence, 14-16 November 2005, 34-38. Copyright IEEE 2005.

Practical Application of Support-based Distributed Search

Peter Harvey Chee Fon Chang Aditya Ghose
Decision Systems Laboratory, University of Wollongong, NSW, Australia
pah06@uow.edu.au c03@uow.edu.au aditya@uow.edu.au

Abstract

Algorithms for Distributed Constraint Satisfaction Problems have tended to mirror existing non-distributed global-search or local-search algorithms. Unfortunately, existing distributed global-search algorithms derive from classical backtracking search methods and require a total ordering over variables for completeness. Distributed variants of local-search algorithms (such as distributed breakout) inherit the incompleteness properties of their predecessors.

A meeting scheduling problem translates to a DisCSP where a global ordering is difficult to maintain and creates undesirable behaviours. We present a practical demonstration of an algorithm in which a global ordering is not required, while avoiding the problems of local-search algorithms.

1 Introduction

Constraint Satisfaction Problems (CSPs) have proven applicable in a wide variety of domains. A CSP is classically defined by a set of variables $\mathcal{V}$, a domain for each variable $\mathcal{D}_v$, and a set of constraints $\mathcal{C}$. A *solution* to a CSP is a complete assignment of values to variables satisfying every constraint.

A Distributed CSP (DisCSP) is formed when the description and solution procedure of a CSP are separated amongst multiple agents. The distributed environment extends the applicability of CSPs to domains such as distributed scheduling and resource contention. Of particular interest is the use of DisCSPs as models for solving other multiagent problems, with DisCSP algorithms defining a procedure for conflict resolution. Common examples include scheduling, task assignment, and limited forms of negotiation where simple decision(s) must be made per agent. A DisCSP can be constructed by representing each of the agents decisions as a variable, with constraints describing any inter-agent relationships.

A DisCSP can be solved by distributed variants of existing global-search or local-search algorithms. However, local-search algorithms [7] are incomplete in both the distributed and non-distributed case. Distributed variants of global-search [1, 2, 5, 6] presented to date make use of a total order over variables.

We argue that any total order impacts the characteristics of backtracking-style search in undesirable ways for use in many multiagent problems. For example, an agent which has a 'higher' rank in the ordering has more 'authority' and is less likely to change value than a 'lower' ranking agent. In an anytime environment this results in higher-ranked agents being granted more stable answers. While some problems may desire such behaviour our concern lies with those problems which do not.

We also argue that it is difficult to add constraints between two previously independent DisCSPs when using a total order. To do so would require a re-computation of the variable ordering and/or an arbitrary decision that the first DisCSP ranks higher than the second DisCSP. If a problem is frequently altered by addition of groups of variables, as is likely to occur in large DisCSP networks, global re-computation will become increasingly difficult. If variable ordering is instead made arbitrarily (for example, ordering by variable identifier) the problem of solution stability for individual agents is exacerbated.

The validity of these arguments become evident in meeting scheduling problems.

Example. The universities of Pluto and Saturn each use an automated system for scheduling meetings amongst their own staff. Staff give constraints of the form 'Alice needs to meet with Bob for 2 hours this Wednesday or Thursday' to agents on their own computers. Individual universities contain a large number of staff with generally sparse connections, so a distributed algorithm is used in which agents communicate directly with each other. Agents are assigned comparable identifiers using finely-tuned schemes specific to each university. These identifiers are chosen to permit backtracking in a distributed global-search algorithm within the university.

Despite best efforts at 'fairness of effort' a static ordering creates problems between research peers. For example, any

Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)
1082-3409/05 $20.00 © 2005 IEEE

trivial change in Alice's meeting times must always be accepted by Bob. Inversely, Bob may request a change to Alice's meetings only after exhausting all possible meeting schedules and detecting infeasibility. This issue is distinct from that of preference orderings, and instead relates to localised solution stability; Alice's schedules are more stable than Bob's.

Worsening matters, Bob at Pluto wishes to arrange a meeting with Carla at Saturn. Their identifiers, while still possibly unique, are not meaningfully comparable for the purpose of backtracking search. To continue using any existing distributed algorithm we must be able to compare identifiers between agents operating at Pluto and Saturn universities. An example solution is to decide that all Saturn identifiers are 'greater' than Pluto identifiers. Unfortunately this would have the same impact on the behaviour of the algorithm as outlined above - meeting schedules for researchers at Pluto would become subservient to those at Saturn. Any changes in meeting times for Saturn researchers, no matter how trivial, must be accepted by Pluto researchers.

Furthermore, any decision for resolving the variable order would require intervention by authorities at each university or the use of a heuristic method such as DisAO [1, 2]. While this decision could be made for pairs or sets of universities, it does not scale well computationally. For example, if Dennis was an independent researcher he must establish 'comparability' with each university and all other researchers. The addition of new researchers frequently raises the possibility of frequent re-computation of variable ordering.

We have highlighted problems that arise from using a total order to establish 'authority' or 'importance' between agents, and maintaining a total order subject to merging of previously independent DisCSPs. The computational disadvantages of a total order were also noted and addressed in the development of Asynchronous Weak-Commitment Search (AWCS) [5, 6]. However, AWCS creates additional links between variables and assumes that nogoods are stored indefinitely; both properties we believe are undesirable in large-scale meeting scheduling. The specific difficulties of large-scale distributed meeting scheduling motivated the development of Support-based Distributed Search (SBDS) [3], which:

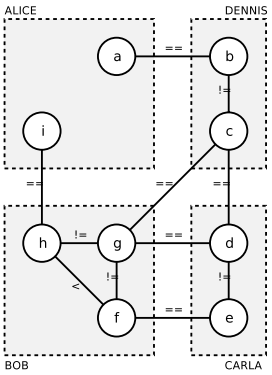
- has no need for 'authority' between variables, effectively avoiding the need for a total order on variables.
- provides fairness in the level of stability for variables.
- does not add links between variables, avoiding the eventual need for 'broadcasting' assignments.
- addresses the risk of cyclic behaviour exhibited by local search algorithms.

We will present a simple meeting scheduling problem and demonstrate how SBDS would solve it.

2 Application of SBDS

We first demonstrate the translation of a meeting scheduling problem to a suitable DisCSP. Consider the following meeting requirements for Alice, Bob, Carla and Dennis.

- Available times are 1pm, 2pm and 3pm.
- Bob, Carla and Dennis have a group meeting.
- Bob must meet separately with Carla.
- Bob must meet Alice before his meeting with Carla.
- Dennis must meet with Alice.
- Double-booking for meeting time-slots is not allowed, except in Alice's case.



Constraint Graph

$$\mathcal{V} = \{a, b, c, d, e, f, g, h, i\}$$

$$\mathcal{D} = \{1\text{pm}, 2\text{pm}, 3\text{pm}\}$$

$$\mathcal{C} = \left\{ \begin{array}{ll} a = b & b \neq c \\ c = g & d \neq e \\ e = f & f \neq g \\ g \neq h & h = i \\ c = d & d = g \\ f > h & \end{array} \right\}$$

Constraint Model

Figure 1. Example meeting scheduling CSP

Figure 1 describes a constraint model and resulting constraint graph for this problem. We translate the time of attending a meeting for each person into a variable, distributed according to person. Equality constraints are used to ensure meeting times are agreed to by all users. For example, both variables a and b are the scheduled time of the meeting between Alice and Dennis. The constraint $a = b$ is interpreted as 'the time Alice decides to meet with Dennis must be the same as the time that Dennis decides to meet with Alice'. Inequality constraints ensure that meetings are scheduled at distinct times.

Given the constraint network in Figure 1 multiple distributions of variables to agents are possible. For simplicity we assume that each variable is represented as a separate agent, though in implementation their actions would be simulated with one agent per user. Agents have control only over the assignment of value to their own variable and are connected to neighbours according to the constraint graph.

A valid solution is reached when all variables are assigned values satisfying the constraints. An example of a valid solution is $a = b =$ 1pm, $c = d = g =$ 2pm, $e = f =$ 3pm, and $h = i =$ 1pm.

Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)
1082-3409/05 $20.00 © 2005 IEEE

If Alice, Bob, Carla and Dennis were to solve this problem manually we would expect a series of arguments for and against possible schedules. They require no sense of 'authority' - each proposal is viewed independently of the person stating it and judged on it's merits against current proposals. Proposals may just be partial schedules, and be communicated to only interested parties. For example:

Alice says to Bob - I propose we meet at 2pm

Alice says to Dennis - I propose we meet at 1pm

Carla says to Bob - I propose we meet (separately) at 1pm

Dennis says to Carla and Bob - I propose we meet as a group at 2pm

Bob says to Alice - I propose we meet at 1pm instead as I have a group meeting at 2pm

Bob says to Carla - I reject your proposed meeting time and I propose we meet at 3pm

This dialogue consists primarily of proposals, beginning with Alice's initial proposal to Bob of a 2pm meeting time. Note that Bob also utilised the proposal from Dennis to construct a counter-proposal to overpower Alice's proposal. The other form of communication in this dialogue was 'rejection'. Bob also flatly rejected Carla's proposal on the basis that 'no possible meeting time with Alice would be possible', though this reason was not given to Carla. These forms of communication are similar to those in [4].

Support-based Distributed Search makes use of this model of communication to solve the DisCSP formulation of this problem; agents establish mutually acceptable assignments by the exchange of *arguments*. Any counter-argument must be 'stronger', and to this end arguments received from neighbours can be used to strengthen an agent's own arguments.

Support-based Distributed Search defines arguments as:

- an infeasible assignment to a set of variables, or;
- a locally feasible assignment to a sequence of variables (where each variable is connected by a constraint to its immediate predecessor).

The first is known within constraint satisfaction literature as a *nogood*. This is equivalent to 'rejection', and is always (and only) generated if an agent finds that there is no possible value it can take while certain other assignments are in effect. The only option when a nogood has been received by an agent is to change its current assignment, or to 'pass-the-buck' and generate a nogood for another agent.

The second type of argument we term an *isgood*. This is equivalent to 'proposal', and is generated by an agent in an attempt to convince other agents to take upon a feasible assignment. A neighbour will accept an isgood if it does not conflict with their current assignment, or is 'strong' enough to convince them to change. A neighbour will reject an isgood if it is able to counter with a nogood or a stronger isgood.

The following describes the reasoning used by each agent in SBDS:

1. Receive any nogoods (rejections) and the latest isgood from each neighbour.
2. If a received isgood is found inconsistent (due to our constraints and learned nogoods) respond with a nogood and discard the isgood.
3. Choose a strong isgood (received from a neighbour) and record that neighbour as our 'support'.
4. Send stronger isgoods to neighbours with conflicting values by extending the isgood from our support.
5. Goto 1.

A stable assignment (meeting schedule) is established when no neighbours hold conflicting values.

We note that our original human dialogue could have been generated by following such an algorithm. For example, Bob receives a proposal (isgood) from Carla which is inconsistent with his constraints, and responds with a rejection (nogood). Similarly, Bob chose Dennis as 'support' and chose his value accordingly. The proposal from Dennis was used to bolster Bob's proposal to Alice.

We will now describe the details of isgoods and nogoods with examples from our meeting scheduling problem.

2.1 Building isgoods

An isgood is written as a non-empty sequence of triples (v, d, n), where $v \in \mathcal{V}$ is a variable, $d \in \mathcal{D}_v$ is the value assigned to that variable, and $n \in \mathbb{Z}^+$ is the number of values in $\mathcal{D}_v$ which were eliminated by preceding assignments to other variables. Each variable occurs at most once in an isgood, and the assignments described must satisfy all constraints defined over those variables.

Example. A partial schedule of $b = $ 1pm and $c = $ 2pm, written in that order, would be represented as an isgood $\langle(b, 1\text{pm}, 0), (c, 2\text{pm}, 1)\rangle$ which indicates that:

- b took value 1pm, and no alternatives were eliminated.
- c took value 2pm, and one alternative value (1pm) had already been eliminated by the preceding assignment of 1pm to b.

Similarly, an assignment of $b = $ 1pm and $a = $ 1pm would be represented as an isgood $\langle(b, 1\text{pm}, 0), (a, 1\text{pm}, 2)\rangle$, as all alternative values for a were eliminated by the preceding assignment of b. Finally, the assignment $a = $ 1pm, $b = $ 1pm and $c = $ 2pm is represented as $\langle(a, 1\text{pm}, 0), (b, 1\text{pm}, 2), (c, 2\text{pm}, 1)\rangle$.

2.2 Rating isgoods

It is often the case that an agent receives isgoods from multiple neighbours and is unable to choose a value consistent with all of them. The agent must choose one neighbour as 'support' and use the most recently received isgood to argue against the values of other neighbours. To be able to 'argue against' other neighbours in a consistent fashion we require a scheme to allow all agents to determine the strength of an isgood.

We define a function str which calculates the strength of an isgood in a recursive fashion. Intuitively it can be regarded as 'a measure of the size of the search space which was exhaustively tested to arrive at this assignment'. We will use $\langle I, (v, d, n)\rangle$ to denote the isgood formed by taking an existing isgood I and appending (v, d, n).

$$\begin{aligned} str(\langle\,\rangle) &= 0 \\ str(\langle I, (v, d, n)\rangle) &= str(I) \times |\mathcal{D}_v| + n + 1 \end{aligned}$$

2.3 Building nogoods

A nogood is written as a set of pairs (v, d) where $v \in \mathcal{V}$ is a variable and $d \in \mathcal{D}_v$. For example, the nogood $\{(g, 3\text{pm}), (h, 2\text{pm})\}$ indicates that the simultaneous assignment of $g = 3\text{pm}$ and $h = 2\text{pm}$ is inconsistent. A nogood may only be used to counter an isgood and must only contain a subset of those assignments described in the isgood.

Example. The assignment $g = 3\text{pm}$ and $h = 2\text{pm}$ would be written as an isgood $\langle(g, 3\text{pm}, 0), (h, 2\text{pm}, 1)\rangle$. However this does not permit an assignment to f. Thus f may send a nogood to h (the last assigned variable) of $\{(g, 3\text{pm}), (h, 2\text{pm})\}$.
This nogood still permits the assignment to g, and permits an alternative assignment for h. In this situation, after receiving the nogood, h may respond with a new isgood $\langle(g, 3\text{pm}, 0), (h, 2\text{pm}, 2)\rangle$.

2.4 Execution

We have implemented and tested the SBDS algorithm, and are able to execute it on our example meeting scheduling problem. A simplified log of communication between agents is shown in Figure 2. Note that, for clarity, we have presented the execution as a series of iterations, with no two neighbours able to emit arguments within the same iteration. This is a stronger condition than required by SBDS.

When reading Figure 2 note that SBDS differs from other algorithms through the lack of a variable ordering, and the structured way in which 'strength' for arguments is increased. This provides the qualitative improvements that are useful for larger meeting scheduling problems.

3 Conclusion

The ability to function in a distributed environment extends the applicability of CSPs to new domains. In many cases a DisCSP can be solved by distributed variants of existing global-search or local-search algorithms. However, we have presented a problem for which such distributed variants of existing algorithms are not suitable. This motivated the development of SBDS, designed specifically for distributed environments.

In Figure 2 we have presented the results of applying one such algorithm (SBDS) to a small but practical distributed problem. It is important to observe the behaviour of SBDS in this instance as representative of a new breed of algorithms. Such algorithms:

- have no total order on variables, so no variable assignment is made arbitrarily more stable than any other.
- do not add links between variables, ensuring that weakly connected problems remain so.
- avoid cyclic behaviour [7] known to plague other distributed search algorithms by incrementing the information transmitted over time.

When we extrapolate the behaviour of SBDS to extremely large meeting scheduling problems the advantages of this approach become apparent. For example, in a deployed system using such an algorithm:

- after adding an agent or constraint, the algorithm can continue uninterrupted.
- after removing an agent or constraint, it is possible for isgoods can be updated and nogoods retracted with no further disturbance of the current system state.
- a collection of separate scheduling systems may be connected and disconnected easily in an ad-hoc manner.

We believe these attributes are of practical importance for a widely-deployed distributed scheduling system. Any system which does not elegantly handle ad-hoc modifications to the problem at hand (including large-scale modifications such as connecting previously separate problems) will suffer as the number of involved agents increases.

References

[1] C. Bessière, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In T. Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, page 772. Springer, 2001.

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

From	To	Argument	Strength
a	b	$\langle(a, 1\text{pm}, 0)\rangle$	1
c	b, d, g	$\langle(c, 1\text{pm}, 0)\rangle$	1
e	d, f	$\langle(e, 1\text{pm}, 0)\rangle$	1
h	f, g, i	$\langle(h, 1\text{pm}, 1)\rangle$	2
b	c	$\langle(a, 1\text{pm}, 0), (b, 1\text{pm}, 2)\rangle$	6
d	e	$\langle(c, 1\text{pm}, 0), (d, 1\text{pm}, 2)\rangle$	6
d	c, g	$\langle(d, 1\text{pm}, 0)\rangle$	1
f	e	$\{(e, 1\text{pm})\}$	-
f	e, g, h	$\langle(f, 2\text{pm}, 0)\rangle$	1
i	h	$\langle(i, 1\text{pm}, 0)\rangle$	1
e	d, f	$\langle(e, 2\text{pm}, 1)\rangle$	2
g	c, d, f	$\langle(g, 1\text{pm}, 0)\rangle$	1
g	h	$\langle(d, 1\text{pm}, 0), (g, 1\text{pm}, 2)\rangle$	6
h	g	$\langle(h, 2\text{pm}, 1)\rangle$	2
h	i, f	$\langle(g, 1\text{pm}, 0), (h, 2\text{pm}, 2)\rangle$	6
c	b	$\langle(c, 2\text{pm}, 0)\rangle$	1
c	d, g	$\langle(b, 1\text{pm}, 0), (c, 2\text{pm}, 1)\rangle$	5
g	c	$\langle(g, 2\text{pm}, 0)\rangle$	1
g	d, f	$\langle(c, 2\text{pm}, 0), (g, 2\text{pm}, 2)\rangle$	6
g	h	$\langle(b, 1\text{pm}, 0), (c, 2\text{pm}, 1), (g, 2\text{pm}, 2)\rangle$	18
i	h	$\langle(i, 2\text{pm}, 0)\rangle$	1
h	g	$\langle(h, 1\text{pm}, 1)\rangle$	2
h	i, f	$\langle(c, 2\text{pm}, 0), (g, 2\text{pm}, 2), (h, 1\text{pm}, 2)\rangle$	21
d	c, e	$\langle(g, 2\text{pm}, 0), (d, 2\text{pm}, 2)\rangle$	6
d	g	$\langle(d, 2\text{pm}, 2)\rangle$	3
i	h	$\langle(i, 1\text{pm}, 0)\rangle$	1
f	e, g	$\langle(h, 1\text{pm}, 0), (f, 3\text{pm}, 1)\rangle$	5
e	f	$\langle(d, 2\text{pm}, 0), (e, 3\text{pm}, 1)\rangle$	5
e	d	$\langle(e, 3\text{pm}, 0)\rangle$	1

Solved: $a = b = h = i = 1\text{pm}$, $c = d = g = 2\text{pm}$, $e = f = 3\text{pm}$.

Notes

In the first iteration a, c, e and h simultaneously take the value 1pm and send isgoods. Due to the ordering constraint h can eliminate 3pm and this is reflected in elimination counts.

In the second iteration b and d counter with stronger isgoods to c and e respectively. Also f sends a nogood to e as the ordering constraint disallows a value of 1pm. Due to the nogood, f need not present a stronger isgood to e.

In the third iteration g counters with a stronger isgood to h. The additional strength comes using just part of the isgood that g received from d.

In the fourth iteration h changes value to 2pm on the basis of g's isgood. The isgood to i and f must be stronger than the isgood h sent last time.
Also c presents an isgood to g using support from b.

In the fifth iteration g changes values to 2pm due to c's isgood. Simultaneously i changes value to 2pm on the basis of h's isgood.

This state does not last long as the change initiated by b propagates. In the sixth iteration h changes value back to 1pm, again on the basis of g's isgood. The isgood from g is also accepted and used by d to choose a value of 2pm.

Finally, the remaining variables i, e and f accept the arguments of their neighbours. With no remaining conflicts the algorithm terminates (the agents idle).

Figure 2. Example execution of SBDS on the given meeting scheduling problem

[3] P. Harvey, C. F. Chang, and A. Ghose. Support-based Distributed Search. In *Distributed Constraint Reasoning Workshop*, 2005. http://www.dsl.uow.edu.au/people/harvey/dcr05.pdf.

[4] I. Rahwan, S. D. Ranchurn, N. R. Jennings, P. McBurnery, S. Parsons, and L. Sonenberg. Argumentation-based Negotiation. *Knowledge Engineering Review*, 18(4):343–375, 2004.

[5] M. Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In U. Montanari and F. Rossi, editors, *CP*, volume 976 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 1995.

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

[7] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *AAAI/IAAI*, pages 352–, 2002.