Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)

Saving Redundant Messages in BnB-ADOPT *

Patricia Gutierrez and **Pedro Meseguer**
IIIA, Institut d'Investigació en Intel.ligència Artificial
CSIC, Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Spain.
patricia|pedro@iiia.csic.es

Abstract

We have found that some messages of BnB-ADOPT are redundant. Removing most of those redundant messages we obtain BnB-ADOPT$^+$, which achieves the optimal solution and terminates. In practice, BnB-ADOPT$^+$ causes substantial reductions on communication costs with respect to the original algorithm.

BnB-ADOPT (Yeoh, Felner, and Koenig 2008) is a reference algorithm for distributed constraint optimization (DCOP), defined as follows. There is a finite number of agents, each holding one variable that can take values from a finite and discrete domain, related by binary cost functions. The cost of a variable assigning a value is the sum of cost functions evaluated on that assignment. The goal is to find a complete assignment of minimum cost by message passing (for details on DCOP definition see (Modi et al. 2005)).

BnB-ADOPT is a depth-first version of ADOPT (Modi et al. 2005), showing a better performance. As ADOPT, it arranges agents in a DFS tree. BnB-ADOPT messages are VALUE(i, j, val, th), –i informs child or pseudochild j that it has taken value val with threshold th–, COST$(k, j, context, lb, ub)$ –k informs parent j that with $context$ its bound are lb and ub–, and TERMINATE(i, j). –i informs child j that i terminates–. A BnB-ADOPT agent executes the following loop: it reads and processes all incoming messages, and takes value. Then, it sends the following messages: a VALUE per child, a VALUE per pseudochild and a COST to its parent. BnB-ADOPT contexts can be updated by VALUEs or COSTs, while in ADOPT contexts are updated by VALUEs only. This is due to timestamps that go with individual values allowing to determine which is more recent (timestamps are called counters referred as ID in (Yeoh, Felner, and Koenig 2008)). Here, we assume that the reader has some familiarity with BnB-ADOPT code.

We show that some BnB-ADOPT messages are redundant. Removing most of those redundant messages we obtain BnB-ADOPT$^+$, keeping optimality and termination. BnB-ADOPT$^+$ causes substantial reductions on communication costs, dividing by a factor from 2 to 6 the number of messages (experimental testing on several benchmarks).

*Partially supported by Spanish proj. TIN2009-13591-C02-02.

Redundant Messages

In the following i, j and k are agents executing BnB-ADOPT. Agent i, holding variable x_i, *takes value* v when the assignment $x_i \leftarrow v$ is made and i informs of it to its neighbors. The *state* of i is defined by (1) its value, (2) its context (values of agents located before i in its branch, timestamps are not part of the context), and (3) for each possible value v and each $j \in children(i)$, the lower and upper bounds $lb(v, j)$/$ub(v, j)$. A message msg sent from i to j is *redundant* if at some future time t, the collective effect of other messages arriving j between msg and t would cause the same effect, so msg could have been avoided.

Lemma 1 *If i takes value v_1 with timestamp t_1, and the next value it takes is v_2 (possibly equal to v_1) with timestamp t_2, there is no message with timestamp t for i st. $t_1 < t < t_2$.*

Proof. No VALUE is sent from i with timestamp bewteen t_1 and t_2: v_1 and v_2 are consecutive. COSTs build their contexts from VALUEs: no VALUE includes a timestamp between t_1 and t_2, so no COST will contain it for i. □

Theorem 1 *If i sends to j two consecutive VALUEs with the same val, the second message is redundant.*

Proof. Let V_1 and V_2 be two consecutive VALUEs sent from i to j with the same value val with timestaps t_1 and $t_2, t_1 < t_2$. When V_1 reaches j, it may happen:

1. V_1 does not update $context_j[i]$. When V_2 arrives: (a) V_2 does not update $context_j[i]$. Future messages will be processed as if V_2 would have not been received, so V_2 is redundant. (b) V_2 updates $context_j[i]$ which has timestamp t. Either (i) $t_2 > t > t_1$ or (ii) $t_2 > t = t_1$; (i) is impossible because Lemma 1; (ii) since $t = t_1$ the value in V_2 is already in $context_j[i]$. Every future message accepted with timestamp t_2 of $context_j[i]$ would also be accepted if timestamp were t_1. Since Lemma 1, V_2 is redundant.
2. V_1 updates $context_j[i] \leftarrow val$, timestamp t_1. When V_2 arrives: (a) V_2 does not update $context_j[i]$: as case (1.a). (b) V_2 updates $context_j[i]$: since V_1 updated $context_j$ and Lemma 1, the timestamp of $context_j[i]$ must be t_1. Updating with V_2 does not change $context_j[i]$ but its timestamp is put to t_2. Since there are no messages with timestamp between t_1 and t_2 (Lemma 1), any future message that could update $context_j$ with t_2 would also update it with t_1. So V_2 is redundant. □

(a) Random DCOPs

p_1	#Messages	#NCCC	#Cycles
	1,393,339	11,002,964	**53,065**
0.4	**657,714**	**10,827,544**	53,074
	68,116,304	508,186,224	**1,987,584**
0.6	**24,809,153**	**499,214,418**	1,987,915
	184,735,389	1,366,404,208	4,740,277
0.7	**59,900,198**	**1,339,303,291**	**4,740,040**
	293,922,594	2,153,776,854	**6,873,799**
0.8	**86,233,163**	**2,112,858,127**	6,873,805

(b) Meeting Scheduling

	#Messages	#NCCC	#Cycles
	96,493	697,774	**4,427**
A	**35,767**	**690,786**	**4,427**
	182,652	879,417	**7,150**
B	**69,453**	**801,384**	**7,150**
	34,374	167,058	**1,278**
C	**13,862**	**157,995**	**1,278**
	47,729	155,833	**1,733**
D	**20,386**	**141,816**	**1,733**

(c) Sensor Network

	#Messages	#NCCC	#Cycles
	7,040	15,097	**226**
A	**1,074**	**14,514**	**226**
	10,258	23,597	**320**
B	**1,859**	**22,659**	**320**
	19,563	118,795	**981**
C	**6,236**	**116,434**	**981**
	56,398	169,748	**1,660**
D	**17,484**	**167,658**	**1,660**

Table 1: Experimental results of BnB-ADOPT (first row) compared to BnB-ADOPT$^+$ (second row)

Theorem 2 *If k sends to j two consecutive COSTs with the same content (context, lower/upper bound) and k has not detected a context change, the second message is redundant.*

Proof. Let C_1 and C_2 be two consecutive COSTs sent from k to j with the same content, and $context_k$ has not changed between sending them. Any message may arrive to j between C_1 and C_2. Upon reception, the more recent values of C_1 (and later of C_2) are copied in $context_j$ (by **PriorityMerge** (Yeoh, Felner, and Koenig 2008)). Copying C_2 more recent values in $context_j$ is not essential. Let us assume that these values are not copied. Then, some messages that would have been ignored between C_1 and C_2 will now be accepted. Since there is no context change between C_1 and C_2, these messages will necessarily include contexts compatible with k context, so they will update timestamps only, generating COSTs with the same bounds. At some point, j will receive all the more recent values of C_2 (necessarily before any context change). After this, j will behave as if it would have copied C_2 more recent values. So if those values are not copied, this will not cause any harm. Because of that, our proof concentrates on bounds. When C_1 arrives:

1. C_1 is not compatible with $context_j$, its bounds are discarded. When C_2 arrives: (a) C_2 is not compatible with $context_j$, its bounds are discarded. So, C_2 is redundant. (b) C_2 is compatible with $context_j$, its bounds are included in j. Since C_1 was not compatible, there is at least one agent above j that changed its value, received by j between C_1 and C_2. There are one or several VALUEs on its/their way towards k or k descendants. Upon reception, one or several COSTs will be generated. The last of them will be sent from k to j with more updated bounds. C_2 could have been avoided because a more updated COST will arrive to j. C_2 is redundant.
2. C_1 is compatible with $context_j$, its bounds are included. When C_2 arrives: (a) C_2 is not compatible with $context_j$, its bounds are discarded. So, C_2 is redundant. (b) C_2 is compatible with $context_j$, it bounds are included but this causes no change in j bounds, unless bounds are reinitialized. In this case there is at least one agent above j that changed its value, same as case (1.b). C_2 is redundant. □

BnB-ADOPT$^+$

Temporary, we define BnB-ADOPT$^+$ as BnB-ADOPT with the following changes: (1) the second of two consecutive VALUEs with the same i, j and val is not sent, (2) the second of two consecutive COSTs with the same k, j, $context$, lb and ub when k detects no context change is not sent.

Theorem 3 *BnB-ADOPT$^+$ terminates with the cost of a cost-minimal solution.*

Proof. By Theorems 1 and 2, messages not sent by BnB-ADOPT$^+$ are redundant so they can be eliminated. BnB-ADOPT terminates with the cost of a cost-minimal solution (Yeoh, Felner, and Koenig 2008), so BnB-ADOPT$^+$ also. □

But the new algorithm is not efficient because we have ignored thresholds. Aiming at efficiency, we define BnB-ADOPT$^+$ as BnB-ADOPT with the following changes:

1. i remembers for each neighbor j the last message sent,
2. a COST from j to i includes a boolean $ThReq$, set to true when j threshold is initialized to ∞,
3. if j has to send i a COST equal to (ignoring timestamps) the last COST sent, the new COST is sent iff (if and only if) j has detected a context change between them,
4. if i has to send j a VALUE equal to (ignoring timestamps) the last VALUE sent, the new VALUE is sent iff the last COST that i received from j had $ThReq = true$; upon reception, this VALUE will update j threshold.

We tested our algorithm on binary random DCOPs, meeting scheduling and sensor network. Binary random DCOPs have 10 variables with 10 values and connectivity: 0.4, 0.6, 0.7, 0.8. Costs are selected randomly from the set $\{0,..., 100\}$. Results appear in Table 1 (a), averaged over 50 instances. For meeting scheduling and sensor network formulations (Yin 2008), we tested 4 cases representing different hierarchical and topologies scenarios. Results appear in Table 1 (b) and (c), averaged over 30 instances. Experiments on random instances show that our algorithm reduces the number of messages by a factor from 2 to 3 when connectivity increases. For meeting scheduling, messages are reduced by a factor of at least 2, and for sensor networks, by a factor between 3 and 6. We have achieved important savings for all problems tested. BnB-ADOPT$^+$ was able of processing only half of messages (or less) and reach the optimal solution maintaining the number of cycles practically constant.

References

Modi, P. J.; Shen, W.; Tambe, M.; and Yokoo, M. 2005. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* (161):149–180.

Yeoh, W.; Felner, A.; and Koenig, S. 2008. Bnb-adopt: An asynchronous branch-and-bound DCOP algorithm. *Proc. of AAMAS-08* 591–598.

Yin, Z. 2008. USC dcop repository. Meeting scheduling and sensor net datasets.