Pseudo-Tree Based Hybrid Algorithm for Distributed Constraint Optimization

Tenda Okimoto†, Makoto Yokoo††, Yuko Sakurai††, and Katsumi Inoue‡

†Transdisciplinary Research Integration Center, Tokyo 1018430, Japan
††National Institute of Informatics, Tokyo 1018430, Japan
‡Kyushu University, Fukuoka 8190395, Japan
* †{tenda,inoue}@nii.ac.jp, ††{ysakurai,yokoo}@inf.kyushu-u.ac.jp

Abstract. A Distributed Constraint Optimization Problem (DCOP) is a fundamental problem that can formalize various applications related to multi-agent cooperation. Considering pseudo-tree based search algorithms is important in DCOPs, since their memory requirements are polynomial in the number of agents. However, these algorithms require a large run time. Thus, how to speed up pseudo-tree based search algorithms is one of the major issues in DCOPs. In this paper, we propose a novel hybrid algorithm which combines a complete algorithm with an incomplete algorithm. Specifically, we use a state-of-the-art complete search algorithm (BnB-ADOPT) and utilize the bounds obtained by an approximate algorithm (p-optimal algorithm) in preprocessing. In the evaluations, we show that this hybrid algorithm outperforms the state-of-the-art DCOP search algorithm. Furthermore, we verify experimentally that a pseudo-tree based approximate algorithm is well-suited with a pseudo-tree based search algorithm.

1 Introduction

A Distributed Constraint Optimization Problem (DCOP) [11] is a fundamental problem that can formalize various applications related to multi-agent cooperation. A DCOP consists of a set of agents, each of which needs to decide the value assignment of its variables so that the sum of the resulting rewards/costs is optimized. Many application problems in multi-agent systems can be formalized as DCOPs, e.g., distributed resource allocation problems including sensor networks [6], meeting scheduling [7], and the synchronization of traffic lights [5].

It is important to develop a complete algorithm for DCOPs. Various complete algorithms have been developed, e.g., ADOPT [11], BnB-ADOPT [17], DPOP [13], and OptAPO [8]. ADOPT is one of the pioneering DCOP search algorithms and BnB-ADOPT is the state-of-the-art DCOP search algorithm. These two algorithms use a graph structure called a pseudo-tree [15] and find an optimal solution. ADOPT and BnB-ADOPT have identical memory requirements and communication frameworks. The main difference is their search strategies. ADOPT employs a best-first search strategy while BnB-ADOPT utilizes a
depth-first branch-and-bound search strategy. DPOP is a representative pseudo-tree based inference algorithm that adapts the bucket elimination principle [2] to a distributed setting. DPOP works on a DFS traversal of the constraint graph.

Considering a pseudo-tree based search algorithm is important in DCOPs. Since each agent has only a fixed amount of available memory, solving a DCOP with memory bounded algorithms is desirable. The memory requirements of pseudo-tree based search algorithms such as ADOPT and BnB-ADOPT are polynomial in the number of agents.

However, a pseudo-tree based search algorithm requires a large run time. The number of messages sent between agents is exponential in the number of agents in the worst case. Thus, how to speed up pseudo-tree based search algorithms is one of the major issues in DCOPs.

In this paper, we propose a novel hybrid algorithm which combines a complete algorithm with an incomplete algorithm. Specifically, we use a state-of-the-art complete search algorithm BnB-ADOPT and utilize the bounds obtained by a pseudo-tree based $p$-optimal algorithm [12] in preprocessing. In the evaluations, we show that this hybrid algorithm outperforms a state-of-the-art DCOP search algorithm. Furthermore, we verify experimentally that a pseudo-tree based approximate algorithm is well-suited with a pseudo-tree based search algorithm.

Several preprocessing techniques have been introduced to speed up DCOP search algorithms [1, 3, 9]. In BnB-ADOPT, once an approximate solution is given, it is used for pruning, i.e., in principle, BnB-ADOPT can be combined with any approximate algorithm. In this paper, we focus on the $p$-optimal algorithm which is a pseudo-tree based approximate algorithm. Our working hypothesis is that, a pseudo-tree based approximate algorithm should have a good chemistry with a pseudo-tree based search algorithm. More specifically, the $p$-optimal algorithm provides the information on lower and upper bounds for each agent of a pseudo-tree. Compared to using the information on a global solution, we can expect the synergy effect when we utilize the information obtained by the $p$-optimal algorithm. We verify this hypothesis experimentally.

Related Work
ADOPT-DP2 [1] uses dynamic programming based preprocessing technique and search algorithm\(^1\). This technique evaluates the global lower bound for all constraints. The bound is passed up to the root node of the pseudo-tree and then is used to guide the search thresholds in ADOPT. This technique gives the optimal solution in the case that a pseudo-tree has no back edges. When a pseudo-tree has back edges, it only estimates the lower bound since ADOPT-DP2 is a memory bounded algorithm. Compared to ADOPT-DP2, our hybrid algorithm utilizes the $p$-optimal algorithm to generate lower and upper bounds for each agent of a pseudo-tree. Furthermore, this algorithm uses ADOPT while our hybrid algorithm utilizes BnB-ADOPT. In our experimentations, we implement BnB-ADOPT-DP2 and use it instead of ADOPT-DP2 for the comparison.

\(^1\) In [1], three preprocessing techniques have been introduced, DP0, DP1, and DP2 that trade-off between the computation time and quality of the lower bound.
Our contributions are twofold:

- We develop a novel pseudo-tree based hybrid algorithm which is faster compared to the state-of-the-art DCOP search algorithm.
- We show empirically that a pseudo-tree based $p$-optimal algorithm is well-suited with a pseudo-tree based search algorithm BnB-ADOPT. In our algorithm, the $p$-optimal algorithm can provide the detailed information on lower and upper bounds for each agent of a pseudo-tree, which can be used for pruning the search space in BnB-ADOPT.

The rest of this paper is organized as follows. Section 2 formalizes DCOP and describes existing approximate and search algorithms for DCOPs. Section 3 introduces a novel hybrid algorithm for DCOPs, Section 4 evaluates the performance of our hybrid algorithm. Finally, we conclude this paper in Section 5 and provide some perspectives for future work.

2 Preliminaries

In this section, we briefly describe the formalization of Distributed Constraint Optimization Problems (DCOPs) and introduce the $p$-optimal algorithm, ADOPT and BnB-ADOPT which are pseudo-tree based DCOP algorithms.

2.1 DCOP

A Distributed Constraint Optimization Problem (DCOP) is defined by a set of agents $S$, a set of variables $X$, a set of constraint relations $C$, and a set of reward functions $F$. An agent $i$ has its own variable $x_i$. A variable $x_i$ takes its value from a finite, discrete domain $D_i$. A constraint relation $(i, j)$ means there exists a constraint relation between $x_i$ and $x_j$. For $x_i$ and $x_j$, which have a constraint relation, the reward for an assignment $\{(x_i, d_i), (x_j, d_j)\}$ is defined by a reward function $r(d_i, d_j) : D_i \times D_j \rightarrow \mathbb{R}$. For a value assignment to all variables $A$, let us denote

$$R(A) = \sum_{(i, j) \in C, \{(x_i, d_i), (x_j, d_j)\} \subseteq A} r(d_i, d_j).$$

Then, an optimal assignment $A^*$ is given as arg max$_A R(A)$, i.e., $A^*$ is an assignment that maximizes the sum of the value of all reward functions. A DCOP can be represented using a constraint graph, in which a node represents an agent/variable and an edge represents a constraint.

Without loss of generality, we make the following assumptions for simplicity. Relaxing these assumptions to general cases is relatively straightforward:

- Each agent has exactly one variable.
- All constraints are binary.
- Each agent knows all constraint related to its variable.
- the maximal value of each reward function is bounded, i.e., we assume $\forall i, \forall j$, where $(i, j) \in C$, $\forall d_i \in D_i, \forall d_j \in D_j, 0 \leq r_{i,j}(d_i, d_j) \leq r_{\text{max}}$ holds.
Fig. 1. Figure (left) shows a DCOP with three variables $x_1$, $x_2$, and $x_3$. Figure (right) represents a pseudo-tree based on the total ordering $<x_1, x_2, x_3>$ where $x_1$ is the root node.

Example 1 (DCOP). Figure 1 (left) shows a DCOP with three variables $x_1$, $x_2$, and $x_3$. $r(x_i, x_j)$ is a reward function where $i < j$. Each variable takes its value assignment from a discrete domain $\{a, b\}$. The optimal solution that maximizes the sum of the value of all reward functions is $\{(x_1, b), (x_2, a), (x_3, a)\}$, and the optimal value is twelve.

A pseudo-tree [15] is a graph structure, which is widely used in DCOP algorithms, e.g., ADOPT and BnB-ADOPT. In a pseudo-tree, there exists a unique root node, and each non-root node has a parent node. The pseudo-tree contains all nodes and edges of the original constraint graph, and the edges are categorized into tree and back edges.

Example 2 (Pseudo-tree). Figure 1 (right) shows a pseudo-tree based on the total ordering $<x_1, x_2, x_3>$. $x_1$ is the root node of this pseudo-tree. The edge between $x_1$ and $x_3$ represents a back edge and others are tree edges.

2.2 $p$-optimal algorithm

The $p$-optimal algorithm [12] is an approximate DCOP algorithm that can provide guarantees on the quality of the solutions. This algorithm is based on a pseudo-tree. It is a one-shot type algorithm, which runs in polynomial-time in the number of agents $n$ assuming $p$ is fixed. In the $p$-optimal algorithm, agents can adjust parameter $p$ so that they can trade-off better solution quality against computational overhead.

The basic idea of this algorithm is that we remove several edges from a constraint graph so that the induced width [2] of the remaining graph is bounded. Then, we compute the optimal solution ($p$-optimal solution) of the remaining graph, which is used as the approximate solution of the original graph. Induced width can be used as a measure for checking how close a given graph is to a
Fig. 2. Example for $p=1$-optimal algorithm

tree. For example, if the induced width of a graph is one, it is a tree. Also, the induced width of a complete graph with $n$ variables is $n - 1$.

**Definition 1 (Width of pseudo-tree).** For a pseudo-tree and a node $x_i$, we call the number of $x_i$’s ancestors as the width of $x_i$.

**Definition 2 (Induced width of pseudo-tree).** For a pseudo-tree, we call the maximal number of width of all nodes as the induced width of the pseudo-tree.

The $p$-optimal algorithm has the following two phases:

**Phase 1:** Generate a subgraph by removing several edges, so that the induced width of the remaining graph is bounded by parameter $p$.

**Phase 2:** Find an optimal solution to the graph obtained in Phase 1 using a pseudo-tree based complete DCOP algorithm.

In Phase 1, this algorithm simplifies a problem/pseudo-tree instance by removing some edges so that (i) it can solve the simplified problem efficiently, and (ii) it can bound the difference between the solution of the simplified problem and an optimal solution.

Let us describe how we remove the edges from the original pseudo-tree. For node $i$ and $j$, we say an edge $(i, j)$ is a back-edge, if $j$ is $i$’s ancestor (but not $i$’s parent). Also, when $(i, j_1), (i, j_2), \ldots, (i, j_k)$ are all back-edges of $i$, and $j_1 < j_2 < \ldots < j_k$ holds, where $j_i < j_{i+1}$ means that $j_i$ appears before $j_{i+1}$ in the ordering, we call $(i, j_1), (i, j_2), \ldots, (i, j_k)$ as first back-edge, second back-edge, .... $k$-th back-edge, respectively. Clearly, a node has at most $w^* - 1$ back-edges, where $w^*$ is the induced width of the pseudo-tree. For obtaining the pseudo-tree whose induced width is $p$, each agent $i$ simply removes its first back-edge, second
back-edge, ..., \((w^* - p)\)-th back-edge. Intuitive, we remove its back-edges from outside of original pseudo-tree.

In Phase 2, any complete DCOP algorithms can be utilized to find a \(p\)-optimal solution. The \(p\)-optimal algorithm uses the obtained \(p\)-optimal solution as an approximate solution of the original graph. In particular, since we already obtained a pseudo-tree whose induced width is bounded, utilizing pseudo-tree-based DCOP algorithms would be convenient. In this paper, we use a representative pseudo-tree based inference algorithm DPOP [13] for solving a \(p\)-optimal solution in Phase 2.

Example 3 \((p=1\text{-optimal algorithm})\). Figure 2 (i) shows a DCOP with three variables. The induced width of (i) is two. Assume that we want to have a \(p=1\)-optimal solution. (ii) shows the remaining graph which is obtained by removing the back edge between \(x_1\) and \(x_3\) from (i). \(p=1\)-optimal solutions of (ii) are \(\{(x_1, a), (x_2, b), (x_3, a)\}\) and \(\{(x_1, b), (x_2, a), (x_3, b)\}\). The approximate values of (i), i.e., the sum of the rewards obtained by \(p=1\)-optimal solutions, are ten and nine.

2.3 ADOPT

The Asynchronous Distributed OPTimization (ADOPT) [11] is one of the representative DCOP search algorithms. ADOPT utilizes a pseudo-tree and finds an optimal solution employing a best-first search strategy.

We briefly describe the execution of ADOPT.

1. Each node evaluates the cost of the current solution and the cost allocation. The node selects the value of its variable according to the evaluation. The value is notified to descendants which are related by constraints (VALUE message).
2. Each node notifies the cost of the current solution to its parent (COST message).
3. Each node decides the cost allocation between itself and its children. The cost allocation is notified to children (THRESHOLD message).

After repeating the above process, the lower and upper bounds of evaluated cost become equal in the root node. The root node then selects the optimal value of its variable and notifies termination to children. The children search the optimal value of their variables and terminate. Finally, all nodes terminate and their allocated values are the optimal solution. The number of messages sent between agents can be exponential in the number of agents that is given by \(O(D_{\text{max}}^n)\), where \(D_{\text{max}}\) is the maximal domain size and \(n\) is the number of agents.

2.4 BnB-ADOPT

The Branch-and-Bound ADOPT (BnB-ADOPT) [17] is the state-of-the-art DCOP search algorithm. This algorithm utilizes a pseudo-tree and finds an optimal so-
lution using depth-first branch-and-bound search strategy. BnB-ADOPT is introduced as a search algorithm for minimization problems. In this paper, we modify BnB-ADOPT for solving a maximization problem.

BnB-ADOPT is quite similar to ADOPT. This algorithm shares most of the data structures and messages of ADOPT. The main difference is their search strategies. ADOPT employs a best-first search strategy while BnB-ADOPT utilizes a depth-first branch-and-bound search strategy. Also, it has been shown [17] that BnB-ADOPT outperforms ADOPT. The worst case complexity of BnB-ADOPT is the same as ADOPT, which is given by $O(|D_{max}|^n)$.

3 Pseudo-Tree Based Hybrid Algorithm

In this section, we develop a novel pseudo-tree based hybrid algorithm called BnB-ADOPT$_p$ which utilizes the $p$-optimal algorithm and BnB-ADOPT. The basic idea of this hybrid algorithm is that we use the $p$-optimal algorithm in preprocessing and generate lower and upper bounds for each agent. The bounds are then used to guide the search thresholds in BnB-ADOPT. In this algorithm, we assume that each agent knows the pseudo-tree of a constraint graph and $r_{max}$, i.e., the maximal value of each reward function.

This hybrid algorithm has the following two phases:

Phase 1: Find a $p$-optimal solution of the simplified problem by removing several constraints from a problem.

Phase 2: Find an optimal solution to the original problem using the information obtained by Phase 1.

In Phase 1, we compute a $p$-optimal solution of the sub-tree obtained by removing several edges from a pseudo-tree, i.e., we compute an optimal solution of the remaining graph and use it as the lower bound of the original graph. Then, each node/agent $i$ has the following information.

- $A^*_i; p$: $p$-optimal solution of sub-tree rooted at $i$.
- $R(A^*_i); p$: reward obtained by $A^*_i; p$.
- $d^*_i; p$: value of $x_i$ in $A^*_i; p$.
- $m_i$: the number of removed back edges from sub-tree rooted at $i$ in Phase 1.

In Phase 2, we utilize the information obtained by Phase 1, i.e., $A^*_i; p$, $R(A^*_i; p)$, $d^*_i; p$ and $m_i$, and find an optimal solution to the original problem. More specifically, we use the following information as an initial value in BnB-ADOPT.

- $d = d^*_i; p$: value assignment of $x_i$ is $d$.
- $TH_i = R(A^*_i; p)$: threshold $TH_i$ is given by $R(A^*_i; p)$.
- $ub_i = R(A^*_i; p) + (m_i \times r_{max})$: upper bound of $x_i$ is given by the sum of $R(A^*_i; p)$ and $(m_i \times r_{max})$.

2 Most search algorithms, e.g., ADOPT and BnB-ADOPT, have been developed for solving a minimization problem.
We use the following notations. For each node $i$,

- $C_i$: set of children of $i$ in a pseudo-tree.
- $CD_i$: set of its descendants (including its children) that it is involved in edges with.
- $p_i$: parent of $i$.
- $A_i$: set of its ancestors (including its parent).

The pseudo-code of this algorithm is quite similar to that of BnB-ADOPT. The significant differences are that this algorithm solves a maximization problem while BnB-ADOPT solves a minimization problem. Furthermore, this algorithm uses the detailed information for a $p$-optimal solution obtained by Phase 1. More specifically, each agent has the detailed upper bound as an initial value (line 16) while each agent in BnB-ADOPT has its upper bound.

Each agent $i$ maintains a threshold $TH_i$ which is initialized to the sum of the rewards of sub-tree rooted at $i$ (line 5). The threshold of the root agent is $p$-optimal value which is used for pruning during the depth-first search. Each agent $i$ uses the condition to determine whether it should change its own value. If it holds $UB_i(d_i) \leq \max \{TH_i, LB_i\}$ (line 25 and 26). When agent $i$ sends a VALUE message to its child $i' \in C_i$, the message includes the threshold $\max \{TH_i, UB_i\} - \sum_{i'' \in C_i, i'' \neq i'} ub_{i'',i'}(d_i)$ for the child (line 31). This threshold is chosen such that $UB_i(d_i)$ for the child reaches $\max \{TH_i, LB_i\}$ and the agent thus takes on a new value when $LB_i$ for the child reaches this threshold. The agent $i$ changes its context when it receives a VALUE message with a context which is different from its current context. The threshold is also changed to the new threshold in the VALUE message. The agent $i$ initializes $lb_{i,i'}(d) = h_{i,i'}(d), ub_{i,i'}(d) = R(A^*_i) + (m_i \times r_{max})$ (line 15 and 16), $LB_i(d)$, $UB_i(d)$, $LB_i$ and $UB_i$ for all values $d \in D_i$ and children $i' \in C_i$ (line 20-24). The agent $i$ takes on the new value $d_i = \arg \max_{d \in D_i} \{UB_i(d)\}$ (line 26) and repeats the process until terminate execution.

**Pseudo-code of BnB-ADOPT**

```plaintext
procedure Start()
[01] Find a $p$-optimal solution;
[02] $d := d^*_{i,p}$;
[03] $X_i := \{(i', d^*_{i',p}, 0) \mid i' \in SCA_i\};$
```

---

3 It is possible to solve a minimization problem with our algorithm. In this paper, since $p$-optimal algorithm solves maximization problems, we modified BnB-ADOPT to solve maximization problems. Otherwise, we use the original BnB-ADOPT and should modify the $p$-optimal algorithm for minimization problems.
ID_i := 1;
TH_i := R(A_{i,p}^*);
forall i' ∈ C_i, d ∈ D_i
    InitChild(i', d);
Backtrack();
loop forever
  if(message queue is not empty)
      while(message queue is not empty)
          pop msg of message queue;
      When Received (msg);
      Backtrack();
procedure InitChild (i', d)
    lb_{i,i'}(d) := h_{i,i'}(d);
    ub_{i,i'}(d) := R(A_{i,p}^*) + (m_i × r_{max});
procedure InitSelf()
    d_i := arg max_{d ∈ D_i} \{ δ_i(d) + \sum_{i' ∈ C_i} lb_{i,i'}(d) \};
    ID_i := ID_i + 1;
    TH_i := R(A_{i,p}^*);
procedure Backtrack()
    forall d ∈ D_i
        LB_i(d) = δ_i(d) + \sum_{i' ∈ C_i} lb_{i,i'}(d);
        UB_i(d) = δ_i(d) + \sum_{i' ∈ C_i} ub_{i,i'}(d);
    LB_i = max_{d ∈ D_i} \{ LB_i(d) \};
    UB_i = max_{d ∈ D_i} \{ UB_i(d) \};
    if(UB_i(d_i) ≤ max{TH_i, LB_i})
        d_i := arg max_{d ∈ D_i} \{ UB_i(d) \}
        ID_i := ID_i + 1;
    if((i is root and UB_i = LB_i) or termination received)
        Send (TERMINATE) to each i' ∈ C_i;
        terminate execution;
    Send (VALUE, i, d_i, ID_i, max{TH_i, LB_i} - δ_i(d_i) - \sum_{i' ∈ C_i} lb_{i,i'}(d_i)) to each i' ∈ C_i;
    Send (VALUE, i, d_i, ID_i, ∞) to each i' ∈ CD_i \ C_i;
    Send (COST, i, X_i, LB_i, UB_i) to p_i if i is not root;
procedure When Received (VALUE, p, d_p, ID_p, TH_p)
    X' := X_i;
    PriorityMerge ((p, d_p, ID_p), X_i);
    if(!Compatible (X', X_i))
        forall i' ∈ C_i, d ∈ D_i
            if(p ∈ SCA_{i'})
                InitChild (i', d);
        InitSelf();
        if(p = p_i)
            TH_i := TH_p;
procedure When Received (COST, e, X_e, LB_e, UB_e)
    X' := X_i;
    PriorityMerge (X_e, X_i);
    if(!Compatible(X', X_i))
forall $i' \in C_i$, $d \in D_i$

if(!Compatible($\{(i'', d'', ID'') \in X' \mid i'' \in SCA_{c'}, X_i\}$))

InitChild($i', d'$);

if(Compatible($X_c', X_i$))

$lb_{i,c}(d) := \max\{lb_{i,c}(d), LB_c\}$

for the unique $(i', d, ID) \in X_c$ with $i' = i$;

$ub_{i,c}(d) := \min\{ub_{i,c}(d), UB_c\}$

for the unique $(i', d, ID) \in X_c$ with $i' = i$;

InitSelf();

procedure When Received (TERMINATE)

record termination message received;

4 Experimental Evaluation

In this section, we evaluate our hybrid algorithm and compare with DCOP search algorithms BnB-ADOPT-DP2 and BnB-ADOPT. Since BnB-ADOPT outperforms ADOPT, we implement BnB-ADOPT-DP2 and use it instead of ADOPT-DP2 [1] for our comparison. Note that we solve maximization problems. In our evaluations, we use the following problem instances. The domain size of each variable is three, and we choose the reward value uniformly at random from the range $[0,1000]$. Each data point in a graph represents an average of 50 problem instances. For comparison, we mostly use setting $p=1$. For a minimization problem with positive cost, one knows the optimistic bound which is zero. On the other hand, there is no corresponding value when we solve a maximization problem. Since we know that $r_{max}$, i.e., the maximal value of each reward function, is equal to 1000, we set $(r_{max} \times m)$ for the upper bound of BnB-ADOPT instead of $\infty^5$, where $m$ is the number of all constraints of a graph. We implemented our hybrid algorithm in Java and carried out all experiments on 2.53GHz core with 4GB of RAM.

Let us explain how we measure the performance of algorithms in our comparison. We use Non-Concurrent Constraint Checks (NCCC) [10]. NCCC are a weighted sum of processing and communication time. Every agent holds a counter of computation steps. Every message carries the value of the sending agent’s counter. When an agent receives a message, it stores the data received together with the corresponding counter. When the agent first uses the received counter it updates its counter to the largest value between its own counter and the stored counter value which was carried by the message. By reporting the cost of the search as the largest counter held by some agent at the end of the search, a measure of non-concurrent search effort that is close to Lamport’s logical time.

---

4 The results are included Phase 1 and 2.

5 When we set the upper bound $\infty$ in maximization problems, BnB-ADOPT is brute force approach.
is achieved. If instead of steps of computation, the number of NCCCs is counted, then the local computational effort of agents in each step is measured.

First, we show that our hybrid algorithm can find an optimal solution quickly when the parameter $p$ increases. Figure 3 shows the performance of BnB-ADOPT$_p$ for graphs with 10 nodes, induced width 5, varying the parameter $p$ from 0 to 5. When the parameter $p$ is zero, the result shows the average number of NCCCs in BnB-ADOPT. In case that the parameter $p$ is five, our algorithm uses the information for optimal solutions in Phase 2. We can see that the average number of NCCCs in our hybrid algorithm becomes smaller when the parameter $p$ increases. The average number of NCCCs in BnB-ADOPT is 33887 and 4175 in BnB-ADOPT$_{p=5}$. This is because the number of removed edges is small, i.e., the $p$-optimal algorithm can provide more detailed information on lower and upper bounds for BnB-ADOPT (Phase 1). In our hybrid algorithm, we can adjust parameter $p$, so that we can trade-off smaller run time against memory overhead. If the relaxed problem is not so different from the original problem, i.e., the induced width is small, our algorithm can find an optimal solution quickly.

Next, we show that our hybrid algorithm outperforms a state-of-the-art search algorithm. For the comparison, we use BnB-ADOPT-DP2 and BnB-ADOPT. Figure 4 represents the average number of NCCCs in BnB-ADOPT$_{p=1}$, BnB-ADOPT-DP2 and BnB-ADOPT for graphs with induced width 3, density 0.3, varying the number of nodes. BnB-ADOPT$_{p=1}$ utilizes the information of $p=1$-optimal solution, i.e., the optimal solution of the tree obtained by removing several edges from a pseudo-tree. We can see that BnB-ADOPT$_{p=1}$ outperforms BnB-ADOPT-DP2 and BnB-ADOPT. When the number of nodes is 19, the average number of NCCCs is 28063 in BnB-ADOPT$_{p=1}$, 33567 in BnB-ADOPT-
Fig. 4. Comparison of BnB-ADOPT\textsubscript{p=1}, BnB-ADOPT-DP2 and BnB-ADOPT for graphs with induced width 3, density 0.3, varying the number of nodes.

DP2, and 51306 in BnB-ADOPT. BnB-ADOPT\textsubscript{p=1} performs approximately 16\% better than BnB-ADOPT-DP2 and 45\% better compared to BnB-ADOPT. Also, in case that the number of nodes is 11, the average number of NCCCs is 344 in BnB-ADOPT\textsubscript{p=1}, 412 in BnB-ADOPT-DP2, and 640 in BnB-ADOPT. BnB-ADOPT\textsubscript{p=1} performs approximately 16\% better than BnB-ADOPT-DP2 and 46\% better compared to BnB-ADOPT. We obtained similar results varying the density and the induced width.

Furthermore, we verify our hypothesis experimentally, i.e., a pseudo-tree based approximate algorithm should have a good chemistry with a pseudo-tree based search algorithm. Figure 5 represents the average number of NCCCs in BnB-ADOPT\textsubscript{p=1} for pseudo-trees where the depth is nine, with 15 nodes, induced width 3, varying the depth from 0 to 9. In case that the depth is zero, BnB-ADOPT\textsubscript{p=1} does not use the information obtained by Phase 1, i.e., it behaves like BnB-ADOPT. When the depth is one, we give the information obtained by Phase 1 only to the root agent of a pseudo-tree, that is, BnB-ADOPT\textsubscript{p=1} utilizes only global lower and upper bounds of all constraints. In case that the depth is nine, the result shows the average number of NCCCs in BnB-ADOPT\textsubscript{p=1} where every agent uses the information obtained by Phase 1. We can see that the average number of NCCCs becomes smaller when the depth increases. This is because the \textit{p}-optimal algorithm can provide the information on lower and upper bounds to each agent of the pseudo-tree which is used in BnB-ADOPT (Phase 2), that is, the pseudo-tree based \textit{p}-optimal algorithm is well-suited with pseudo-tree based search algorithm BnB-ADOPT. Also, in case that the depth is zero, the average number of NCCCs in BnB-ADOPT\textsubscript{p=1} (BnB-ADOPT) is smaller compared to that in BnB-ADOPT\textsubscript{p=1} where the depth is one. This is because
the run time of preprocessing does not count since it behaves like BnB-ADOPT when the depth is zero.

Moreover, the average number of NCCCs is almost the same when the depth is less than six. However, it reduces extremely at the point where the depth is six. We examined the number of agents who used the information at this critical (tipping) point and it was 9 of 15 agents, i.e., 60% of all agents. We obtained the similar results with different parameter $p$. For BnB-ADOPT$_{p=2}$, the critical point appears at the point where the depth is five. Our future works include more detailed analysis for this critical point. We will examine the number of agents who use the information at the critical point in different parameter settings, e.g., the number of agents, density and induced width, and also in different graph structures such as scale-free and small world graphs.

In summary, these experimental results reveal that (i) our hybrid algorithm can find an optimal solution quickly when the parameter $p$ increases, (ii) BnB-ADOPT$_{p=1}$ outperforms BnB-ADOPT-DP2 and BnB-ADOPT, and (iii) the pseudo-tree based $p$-optimal algorithm have a good chemistry with pseudo-tree based BnB-ADOPT.

Let us describe why BnB-ADOPT$_{p=1}$ outperforms BnB-ADOPT-DP2. This is because our algorithm can utilize the detailed information on the lower and upper bounds obtained by the $p$-optimal algorithm for each agent, which can be used for pruning the search space in BnB-ADOPT. On the other hand, BnB-ADOPT-DP2 can utilize only the global upper bound, i.e., only the root node has the information. Our future works include more detailed analysis for pruning, e.g., examining the frequency of pruning and at which node pruning occurs.
From the verification results of our hypothesis, we expect that our hybrid algorithm outperforms other hybrid algorithms which utilize BnB-ADOPT and not pseudo-tree based approximate algorithms, e.g., DALO [16] and the bounded max-sum algorithm [14]. Since they are not pseudo-tree based algorithms, they can provide only global upper bounds. On the other hand, our algorithm can provide lower and upper bounds for each agent.

5 Conclusion

In this paper, we focus on preprocessing and develop a novel pseudo-tree based hybrid algorithm called BnB-ADOPT$_p$ which utilizes the $p$-optimal algorithm and the representative, state-of-the-art search algorithm BnB-ADOPT. This algorithm uses the $p$-optimal algorithm in preprocessing to generate lower and upper bounds for each agent of a pseudo-tree, which can be used for pruning the search space in BnB-ADOPT. In the evaluations, we showed that our hybrid algorithm outperforms BnB-ADOPT-DP2 and BnB-ADOPT. Furthermore, we verified experimentally that the pseudo-tree based $p$-optimal algorithm is well-suited with pseudo-tree based BnB-ADOPT.

As future works, we will examine detailed analysis for critical points in different parameter settings and graph structures. Furthermore, we will analyze the frequency of pruning in search space. ADOPT($k$) [4] is a search algorithm that generalizes ADOPT and BnB-ADOPT. It behaves like a hybrid algorithm of ADOPT and BnB-ADOPT when $1 < k < \infty$. The comparison with ADOPT($k$) is our future work. Also, we will compare our algorithm with BnB-ADOPT utilizing other preprocessing techniques [3, 9].

Acknowledgment

This research is supported by grant for the Systems Resilience project from the Transdisciplinary Research Integration Center in Japan.

References


