Improvement of Arc Consistency in Asynchronous Forward Bounding algorithm

Rachid Adrdor[1], Lahcen Koutti[1]
rachid.adrdor@edu.uiz.ac.ma l.koutti@uiz.ac.ma
[1] Ibn Zohr University, Faculty of Sciences, Department of Computer Science, Agadir, Morocco

AJCAI 2021 : The 34th Australasian Joint Conference on Artificial Intelligence, 2-4 February 2022, Sydney

Abstract

The AFB_BJ$^+$-AC* algorithm is one of the latest algorithms used to solve Distributed Constraint Optimization Problems (DCOPs). It is based on simple arc consistency (AC*) to speed up the process of solving a problem by permanently removing any value that doesn't belong to its optimal solution. In this paper, we use a directional arc consistency (DAC*), the next higher level of AC*, to erase more values and thus to quickly reach the optimal solution of a problem. Experiments on some benchmarks show that the new algorithm, AFB_BJ$^+$-DAC*, is better in terms of communication load and computation effort.

I Introduction

In a DCOP, variables, domains, and constraints are distributed among a set of agents. Each agent has full control over a subset of variables and constraints. A DCOP is solved in a distributed manner via an algorithm allowing the agents to cooperate with each other to find a solution with a minimal cost. A solution to a DCOP is a set of value assignments, each representing the value assigned to one of the variables in that DCOP. AFB_BJ$^+$-AC*[1] is one of the recent algorithms which uses soft arc consistency (AC*) to solve DCOPs. In this work, instead of using AC* with AFB_BJ$^+$, we use Directional AC* (DAC*). This helps to largely narrow down agents' domains of a given DCOP and thus quickly reach its optimal solution. This change produces a new algorithm called AFB_BJ$^+$-DAC*. Our experiments on different benchmarks show the superiority of AFB_BJ$^+$-DAC* algorithm in terms of communication load and computation effort.

II Background

1 DCOP

A DCOP [2] is defined by 4 sets:

- $\mathcal{A} = \{A_1, A_2, ..., A_k\}$: Agents;
- $\mathcal{X} = \{x_1, x_2, ..., x_n\}$: Variables;
- $\mathcal{D} = \{D_1, D_2, ..., D_n\}$: Domains;
  - ⋆ D_i : the possible values of x_i
- $\mathcal{C} = \{c_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+\} \cup \{c_i : D_i \rightarrow \mathbb{R}^+\}$: Constraints

Objective:

find a solution S with $\sum c_i + c_{ij}, C_i, C_{ij} \in S$ is minimized

For simplicity purposes, we consider a restricted version of DCOP where two variables, at most, are linked by one constraint (i.e., unary or binary constraint) and each agent is responsible for a single variable ($k = n$).

2 Directional Arc Consistency (DAC*)

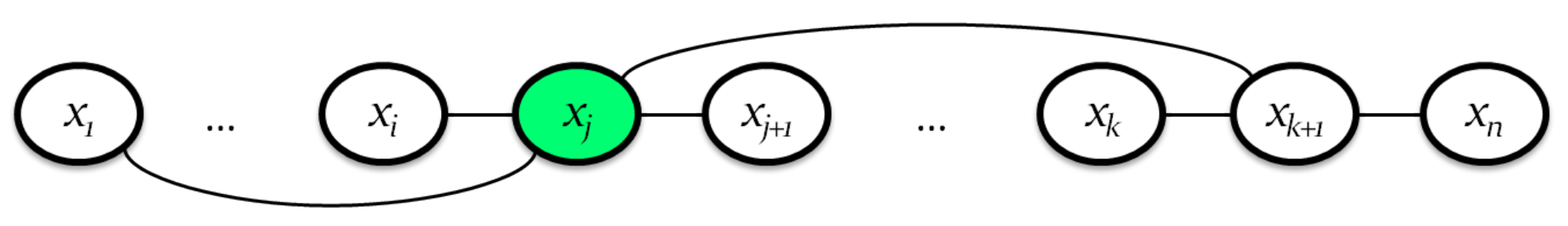
DAC* is a set of rules that are applied to a problem to remove values that are not part of its optimal solution. A problem is DAC* if each variable x_i of this problem is DAC* with its neighbors x_j, such that $j > i$. A variable x_i is DAC* with respect to its neighbor x_j, such that $j > i$, if each value $v_i \in D_i$ satisfies $C_\phi + c_i(v_i) < UB_i$, and there is a value $v_j \in D_j$ which satisfies $c_{ij}(v_i, v_j) + c_j(v_j) = 0$. v_j is called a *full support* of v_i.

- ⋆ $c_{ij}(v_i, v_j)$ is the binary cost of (v_i, v_j).
- ⋆ $c_j(v_j)$ is the unary cost of v_j.
- ⋆ C_ϕ is the global lower bound.
- ⋆ UB_i is the global upper bound.

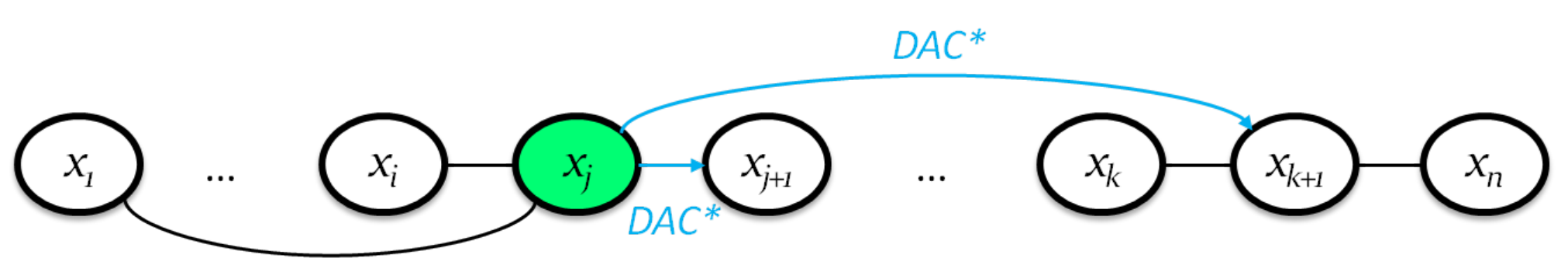
AFB_BJ$^+$-DAC* algorithm

AFB_BJ$^+$-DAC* algorithm works according to five main steps:

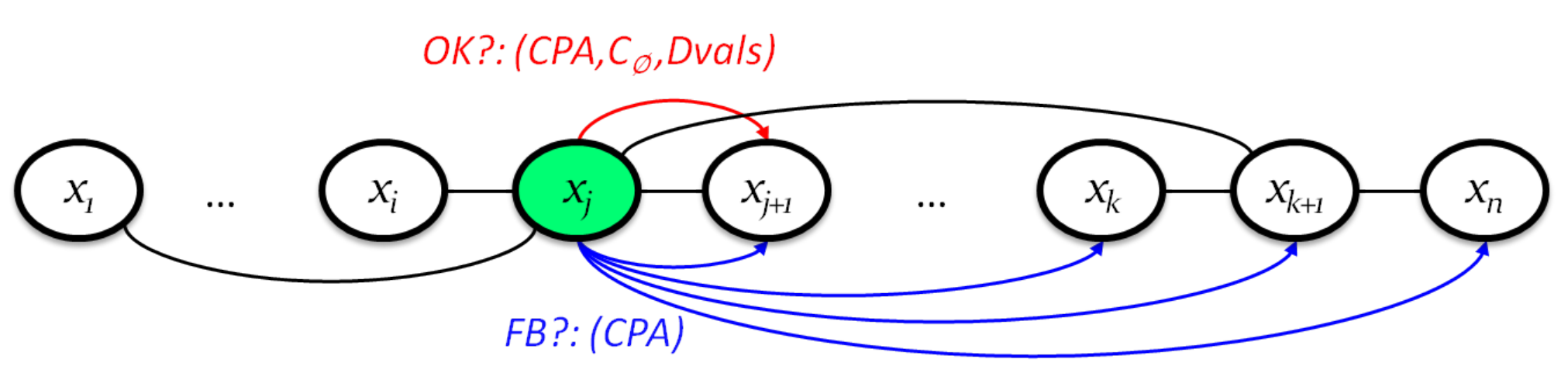
1. Initialization : a static order is applied to agents of the problem. Each agent initializes its data structures and the first agent starts enforcing DAC*.



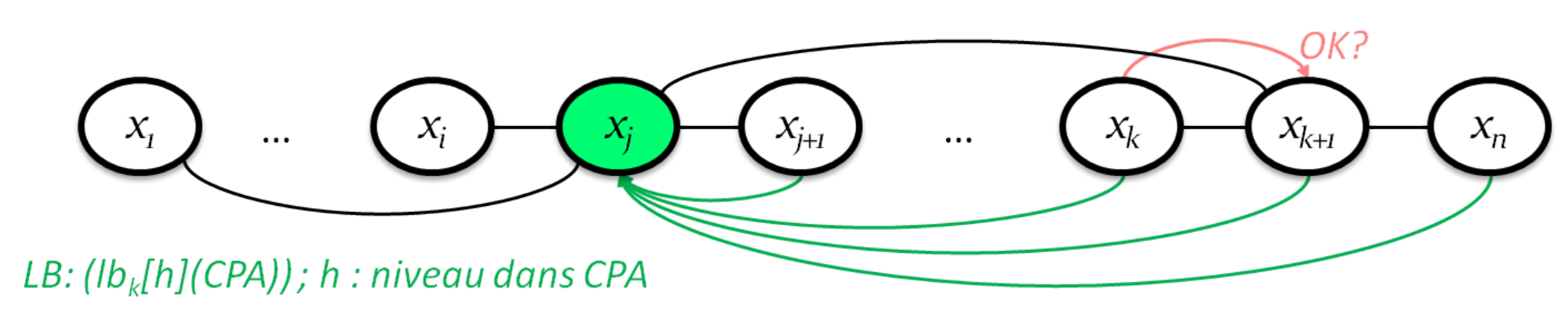
2. Enforcing DAC* : the current agent x_j updates its binary constraints shared with its higher neighbors using the received extension values. Then, it performs, for each higher neighbor x_i, two projections: the first one to update its unary costs, and the second one to update the value of C_ϕ. After all, it filters its domain D_j by removing any value v_j that satisfies $c_j(v_j) + C_\phi \geq UB_j$. Finally, it performs a cost extension to its lower neighbors x_k by shifting its unary costs to binary costs. Then, it performs a binary projection on its lower neighbors to keep the symmetry of the binary constraints shared between them.



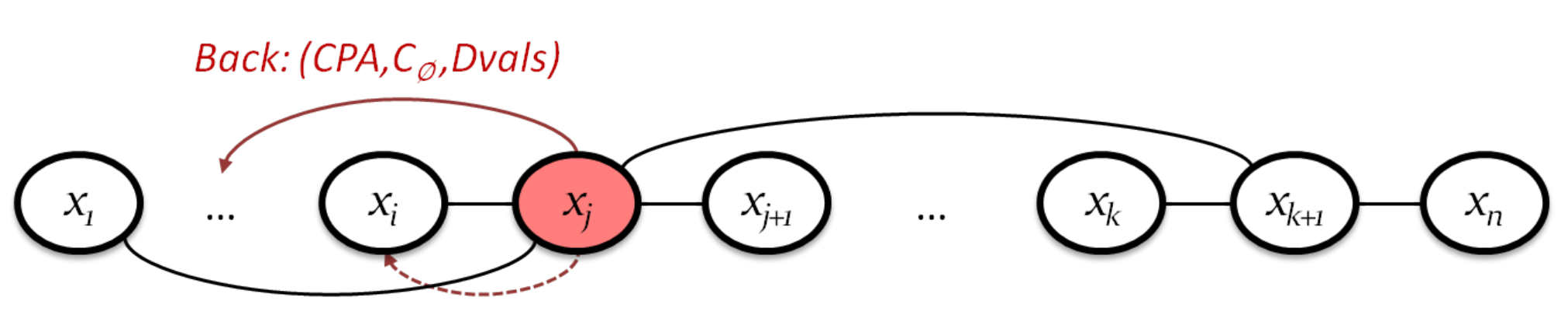
3. Assigning variables : the current agent x_j chooses, for x_j, a value from its previously filtered domain D_j to extend the CPA Y^j by its value assignment (x_j, v_j). If x_j has successfully extended the CPA, it sends an **ok?** message to the next agent asking it to continue the extension of CPA Y^j. This message loads the extended CPA Y^j, its guaranteed costs, the C_ϕ, the list of extension values, and the list of deleted values. At the same time, it sends **fb?** messages to unassigned agents asking them to evaluate the included CPA and send their estimates on it. $Y = Y^j = [(x_1, v_1), \ldots, (x_j, v_j)]$ is a current partial assignment (CPA).



4. Evaluating the CPA : When receiving an **fb?** message, each receiving agent computes the lower bounds corresponding to the received CPA Y^j, then it sends them to the requesting agent x_j via an **lb** message. The lower bounds represent the cost estimates, on the CPA Y^j, of each agent not yet assigned with respect to its lower neighbors. When receiving an **lb** message, x_j computes the global lower bound for the evaluated CPA Y^j and checks if it exceeds UB_j.



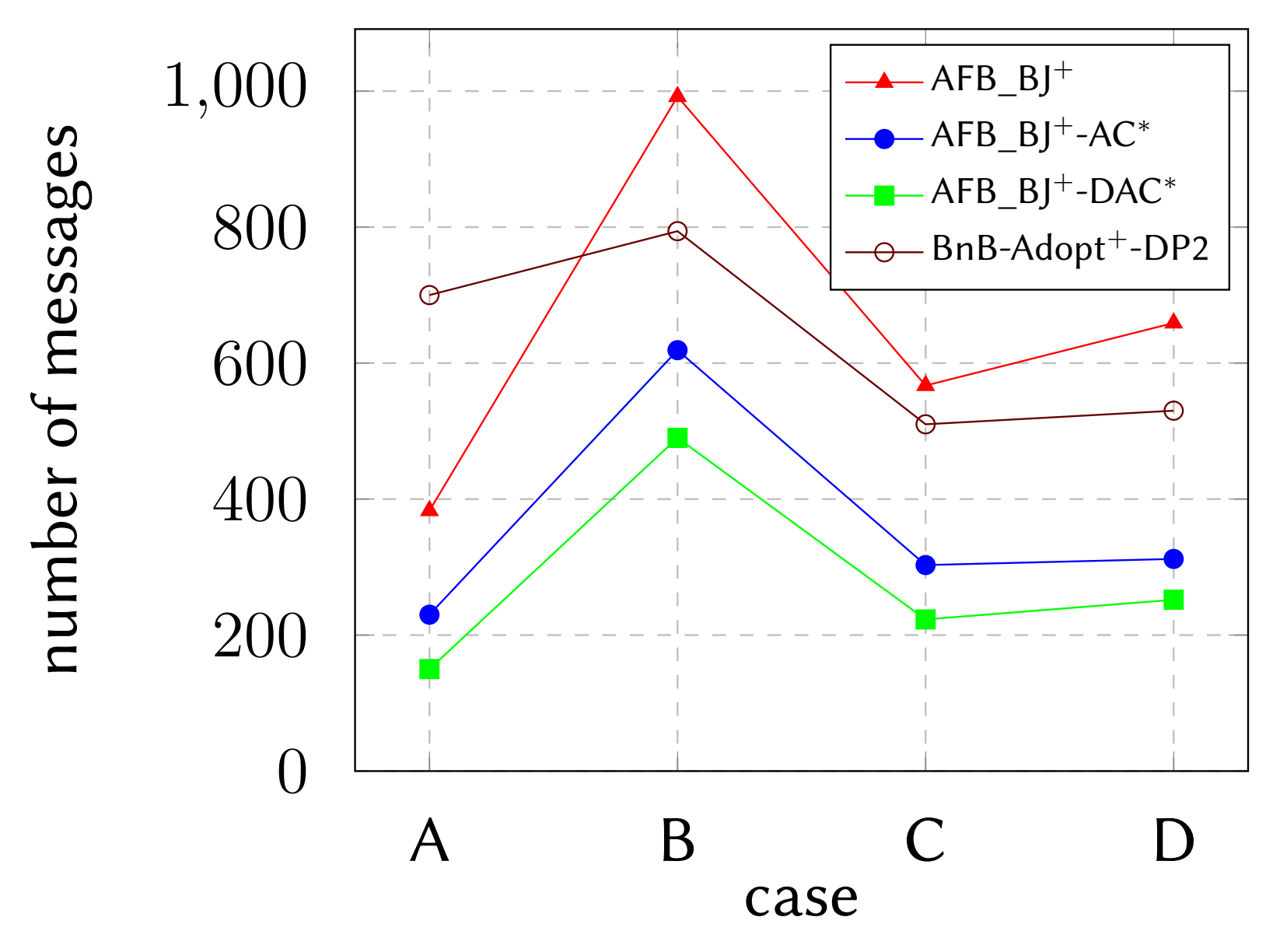
5. Backjumping : If the global lower bound of CPA exceeds UB_j, x_j changes the value assigned to its variable by a more appropriate one if it exists. If it does not exist, it backjumps to the previous agents exactly the guilty agent by sending it a **back** message. If the guilty agent does not exist or the domain of x_j becomes empty, x_j stops its execution and informs the others via **stp** messages.



AFB_BJ$^+$-DAC* continues in this manner by repeating these steps until a solution with minimal cost is found.

Experimental Results

We experimentally compare AFB_BJ$^+$-DAC* with its older versions [2, 1] and with the BnB-Adopt$^+$-DP2 algorithm [2], which is its famous competitor.
To compare the algorithms, we use two metrics, the total of messages exchanged ($msgs$) for the communication load and the total of non-concurrent constraint checks ($ncccs$) for the computation effort.
An example of benchmarks used in these experiments is meetings scheduling. These are problems in which a number of participants seek to meet, either in pairs or in groups, at a given place and date. The objective is therefore to know how to plan these meetings so that all the participants are satisfied. We have evaluated 4 cases A, B, C, and D, which are different in terms of meetings/participants.



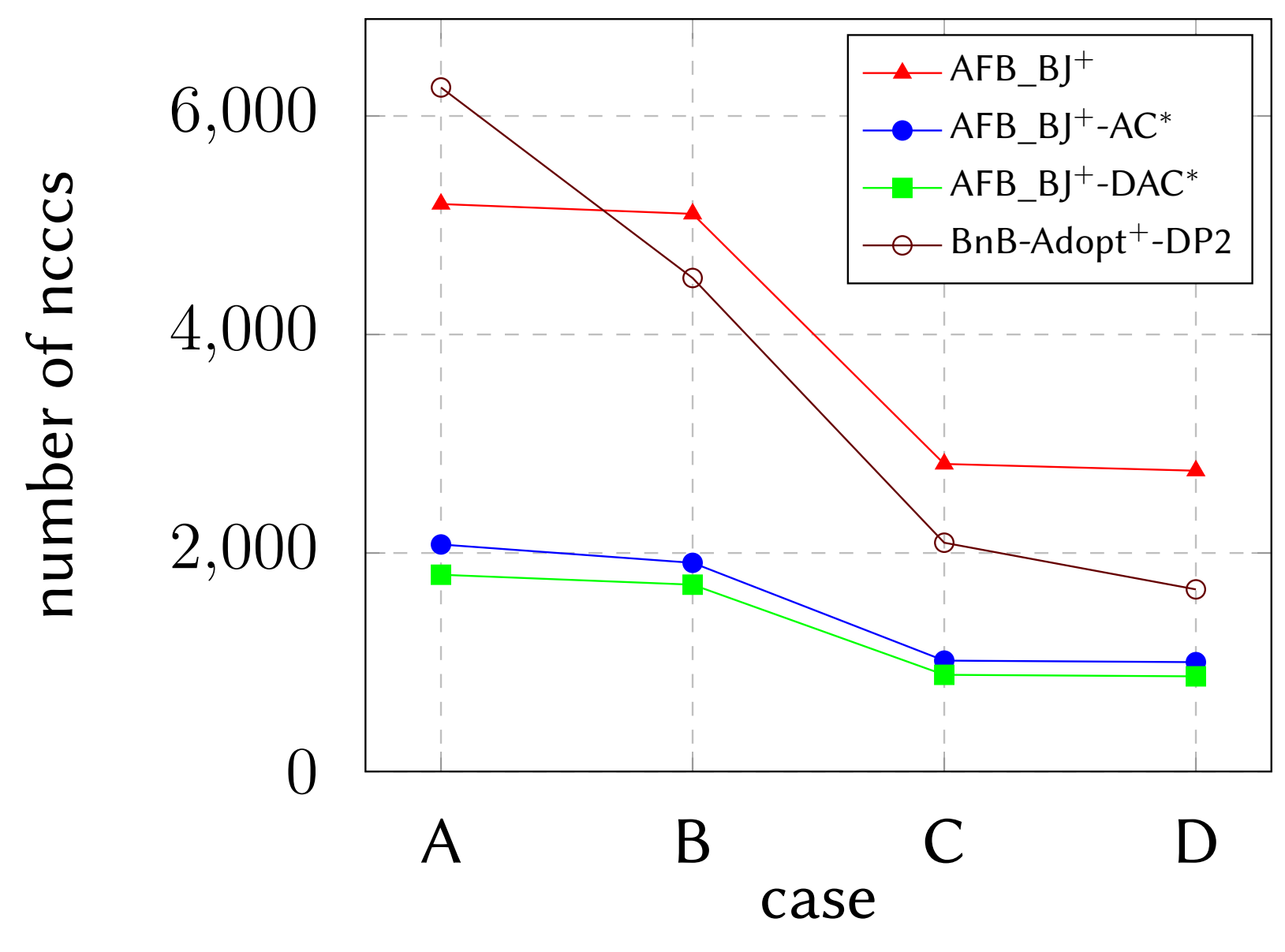


Figure 1: Total of $msgs$ sent and $ncccs$ for meetings scheduling

The results obtained (Fig. 1) show a clear improvement of the AFB_BJ$^+$-DAC* compared to others, whether for $msgs$ or for $ncccs$.
By analyzing these results, we can conclude that the AFB_BJ$^+$-DAC* is better than its earlier versions because of the existence of DAC* which allows agents to remove more suboptimal values.

Conclusion

In this paper, we have introduced the AFB_BJ$^+$-DAC* algorithm. It relies on DAC* to generate more deletions and thus quickly reach the optimal solution of a problem. DAC* mainly relies on performing a set of cost extensions in one direction from an agent to its lower priority neighbors in order to perform AC* multiple times. Experiments on some benchmarks show that the AFB_BJ$^+$-DAC* behaves better than its older versions. As future work, we propose to exploit the change in the size of the agent domains in variable ordering heuristics.

References

[1] Rachid Adrdor and Lahcen Koutti. Enhancing AFB_BJ$^+$AC* algorithm. In *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, pages 1–7. IEEE, July 2019.
[2] Mohamed Wahbi, Redouane Ezzahir, and Christian Bessiere. Asynchronous forward bounding revisited. In *International Conference on Principles and Practice of Constraint Programming*, pages 708–723. Springer, 2013.