Noname manuscript No.
(will be inserted by the editor)

Adding Laziness in BnB-ADOPT$^+$

Jimmy H. M. Lee · Pedro Meseguer · Wen Su

the date of receipt and acceptance should be inserted later

Received: date / Accepted: date

Abstract In distributed constraint optimization, agents executing BnB-ADOPT$^+$ react eagerly to cost changes: they send non-redundant COST messages to their parents as soon as they receive new messages. We have observed that a lazier reaction (not sending COST messages until a condition is met) substantially decrements the number of messages sent and causes only a small variation in ENCCCs. This approach combines nicely with soft arc consistency maintenance during search. We provide experimental evidence of the benefits of this approach on several benchmarks.

1 Introduction and Background

BnB-ADOPT$^+$ [6,3] is a solving algorithm for Distributed Constraint Optimization Problems (DCOPs) that optimizes a global utility function composed of joint utilities of subsets of agents. Each agent executes a copy of the algorithm, which communicates with the other agents using three kinds of messages: VALUE, COST and TERMINATE. In this letter we consider the reduction in communication requirements of BnB-ADOPT$^+$ (reducing the number of messages) made possible by not sending out every COST message. This *lazy* strategy sends out only those COST messages that satisfy some extra conditions (in contrast to the *eager* strategy of the original BnB-ADOPT$^+$ that sends out a COST message as soon as some change has been produced). Interestingly, this lazy approach combines nicely with soft arc consistency maintenance [2], producing an algorithm that uses fewer messages than any of these approaches taken separately. Regarding ENCCCs [1], experimentally we observe a slight variation that does not harm performance.

Jimmy H. M. Lee, Wen Su
Dept Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong
E-mail: {jlee,wsu}@cse.cuhk.edu.hk

Pedro Meseguer
IIIA - CSIC, Campus UAB, 08193 Bellaterra, Spain
E-mail: pedro@iiia.csic.es

DCOP. A DCOP is defined by $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains, where D_i is the domain of variable x_i; $\mathcal{F}$ is a set of binary cost functions, where each cost function $f_{ij} : D_i \times D_j \mapsto \mathbb{N} \cup \{0, \infty\}$ specifies the cost of each combination of values of variables x_i and x_j; $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of agents and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. We assume that each agent controls only one variable. The cost of an assignment of a subset of variables is the evaluation of all cost functions on that assignment. Agents communicate through messages, which are never lost and delivered in the order that they were sent between every pair of agents. In the *constraint graph* of a DCOP instance, nodes correspond to variables and edges connecting pairs of variables correspond to cost functions on these variables. A *depth-first search* (DFS) *pseudo-tree* arrangement of that graph differentiates edges in two classes: *tree edges* such that they form a rooted tree of the constraint graph and *pseudoedges* formed by the remaining edges. This arrangement should satisfy that every pair of constrained variables appears in the same branch of the rooted tree. Edges connect nodes as *parent-child* relation, while pseudoedges connect nodes as *pseudo-parent-pseudochild* relation.

BnB-ADOPT . The BnB-ADOPT algorithm [6], inspired by the ADOPT algorithm [4], computes an optimal solution of a DCOP instance. Both work on a pseudotree of the constraint graph, but they differ in the search strategy: ADOPT uses *best-first search*, while BnB-ADOPT implements a *depth-first branch-and-bound search* strategy. As a consequence, agents change their value assignments differently. The use of thresholds is also different (in ADOPT a threshold is a lower bound but in BnB-ADOPT a threshold is an upper bound). BnB-ADOPT messages are VALUE(i, j, val, th), for a_i to inform child or pseudochild a_j that it has taken value val with threshold th, COST$(k, j, context, lb, ub)$ for a_k to inform parent a_j that with $context$ its bound are lb and ub, and TERMINATE(i, j), for a_i to inform child a_j that a_i terminates. A BnB-ADOPT agent executes the following loop: it reads and processes all incoming messages, and assigns a value. Then, it sends the following messages: a VALUE message per child or pseudochild, and a COST message to its parent. The agent located at the pseudotree root selects values in sequence. It changes value when either (i) the lower and upper bounds of the currently assigned value are equal (meaning that this is the exact minimum cost for that value; in its original form BnB-ADOPT is a minimization algorithm) or (ii) the lower bound of that value is higher than the lowest exact cost already found for a value previously explored. The same strategies are used for agents in non-root nodes, where assignments of agents higher in the pseudotree have to be taken into account when computing costs (they also generate thresholds for children agents). When an agent changes value, this causes to reinitialize all descendent agents that are constrained with it (for details, see [6]). BnB-ADOPT$^+$ [3] is a new version that removes most of the redundant messages. Each agent keeps the last VALUE sent to each child/pseudochild and the last COST message sent to its parent. When this agent has to send a new VALUE/COST message, it sends that message when it is different from the last VALUE/COST message sent for that destination (unless the last COST received from that child contains the boolean field $ThReq =$ true, see [3] for details).

2 Lazy BnB-ADOPT

The basic idea behind the lazy version of BnB-ADOPT$^+$ is that, instead of reacting eagerly to each received message and sending non-redundant VALUE and COST messages, some COST messages can be avoided as long as the one with the highest lower bound contribution is sent, for any agent and context. Intuitively, we want to avoid sending those COST messages that contain a smaller lower bound as they are likely not to contribute enough to effectively prune a value at higher levels of the pseudotree. We implement this idea by adding extra conditions for COST messages to be sent. First of all, we prove that such conditions do not compromise the correctness, termination and optimality of the modified algorithm. In the following, we discuss different conditions for sending COST messages. We prove our results first on the original BnB-ADOPT, and then we prove that they also hold for BnB-ADOPT$^+$. Here we assume some familiarity with BnB-ADOPT$^+$. We adopt the notation of [6] (a is a generic agent, d^a is its current value, X^a is its current context, formed by the current values of constrained agents located higher than a in the pseudotree, TH^a is its threshold, $LB^a(d)$ and $UB^a(d)$ are its lower and upper bounds for value d, and LB^a and UB^a are the minimum lower and upper bounds of its values).

BASIC APPROACH. We consider the limit condition for agent a to send COST messages: $LB^a \geq min\{TH^a, UB^a\}$.[1] We start with the following Lemma, which is needed to prove our main result: a modified BnB-ADOPT that sends COST messages only when $LB^a \geq min\{TH^a, UB^a\}$ remains complete and terminates with the minimum cost.[2]

Lemma 1 *If the context X^a of agent a executing BnB-ADOPT does not change, after trying all its values, agent a will eventually satisfy that for each value d $LB^a(d) \geq min\{TH^a, UB^a\}$.*

Proof. If the context X^a does not change, agent a tries all its values. According to the BnB-ADOPT pseudocode [6], for each value d, agent a performs search until $LB^a(d) \geq min\{TH^a, UB^a\}$. This is a consequence of how BnB-ADOPT works (see the pseudocode in [6], specially from line 23 in the Backtrack procedure): a value d^a is pursued by agent a until its lower bound reaches $min\{TH^a, UB^a\}$; then a changes value, selecting a new current value d^a. □

Proposition 1 *BnB-ADOPT remains complete and terminates with the minimum solution cost when only COST messages with $LB^a \geq min\{TH^a, UB^a\}$ are sent.*

Proof. By Lemma 1, if the context X^a does not change, an agent a executing BnB-ADOPT, after trying all its values, satisfies that for each value d $LB^a(d) \geq min\{TH^a, UB^a\}$. The last COST message that a potentially sends for context X^a is when $LB^a \geq UB^a$

[1] It is a kind of *limit condition* because failing to send a COST message satisfying this condition may result in incompleteness of the modified algorithm (imagine that the current value of every agent is one that minimizes total cost; if an agent a does not send a COST to its parent when $LB^a \geq UB^a$, the parent will wait infinitely and the algorithm would not terminate).

[2] Here, the COST message contains LB^a and UB^a, which are copied from the sender agent a.

or when $LB^a \geq TH^a$. If agent a sends this COST message, everything that can be achieved with previous messages can be achieved with this one, since the message contains the highest/lowest possible contribution to the lower/upper bound of the cost that any value of this agent a may cause in context X^a. Therefore, replacing previous COST messages with inferior lower bounds by this one satisfying $LB^a \geq min\{TH^a, UB^a\}$ does not affect the termination of BnB-ADOPT with the minimum solution cost. If context X^a changes before agent a finishes trying all its values, it happens that the condition $LB^a \geq min\{TH^a, UB^a\}$ is not satisfied, and no COST message is sent. A context change means that an ancestor of a has realized that its current value contained in the context X^a is too costly. Thus another value must be tried, which causes a context update. Agent a is reinitialized, computing the costs of each value according to the new context. Observe that a not sending any COST message for context X^a causes no problem, because that ancestor agent is no longer interested in the cost of a for X^a (any such message would be discarded anyway when it will arrive to such ancestor, because it contains an outdated context). Therefore, no matter whether there is a context change or not, sending COST messages when the condition $LB^a \geq min\{TH^a, UB^a\}$ is satisfied does not affect the termination of BnB-ADOPT with the minimum solution cost. □

Proposition 2 *BnB-ADOPT$^+$ remains complete and terminates with the minimum solution cost when only COST messages with $LB^a \geq min\{TH^a, UB^a\} \vee ThReq = true$ are sent.*

Proof. BnB-ADOPT$^+$ [3] is simply BnB-ADOPT where some messages proved redundant have been removed, keeping the completeness and termination with minimum cost of the original algorithm. Therefore, the arguments used to justify the extra condition $LB^a \geq min\{TH^a, UB^a\}$ also apply to BnB-ADOPT$^+$. Regarding the second condition $ThReq = true$, this is included for efficiency purposes (in BnB-ADOPT$^+$, COST messages have $ThReq$ as an extra field, which indicates when the parent has to send the threshold to a child in the next VALUE message; this condition is included to propagate the right thresholds after a context change). □

The lazy basic version, that we call BnB-ADOPT$^+_{lb}$, sends only COST messages satisfying the condition $LB^a \geq min\{TH^a, UB^a\} \vee ThReq = true$. This does not compromise the completeness and termination of the new algorithm. However, not sending out every possible COST message implies that some opportunities for value pruning at higher levels of the pseudotree might be lost. Pruning a value is equivalent to give up traversing the search tree rooted at that value, which may result in saving some VALUE and COST messages needed to traverse that search tree at lower levels, as well as saving search effort recorded in ENCCCs. This motivates the next part.

STOCHASTIC APPROACH. It is easy to see that any disjunction involving the condition(s) we have devised for sending COST messages in Propositions 1 and 2 maintains the completeness and termination with minimum cost of the modified algorithm. This allows us a wide spectrum of possibilities for the degree of laziness, since any disjunction including $LB^a \geq min\{TH^a, UB^a\}$ (in BnB-ADOPT$^+$ in disjunction with $ThReq = true$) can be used as prerequisite to send COST messages. Aiming to

find an extra condition that is efficient in practice, we have considered the following desirable criteria:

- As the number of nodes in a search tree grows exponentially with depth, the condition for deep nodes should be stronger than for shallow nodes, in order to keep under control the amount of COST messages from deep nodes.
- Pruning a value implies savings in messages and ENCCCs. The higher an agent is in the pseudotree, the more savings may cause value pruning. To enforce pruning, agents at higher levels should have a weaker condition than other agents lower in the pseudotree.
- Agents at leaf nodes in the pseudotree must always send their COST messages. This is always satisfied since we always have $LB^a = UB^a$ at a leaf agent a.

We have produced a lazy random version, called BnB-ADOPT$^+_{lr}$, that, in disjunction with the conditions mentioned in Propositions 1 and 2, includes a new condition of stochastic nature, satisfying that the deeper the agent is in the pseudotree, the smaller the probability of satisfaction of that new condition. This new condition [3] is $random(tree\text{-}height) + 1 \geq agent\text{-}depth$, where $tree\text{-}height$ is the height of the pseudotree, $agent\text{-}depth$ is the depth of the agent in the pseudotree and $random(x)$ generates a random integer in $[0, x-1]$. Experimentally, this stochastic condition has given good results.

COMBINING WITH SOFT ARC CONSISTENCY MAINTENANCE. The connection of this approach with maintaining soft arc consistency (AC) during search [2] does not present any theoretical difficulty. On the one hand, the lazy approach keeps the completeness and termination of the BnB-ADOPT$^+$ algorithm, so that soft AC maintenance can work on top of the lazy versions without any trouble. On the other hand, since distributed search is done on original cost functions (soft AC is enforced on copies of original cost functions), the action of soft AC maintenance is limited to value deletions in domains, without interference with distributed search.

Although the extra condition on sending COST messages may not have a theoretical impact, it may offset the good performance obtained by the addition of soft AC maintenance. The point here is that COST messages contain elements needed to enforce soft AC. Limiting the sending of COST messages may cause to work with old values of these elements, but not the most updated version. However, experimental results indicate that in practice this is not a big issue. In particular, the lazy random version achieves a good performance, combining message savings obtained from either soft AC maintenance or COST messages limitations. We provide specific results on this in Section 3.

[3] Testing other conditions of the same kind, we observe that there is a trade-off here: the easier it is to send a COST message, the more pruning is obtained and less ENCCCs are incremented (on average over 30 or 50 instances depending on the benchmark). In the limit, sending every possible COST message takes advantage of every opportunity for pruning, at the cost of sending many COST messages (although not all of them would effectively contribute for pruning a value).

3 Experimental Results

We assess the efficiency gain of adding laziness into BnB-ADOPT$^+$ by comparing the performance of the original algorithm with its lazy versions, also including the combination with soft AC maintenance (MAC), on the following benchmarks:

- Binary random DCOP. Instances are characterized by $\langle n, d, p\rangle$, where n is the number of variables, d is the domain size and p is the network connectivity. We have generated four classes of random DCOP instances: $\langle n = 10, d = 10, p = 0.2, 0.3, 0.4, 0.5\rangle$, 50 instances for each class. Costs are selected from a uniform cost distribution. Two types of binary cost functions are used, small and large. Small cost functions extract costs from the set $\{0, \ldots, 10\}$ while large ones extract costs from the set $\{0, \ldots, 1000\}$. The proportion among them is 3/4 small and 1/4 large (this is done to introduce some variability among tuple costs).
- Soft graph coloring. Same as binary random DCOP, instances are also characterized by $\langle n, d, p\rangle$ with the same parameter meaning. This is the softened version of the graph coloring problem, where the assignment of values v_i and v_j to agents a_i and a_j has a cost equal to $d^2 - |v_i - v_j|^2$. We generated several classes with the following parameters: $\langle n = 6, 7, 8, 9, d = 8, p = 0.4\rangle$, 50 instances for each parameter setting.
- Meeting scheduling. Obtained from the DCOP repository [7], this benchmark presents 4 classes, all with domain 9: case A (8 variables), case B (10 variables), case C (12 variables) and case D (12 variables). Each class contains 30 instances.
- Sensor networks. Also from the DCOP repository [7], this benchmark considers 4 classes, all with domain 9: case A (16 variables), case B (16 variables), case C (10 variables) and case D (16 variables). Each class contains 30 instances.

Execution was in a discrete event simulator, evaluating performance in terms of communication cost (total number of messages exchanged) and computation effort (equivalent non-concurrent constraint checks, ENCCC [1]). Inspired by [5], upon receiving a message, the receiving agent updates its ENCCC counter using the following rule $ENCCC_{agent} = max\{ENCCC_{agent}\ , ENCCC_{message} + 1000\}$.

Results appear in Table 1, where we can see the benefits of our approach. At first sight, we see a clear reduction in the number of messages sent by the implementations including the lazy idea (the larger and more difficult instances, the higher the reduction in number of messages sent). We also observe a trade-off between message saving and ENCCC increment.

Specifically, considering the lazy basic approach, we observe a clear trend in all classes of all benchmarks tested: BnB-ADOPT$^+_{lb}$, requires substantially less messages than the original BnB-ADOPT$^+$, at the extra cost of increasing its ENCCCs by a small amount. The lazy approach decrements the number of COST messages used, but increases the number of VALUE messages. The final balance is a total message reduction. These data allow a clear interpretation. Since we are limiting at maximum the COST messages that can be sent, this has a clear impact in the number of values pruned. Less pruning means more search effort, so more messages are required (which justifies the increment in VALUE messages) and more ENCCCs are done.

The lazy random algorithm, BnB-ADOPT$^+_{lr}$, allows for some more pruning opportunities, because it is sending COST messages under weaker conditions than the lazy basic one. This explains why the number of VALUE messages decreases, while the number of COST messages increases. The total balance is slightly higher than the lazy basic algorithm. More pruning opportunities means less search effort. And this is reflected in the decrement of ENCCCs with respect to BnB-ADOPT$^+_{lb}$. The lazy random algorithm appears as an intermediate point between the original one and the lazy basic one: it requires more messages but performs less ENCCCs than BnB-ADOPT$^+_{lb}$. If compared with original BnB-ADOPT$^+$, the reduction in number of messages is still substantial, while the increment in ENCCCs is quite small. Interestingly, when adding soft AC maintenance (MAC), results follow the same trend.

4 Conclusions

From this work we conclude that a lazy approach is a simple but powerful method to decrement the communication requirements inside the BnB-ADOPT$^+$ solving algorithm, at the extra cost of a small increment in ENCCCs on average. We expressed this idea by adding some particular conditions in the BnB-ADOPT$^+$ code that should be satisfied prior to sending a COST message. Regarding the algorithmic extensions presented, the lazy random one shows a more balanced behavior than the lazy basic one in the four benchmarks tested. This approach is not only supported by experimental results, but we have proved that the above mentioned conditions maintain the completeness and termination with minimum cost of the modified algorithms.

Acknowledgements

This research has been supported by the following grants: CUHK413808, CUHK413710 and CUHK413713 from the Research Grants Council of Hong Kong SAR, the CSIC/RGC Joint Research Scheme grants S-HK003/12 and 2011HK0017, TIN2013-45732-C4-4-P from Spanish MINECO, and Generalitat de Catalunya SGR-2014-118.

References

1. A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proc. AAMAS 2006*, pages 1427–1429, 2006.
2. P. Gutierrez, J. Lee, K. Lei, T. Mak, and P. Meseguer. Maintaining soft arc consistency in BnB-ADOPT+ during search. In *Proc. CP 2013*, LNCS 8124, pages 365–380, 2013.
3. P. Gutierrez and P. Meseguer. Removing redundant messages in N-ary BnB-ADOPT. *Journal of Artificial Intelligence Research*, pages 287–304, 2012.
4. P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.
5. O. Peri and A. Meisels. Synchronizing for performance - DCOP algorithms. In *Proc. ICAART 2013*, pages 5–14, 2013.
6. W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010.
7. Z. Yin. USC DCOP repository. Meeting scheduling and sensor net datasets, http://teamcore.usc.edu/dcop, 2008.

(a) Binary Random $n = 10, d = 10$

		no MAC				MAC			
p	Algorithm	#msg	#V	#C	ENCCC	#msg	#V	#C	ENCCC
0.2	BnB-ADOPT$^+$	587 (100%)	207	370	159417 (100%)	365 (100%)	95	182	84050 (100%)
	BnB-ADOPT$^+_{lb}$	476 (-19%)	209	258	160948 (+1%)	309 (-15%)	105	122	90649 (+8%)
	BnB-ADOPT$^+_{lr}$	525 (-11%)	208	308	159899 (+0%)	328 (-10%)	97	143	85573 (+2%)
0.3	BnB-ADOPT$^+$	12587 (100%)	3589	8988	2820305 (100%)	5496 (100%)	1345	3597	1200343 (100%)
	BnB-ADOPT$^+_{lb}$	7970 (-37%)	3667	4294	2874389 (+2%)	4104 (-25%)	1558	1687	1395567 (+16%)
	BnB-ADOPT$^+_{lr}$	9305 (-26%)	3598	5698	2826676 (+0%)	3944 (-28%)	1366	2013	1218497 (+2%)
0.4	BnB-ADOPT$^+$	204721 (100%)	45289	159423	36938269 (100%)	46868 (100%)	8367	30700	7707417 (100%)
	BnB-ADOPT$^+_{lb}$	105844 (-48%)	46842	58992	38381195 (+4%)	41526 (-11%)	12353	14679	11557117 (+50%)
	BnB-ADOPT$^+_{lr}$	125419 (-39%)	45473	79937	37108586 (+0%)	32330 (-31%)	8863	14712	8205229 (+6%)
0.5	BnB-ADOPT$^+$	1252953 (100%)	265956	986978	226314029 (100%)	162098 (100%)	29172	114581	26672325 (100%)
	BnB-ADOPT$^+_{lb}$	611318 (-51%)	280120	331190	238771208 (+6%)	110036 (-32%)	40862	45666	38394064 (+44%)
	BnB-ADOPT$^+_{lr}$	676398 (-46%)	268323	408066	228806919 (+1%)	91572 (-44%)	29602	43736	27366962 (+3%)

(b) Soft Graph Coloring

		no MAC				MAC			
n, m, p	Algorithm	#msg	#V	#C	ENCCC	#msg	#V	#C	ENCCC
6, 8, 0.4	BnB-ADOPT$^+$	520 (100%)	173	342	178563 (100%)	305 (100%)	78	150	77168 (100%)
	BnB-ADOPT$^+_{lb}$	376 (-28%)	174	197	179688 (+1%)	248 (-18%)	83	87	80830 (+5%)
	BnB-ADOPT$^+_{lr}$	434 (-17%)	173	256	178183 (- 0%)	272 (-11%)	81	111	78753 (+2%)
7, 8, 0.4	BnB-ADOPT$^+$	3796 (100%)	1065	2725	1131350 (100%)	1913 (100%)	438	1111	476447 (100%)
	BnB-ADOPT$^+_{lb}$	2347 (-38%)	1091	1251	1170465 (+3%)	1344 (-30%)	468	489	514292 (+8%)
	BnB-ADOPT$^+_{lr}$	2783 (-27%)	1076	1701	1145571 (+1%)	1492 (-22%)	449	659	492034 (+3%)
8, 8, 0.4	BnB-ADOPT$^+$	29483 (100%)	6579	22897	7559987 (100%)	17220 (100%)	3354	11598	3921667 (100%)
	BnB-ADOPT$^+_{lb}$	15058 (-49%)	6850	8200	7973947 (+5%)	9868 (-43%)	3539	3900	4226683 (+8%)
	BnB-ADOPT$^+_{lr}$	18190 (-38%)	6620	11563	7636022 (+1%)	11269 (-35%)	3420	5488	4029280 (+3%)
9, 8, 0.4	BnB-ADOPT$^+$	200250 (100%)	42491	157301	42120374 (100%)	80792 (100%)	15595	55800	15182978 (100%)
	BnB-ADOPT$^+_{lb}$	95624 (-52%)	44881	50735	44342293 (+5%)	44323 (-45%)	16561	17669	16535342 (+9%)
	BnB-ADOPT$^+_{lr}$	106844 (-47%)	43366	63470	42582980 (+1%)	47654 (-41%)	15843	21983	15549240 (+2%)

(c) Meeting Scheduling

		no MAC				MAC			
Class	Algorithm	#msg	#V	#C	ENCCC	#msg	#V	#C	ENCCC
A	BnB-ADOPT$^+$	8040 (100%)	1993	6040	2243596 (100%)	5589 (100%)	1190	3624	1430625 (100%)
	BnB-ADOPT$^+_{lb}$	4619 (-43%)	2139	2474	2462052 (+10%)	3994 (-29%)	1533	1515	1941389 (+36%)
	BnB-ADOPT$^+_{lr}$	5448 (-32%)	2013	3427	2283774 (+2%)	3879 (-31%)	1254	1827	1527059 (+7%)
B	BnB-ADOPT$^+$	5205 (100%)	1305	3891	936467 (100%)	6093 (100%)	936	2972	845143 (100%)
	BnB-ADOPT$^+_{lb}$	3258 (-37%)	1360	1890	990148 (+6%)	5119 (-16%)	1123	1355	1082620 (+28%)
	BnB-ADOPT$^+_{lr}$	3876 (-26%)	1313	2553	947579 (+1%)	4857 (-20%)	976	1670	905592 (+7%)
C	BnB-ADOPT$^+$	2544 (100%)	859	1674	518977 (100%)	2315 (100%)	592	1196	397394 (100%)
	BnB-ADOPT$^+_{lb}$	1941 (-24%)	881	1049	536246 (+3%)	2077 (-10%)	698	745	489370 (+23%)
	BnB-ADOPT$^+_{lr}$	2188 (-14%)	861	1315	522289 (+1%)	2045 (-12%)	618	883	424978 (+7%)
D	BnB-ADOPT$^+$	2733 (100%)	883	1839	553076 (100%)	3216 (100%)	574	1222	441624 (100%)
	BnB-ADOPT$^+_{lb}$	2073 (-24%)	920	1142	583868 (+6%)	3338 (+4%)	704	819	554044 (+25%)
	BnB-ADOPT$^+_{lr}$	2382 (-13%)	884	1487	553878 (+0%)	2971 (-8%)	587	941	449040 (+2%)

(d) Sensor Networks

		no MAC				MAC			
Class	Algorithm	#msg	#V	#C	ENCCC	#msg	#V	#C	ENCCC
A	BnB-ADOPT$^+$	615 (100%)	174	426	155183 (100%)	970 (100%)	147	389	154379 (100%)
	BnB-ADOPT$^+_{lb}$	417 (-32%)	175	228	155718 (+0%)	794 (-18%)	154	193	158737 (+3%)
	BnB-ADOPT$^+_{lr}$	496 (-19%)	174	306	155183 (+0%)	847 (-13%)	148	265	155898 (+1%)
B	BnB-ADOPT$^+$	809 (100%)	218	575	176339 (100%)	1266 (100%)	193	541	179225 (100%)
	BnB-ADOPT$^+_{lb}$	513 (-37%)	219	279	177210 (+0%)	1016 (-20%)	207	250	189543 (+6%)
	BnB-ADOPT$^+_{lr}$	585 (-28%)	218	352	176406 (+0%)	1042 (-18%)	195	312	181111 (+1%)
C	BnB-ADOPT$^+$	577 (100%)	202	366	166895 (100%)	538 (100%)	125	236	113288 (100%)
	BnB-ADOPT$^+_{lb}$	462 (-20%)	207	246	170781 (+2%)	475 (-12%)	137	152	122745 (+8%)
	BnB-ADOPT$^+_{lr}$	511 (-11%)	202	300	166962 (+0%)	488 (-9%)	128	185	115268 (+2%)
D	BnB-ADOPT$^+$	1319 (100%)	452	852	286133 (100%)	1652 (100%)	348	676	235921 (100%)
	BnB-ADOPT$^+_{lb}$	1025 (-22%)	458	552	289385 (+1%)	1483 (-10%)	376	430	261095 (+11%)
	BnB-ADOPT$^+_{lr}$	1182 (-10%)	451	715	284828 (-1%)	1531 (-7%)	350	548	238227 (+1%)

Table 1 Experimental results of (a) Binary Random DCOP, (b) Soft Graph Coloring, (c) Meeting Scheduling and (d) and Sensor Networks benchmarks, indicating the percentage of variation (increment or decrement of #msg and ENCCCs) of the lazy versions with respect to the original algorithm (whose values are taken as the reference, 100%). Averages and percentages are rounded to integers (+0% and -0% indicate very small positive and negative variations). #V and #C count for number of VALUE and COST messages respectively. In the MAC part, #msg is clearly higher than the sum of VALUE and COST messages because in this case more message types are used (not reported in the tables for space reasons). Observe that for some instances of (c) and (d), the number of messages required with MAC is higher than that without MAC: in easy instances, maintaining soft AC does not pay off (savings do not compensate the overhead).