

domain. Each constraint involves two agents and specifies its non-negative constraint cost as a function of the values of these two agents. A solution is an agent-value assignment for all agents, while a partial solution is an agent-value assignment for a subset of agents. The cost of a solution is the sum of the constraint costs of all constraints. Solving the DCOP problem optimally means to determine the cost of a cost-minimal solution. Each agent needs to decide which value to take on based on its knowledge of the constraints that it is involved in and messages that it can exchange with the other agents. These messages can be delayed by a finite amount of time but are never lost.

3. BnB-ADOPT

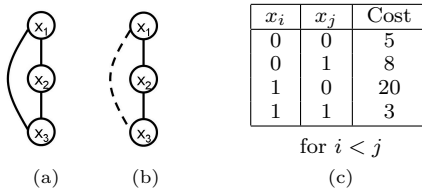


Figure 1: Example DCOP Problem

DCOP problems can be represented with constraint graphs, whose vertices are the variables and whose edges are the constraints. Figure 1(a) shows the constraint graph of an example DCOP problem with three agents that can each take on the values zero or one. There is a constraint between each pair of agents. Figure 1(c) shows the constraint costs of the example DCOP problem, which are the same for all three constraints. Constraint trees are spanning trees of constraint graphs with the property that edges of the constraint graphs can connect vertices only with their ancestors or descendants in the constraint trees. Sibling subtrees represent disjoint subproblems of the DCOP problem. Figure 1(b) shows one possible constraint tree of the example DCOP problem, where the dotted line is part of the constraint graph but not the constraint tree. This constraint tree is actually a constraint chain and thus there are no disjoint subproblems. The operation of DCOP algorithms on constraint trees can be visualized with search trees. Figure 2 shows a search tree for this constraint tree, where levels 1, 2 and 3 of the search tree correspond to agents x_1 , x_2 and x_3 , respectively. Left branches correspond to the agents taking on the value zero and right branches to the agents taking on the value one. Each non-leaf node thus corresponds to a partial solution of the DCOP problem and each leaf node to a solution. Figure 2(a) shows the identifiers of the nodes that allow us to refer to them easily, and Figure 2(b) shows the sums of the constraint costs of all constraints that involve only agents with known values. These sums correspond to the f -values of an A* search [5] since we assume for simplicity of illustration that all heuristics are zero for our example DCOP problem. For example, node f corresponds to agent x_1 taking on value one, agent x_2 taking on value zero and agent x_3 having an unknown value. Thus, the sum of the constraint costs of all constraints that involve only agents with known values is 20, namely the cost of the constraint that involves agents x_1 and x_2 .

ADOPT [12] is a best-first search algorithm. Best-first

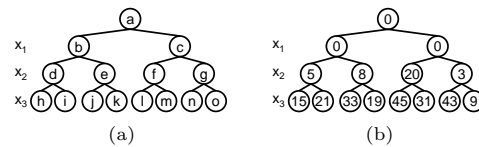


Figure 2: Search Tree

search expands nodes in the search tree for the first time in order of increasing f -values until it finds a solution. For our example DCOP problem, best-first search expands nodes in the search tree for the first time in the order a , b , c , g , d , e , and o . ADOPT is basically a distributed version of RFBS [7]. In order to be memory-bounded, it maintains only a branch from the root node to the currently expanded node and thus needs to repeatedly reconstruct nodes that it purged from memory. For example, it has the branch from a to e in memory when it expands node e but then needs to have the branch from a to o in memory when it expands node o . Thus, it needs to reconstruct the part of this branch from a to g . Depth-first branch-and-bound search, on the other hand, expands the children of a node in order of increasing f -values and prunes those nodes whose f -values are no smaller than the smallest f -value of any leaf node that it has already observed. It backtracks once all children of a node have been expanded or pruned. For our example DCOP problem, depth-first branch-and-bound search expands nodes in the search tree in the order a , b , d , h , (i) , e , (k) , (j) , c , g , and o , where it prunes the nodes in parentheses. It is memory-bounded without having to repeatedly reconstruct nodes that it purged from memory but expands some nodes that a best-first search does not expand, such as node h . Centralized depth-first branch-and-bound search algorithms often run faster than centralized best-first search algorithms. They can be used to solve DCOP problems but typically order the agents completely. Distributed depth-first branch-and-bound search algorithms might be able to solve DCOP problems faster by operating on disjoint subproblems concurrently, as demonstrated by AOBB [10]. One can convert centralized depth-first branch-and-bound search algorithms relatively easily into synchronous distributed DCOP algorithms. Asynchronous distributed DCOP algorithms might be able to solve DCOP problems faster by not having to synchronize the agents tightly.

We therefore develop BnB-ADOPT, a novel asynchronous distributed DCOP algorithm that performs depth-first branch-and-bound search, by using the existing architecture and communication framework of ADOPT, resulting in an asynchronous distributed version of AOBB [10]. However, we do not describe BnB-ADOPT as an extension of ADOPT since this requires the readers to have an in-depth understanding of ADOPT. Instead, we give a stand-alone description of BnB-ADOPT that requires no knowledge of ADOPT, with the intention to create a self-contained and hopefully easy-to-read overview. In the following, we introduce the notation that we need for describing BnB-ADOPT and describe some of its key variables, including how they are updated. We describe a simplified depth-first search version of BnB-ADOPT, which we then enhance by performing branch-and-bound and increasing concurrency. We show the pseudocode of BnB-ADOPT, outline its correctness proof and finally describe our experimental results.

