Dilemma First Search
for Effortless Optimization of NP-hard Problems

Julien Weissenberg
CVL, ETH Zurich

Hayko Riemenschneider
CVL, ETH Zurich

Ralf Dragon
CVL, ETH Zurich

Luc Van Gool
CVL, ETH Zurich

Abstract—**To tackle the exponentiality associated with NP-hard problems, two paradigms have been proposed. First, Branch & Bound, like Dynamic Programming, achieve efficient exact inference but requires extensive information and analysis about the problem at hand. Second, meta-heuristics are easier to implement but comparatively inefficient. As a result, a number of problems have been left unoptimized and plain greedy solutions are used. We introduce a theoretical framework and propose a powerful yet simple search method called Dilemma First Search (DFS). DFS exploits the decision heuristic needed for the greedy solution for further optimization. DFS is useful when it is hard to design efficient exact inference. We evaluate DFS on two problems: First, the Knapsack problem, for which efficient algorithms exist, serves as a toy example. Second, Decision Tree inference, where state-of-the-art algorithms rely on the greedy or randomness-based solutions. We further show that decision trees benefit from optimizations that are performed in a fraction of the iterations required by a random-based search.**

I. Introduction

Search algorithms are of prime importance to deal with the intractable search space associated with NP-hard problems. Search algorithms are strategies which guide the search to find approximate solutions to problems where there is no principled way of inferring an exact one. This is helpful when the search space is too large to be explored in a brute-force fashion. However, search algorithms have a challenging nature. Specifically, they can be (1) efficient yet problem-specific and hard to design, (2) relatively inefficient, or (3) stochastic. Efficient (1) search algorithms such as Branch & Bound (BB) require appropriate problem-specific bound estimates which can take decades to be found [7]. In contrast, (2) some, e.g. Genetic Algorithms (GA), are relatively easy to implement but their optimization is only based on the final outcome. That is, they sample a candidate solution and test its performance, ignoring some of the information that could be given by a decision heuristic at intermediate steps. Further, (3) stochastic search algorithms suffer from repeatability issues.

In this paper, we present a novel deterministic and efficient search algorithm, denoted as Dilemma First Search (DFS), to tackle problems which can be framed as a sequence of actions, e.g. decisions or iterations. We present a framework to retrieve candidate solutions ordered by their likelihood of being optimal. DFS is derived from this framework, making few assumptions. We show in the experiments that the assumptions in DFS are reasonable and yield convincing results for two different NP-hard problems. We consider DFS as "effortless" since the only problem-specific requirement is a decision heuristic, which is needed in any case to build a greedy solution. DFS is a backjumping algorithm, which uses the

evidence at the time of constructing the sequence of actions that some decisions are harder to make than others (dilemmas). In order to improve an existing solution, we reconsider such dilemmatic actions first. A decision between several actions is hard to make when they are deemed equally likely to lead to the optimal solution. Intuitively, solving a search problem consists in finding the sequence of *right* decisions. Since not all decision combinations can be explored, it is sensible to explore the most likely combinations first. These can be derived by changing decisions associated with the highest uncertainty first. Our main contributions are:

- a novel search algorithm, Dilemma First Search,

- a probabilistic theoretical framework for search algorithms in general,

- optimizations based on DFS for decision tree inference and Knapsack type problems.

II. Related Work

Given the many topics related to search algorithms, we present a brief overview and focus on the two exemplar problems, i.e. the Knapsack and the inference of decision trees.

Some of the most widely used search algorithms are compared in Table I. For a complete overview, we refer the reader to [15]. We distinguish between deterministic and stochastic approaches. First, backtracking (BT) is the most general deterministic concept. It is guaranteed to find the optimal solution, but suffers from not using heuristics to decide upon the order of the search. Limited Discrepancy Search (LDS) [11] is a form of backtracking assuming the best solutions are more likely to be within a local neighborhood from the greedy solution. Tabu Search [9] prevents the search algorithm from re-visiting similar states, thus enforcing diversity. When an appropriate lower bound can be found, Branch & Bound (BB) and related Branch & Cut (BC) prove to be some of the most efficient search algorithms. We refer the reader to [30] for an overview about BB and for other state-of-the-art methods in this domain to [27], [21]. Unfortunately, finding a good formulation can be very tedious and it is not unusual that it takes decades [7]. Stochastic methods such as Genetic Algorithms (GA) and Simulated Annealing (SA) use randomness to be able to overcome local minima. However, their inherent randomness makes repeatability an issue. Monte Carlo tree search (MCTS) algorithms have been successfully used in games, e.g. Go. MCTS backpropagates the result of a playout to gather information about the payout distribution of actions in the tree. The problem is analogous to a multi-armed bandit problem. Thus, MCTS algorithms, e.g. UCT [13],

		Greedy	GA	BB	LDS	Tabu	SA	DFS
	Stochastic	-	✓	-	-	-	✓	-
Need	Lower bound	-	-	✓	-	-	-	-
	Decision heuristic	✓	-	-	✓	-	-	✓
Efficiency	Redundant computation	-	✓	-	-	-	✓	-
	Space requirement	small	little	little	adjustable	adjustable	little	adjustable
	Fast initial solution	✓	-	-	✓	✓	✓	✓
	Reuse computation	-	-	✓	✓	-	-	✓
	Optimal solution?	-	-	✓	✓	✓	-	✓

TABLE I.   OVERVIEW OF SEARCH ALGORITHMS CONSIDERING THEIR REQUIREMENTS AND EFFICIENCY COMPARED TO OUR DILEMMA FIRST SEARCH.

use random sampling and balance exploitation vs. exploration. This last problem is addressed by many reinforcement learning approaches, such as R-max [4]. The key difference between MCTS and DFS is their application domains. In MCTS, the probability distribution associated with actions is initially unknown, while DFS uses a *known* probability distribution over the actions. For all problems where a greedy solution exists, such a distribution is available.

The Knapsack problem has been studied in depth because it is a very simply formulated NP-hard problem. [10] gives a comprehensive overview of the problem. In decision tree inference, the idea is that a tree is constructed, wherein each node denotes a decision which splits up the data to be classified or regressed into subsets. The leaves of such trees are used for Classification and Regression problems [6]. In sequence, the works on ID3 [23] and C4.5 trees by Quinlan [24] focus on optimal attribute selection. All possible attributes are tested and the one giving the maximal information gain (by entropy or Gini index) is chosen. Yet this formulation uses a greed optimization which is particularly limiting for high-dimensional data. The continuation of research led to a combination of weak (pruned C4.5) classifiers to achieve better generalization. Forests of weak random decision trees were introduced by [1], where the data is subsampled in multiple ways and the parameters are randomly selected [5]. Furthermore, much research has focused on how to best combine the ensembles of classifiers [22], [5]. Finally, the optimization of decision trees has been studied by [2], [17] and very recently is an ongoing topic [25], [20]. In all, the limits of decision trees can be summarized in their ability to handle high-dimensional data, lack of global decision viewing, lack of generalization and extensive requirement of resources in memory and computational power. Some of these limits have been overcome in random forests by introducing randomness in the selection of training data and parameters, as well as using multiple trees. As a result, training yields non-repeatability issues and testing times are higher due to the use of numerous trees. In contrast, we focus on improving a single tree by optimizing it.

In summary, Dilemma First Search (DFS) proposes an optimized search strategy to explore the search space in a principled way by making full use of the information given by a decision heuristic.

III.   A PROBABILISTIC FRAMEWORK FOR SEARCH ALGORITHMS

We introduce a probabilistic framework for decision making for a sequence of actions. Further, we show how the exploration of the solution space can be optimally performed.

A. Best sequence of action problem statement

Formally, the problem $(\mathcal{S}, \mathcal{A}, P, E)$ is considered, where $\mathcal{S}$ is a finite set of states S and $\mathcal{A}$ is a finite set of actions A. The state $S = (A_1, ..., A_N)$ is defined by a sequence of N actions. Each action is associated with a probability $P_{\text{best}}(A_i|S_k)$, as defined in Definition 1.

Definition 1. *Denote by $\hat{S}^*(S_k)$ the optimal candidate solution in a subtree whose root is S_k. $P_{best}(A_i|S_k)$ is the probability that $\hat{S}^*(S_k)$ is contained in the subtree given by taking action A_i from state S_k.*

We call a state candidate solution $\hat{S}$ if it solves the problem. Let $\hat{\mathcal{S}}_{exp}$ and $\hat{\mathcal{S}}_{unexp}$ denote the sets of explored and unexplored candidate solutions, and $\mathcal{S}_{exp}$ and $\mathcal{S}_{unexp}$ the sets of explored and unexplored states. An algorithm which aims at finding the optimal solution, i.e. at finding $\hat{S}^* = \arg\min_{\hat{S}} E(\hat{S})$ is referred to as a search algorithm. Note that $P_{best}(A_i|S_k)$ and the energy function $E(\hat{S})$ are problem-specific. Due to the combinatorial nature, $(\mathcal{S}, \mathcal{A}, \mathcal{P}, E)$ is an NP-hard problem. We can represent this problem by a tree structure, where nodes are states and edges are actions: $S_k \xrightarrow{A_k \in \mathcal{A}_k} S_{k+1}$, where $\mathcal{A}_k$ is the set of actions at state S_k. In this tree, each path from the root to a leaf corresponds to a candidate solution $\hat{S}$. Please note that selecting the maximum $P_{\text{best}}(A_k|S_k)$ at each state gives the greedy solution. An example of a search tree is given in Fig. 1.

Given an unknown, arbitrary and limited amount of computational resources, we want to maximize the probability that we retrieve the optimal solution. Hence, we want to evaluate candidate solutions in order of their likelihood of being optimal.

B. Fully explored search tree

When the tree is fully explored, the probability of each candidate solution $\hat{S}$ being the optimal $\hat{S}^*$ is given by $P_{\text{opt}}(\hat{S})$:

$$P_{\text{opt}}(\hat{S}) = \prod_{k=0}^{\text{depth}(\hat{S})} P_{\text{best}}(A_k|S_k). \tag{1}$$

Please note, the tree needs to be fully explored to guarantee the candidate solutions are evaluated in order of decreasing probability of being the optimal: when the tree is partially explored, the probabilities $P_{\text{best}}(A_k|S_k)$ are not all known.

C. Partially explored search tree

Exploring the full tree is time and memory consuming and can often not be done in practice. Therefore, we require an

anytime algorithm. An anytime algorithm is one which is able to return intermediate solutions of increasing quality, i.e. which can be stopped at "any time" [12]. The idea is to estimate which state to explore first in a partially explored tree. In order to do so, we have to be able to compute the expected probability of partially explored candidate solutions $\hat{S}$. For this, we make the following assumptions.

Assumption 1. $P_{best}(A|S_{unexp}) = P_{uni}$ *i.e. the expected values of the probabilities of actions are uniformly distributed in unexplored parts of the tree.* $P_{uni} = \frac{1}{b}$, *where b is the tree branching factor.*

Assumption 2. *The expected depth d of the tree is uniform and known.*

Assumption 3. *The branching factor b of the tree is constant and known.*

In the following, we first explain how to calculate candidate solution probabilities in the tree. We show that the greedy solution is most likely the best candidate solution (base case). In addition, given a partially explored tree where the first n most likely candidate solutions have been found, the $n+1^{th}$ most likely candidate solution can be inferred (inductive step).

We first calculate the expected probability of each candidate solution consisting of explored and unexplored states following As. 1, 2 and 3:

$$P_{\text{opt}}(\hat{S}) = \prod_{k=1}^{\text{explored}} P_{\text{best}}(A_k|S_k) \cdot \prod_{k=\text{explored}+1}^{d} P_{\text{uni}} \qquad (2)$$

where A_k maximizes P_{best} at S_k and P_{uni} is the uniform probability distribution.

Lemma 1. $P_{uni} \leq P_{greedy}(A_k|S_k) \leq 1$
where P_{uni} is the uniform probability distribution and $P_{greedy}(A_k|S_k) = \arg\max_A P_{best}(A_{i,k}|S_k)$, $A_{i,k}$ being the $i^{t}h$ action which can be taken at state S_k.

Theorem 1 (Base case)**.** *When the tree is unexplored, the greedy solution finds $\arg\max_{\hat{S}} P_{opt}(\hat{S})$ when exploring at most depth d nodes.*

We now want to find the next candidate solution that maximizes $P_{\text{opt}}(\hat{S})$.

Theorem 2 (Induction)**.** *Given the set of nth most likely candidate solutions have been explored, the $n+1^{th}$ most likely candidate solution are inferred by exploring at most d more nodes.*

Corollary 1. *The average number of candidate solutions to be evaluated in order to find $\hat{S}^*$ is smallest when α is chosen according to $\arg\min_\alpha(L_{opt}(\hat{S}_n) - L_\alpha)$, which can be approximated by $\arg\min_\alpha(L_{best}(S_\alpha) - L'_{best}(S_\alpha) - depth(\alpha) \cdot const.)$ (See Appendix in supp. mat. for the derivations).*

D. Search algorithm strategies

We now informally examine the implications of the results of the previous section and relate them to existing search algorithms. Note that probabilities can be approximated by a normalized decision heuristic. If the probability distribution

has a low entropy, the search is expected to be more efficient. Conversely, if probabilities are distributed more uniformly, the search is expected to require more iterations since less information can be leveraged. First, if the computational resources are limited to computing a single candidate solution, then computing the greedy solution yields the best chances of obtaining the optimal solution. Moreover, from Theorem 2 and Eq. (2), altering high-quality sets of solutions is expected to yield higher than average quality candidate solutions. In particular, Simulated Annealing (SA) and Genetic Algorithms (GA) respectively use the best candidate or set of candidates so far to build new candidate solutions. It is also sensible to substitute the greedy solution with the second best choice when backtracking, as in Limited Discrepancy Search (LDS). However, LDS is sub-optimal as it performs back-tracking but does not select the optimal node to explore given the information at hand. Finally, we note that Branch & Bound (BB) assumes more information is available, namely an estimate of the lower and upper bounds. These are not always easily available and have to be manually designed for each problem. In the following section, we propose an algorithm based on the theoretical derivations presented so far.

IV. DILEMMA FIRST SEARCH

An efficient search algorithm based on the probabilistic search framework consists of three parts: (1) selecting a dilemmatic state $S_\alpha = (A_1, ..., A_n)$; (2) changing its action A_n to A'_n; (3) growing a complete candidate solution starting from the partial solution of the selected state S_α. An example illustration of these steps for improving a decision tree is shown in Figure 1. Before we detail our method and its properties, we give an intuition of the method.

Intuition: The proposed method is similar to the way humans improve sequences of decisions. Suppose you would like to find the fastest path to the top of a mountain without a map. At every intersection, a decision has to be made. The steepness of the path is a good indicator and can be used as a decision guide. A first route can be obtained by taking the path with the most appropriate steepness (greedy search). To improve an existing route (candidate solution), it is sensible to first revisit the path intersections (states) where you were most hesitating. At dilemmatic intersections the selected path was not considerably better than the second best. In other words, going up the 2nd-best option is likely to yield a new, competitive route (2nd-best candidate solution).

Algorithm: DFS optimizes a sequence of actions $\hat{S}$ to minimize an energy function $E(\hat{S})$, by revisiting actions which were most dilemmatic first. When a decision is revisited, it will be altered to the next-best choice indicated by $P(A|S)$ and the subsequent actions from that state are chosen in a greedy fashion until a candidate solution is reached.

As displayed in Algorithm 1, in the first iteration, a greedy solution (l. 5–9) is built. Simultaneously, each explored state S is filed into a dilemma queue $\mathcal{Q}$ (l. 9). When a candidate solution is reached, it is added to the set of candidate solutions $\hat{\mathcal{S}}$ (l. 10). In the following iterations, the state with the largest dilemma estimator $\delta(S_\alpha)$ is selected and removed from the queue (l. 11-12). The dilemma estimator $\delta(S_\alpha)$ is derived as:

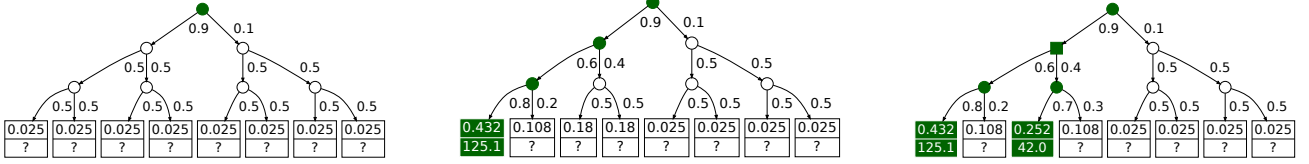$$\delta(S_\alpha) = 1/(L_{\text{best}}(S_\alpha) - L'_{\text{best}}(S_\alpha)) \qquad (3)$$

Fig. 1. Example of Dilemma First Search. Each node corresponds to a state S, each edge is an action associated with a probability $P(A|S)$. The leaf nodes represent the set of candidate solution $\hat{S}$ and are labelled with the probability that they are the optimal solution $\hat{S}^*$, along with their energy $E(\hat{S})$ when explored. Nodes are left blank when they are unexplored and coloured in dark green otherwise. The dilemma node is indicated by a square. In the first iteration (left and centre), the tree is initialized and a greedy solution is built. Then (right), the most dilemmatic node is selected and a new candidate solution is grown.

Algorithm 1: Dilemma First Search (DFS)

Data: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, E)$
Result: $\hat{S}^*$

1 **begin**
2    $\hat{\mathcal{S}}_{exp} \leftarrow \emptyset$; $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{A}_S \leftarrow \emptyset \quad \forall \mathcal{S}$; $S_\alpha \leftarrow ()$;
3   **while** *termination criterion not met* **do**
     /* greedy candidate search   */
4      $S \leftarrow S_\alpha$;
5   **while** *S is not a candidate solution* **do**
6        $A \leftarrow \arg\max_{A \in \mathcal{A} \setminus \mathcal{A}_S} P(A|S)$;
7        $\mathcal{A}_S \leftarrow \mathcal{A}_S \cup \{A\}$;
8        $S \leftarrow (S, A)$;
9        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{S\}$;
10      $\hat{\mathcal{S}}_{exp} \leftarrow \hat{\mathcal{S}}_{exp} \cup \{S\}$;
     /* select dilemma state S_α */
11      $S_\alpha \leftarrow \arg\max_{S \in \mathcal{Q}} \delta(S)$;
12      $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{S_\alpha\}$;
13    $\hat{S}^* \leftarrow \arg\min_{\hat{S} \in \hat{\mathcal{S}}_{exp}} E(\hat{S})$;

Eq. (3) follows from Cor. 1, where const. is assumed to be close to zero. This assumption is valid if the alternative path is as likely to yield the optimal solution as the original path from which it is derived from.

Anytime means that DFS returns intermediate solutions of increasing quality (l. 6). The more computational resources are available, the more likely the solution will be optimal.

Convergence is guaranteed since DFS eventually explores the whole search tree and therefore converges to the optimal solution. Please note, the ordering of exploration is important. DFS retrieves most promising candidate solutions first.

The two following properties show how DFS relates to the probabilistic framework given in Sect. III. **First iteration** of DFS yields the greedy solution, in accordance with Theorem 1. **Subsequent iterations** create new competitive candidate solutions by selecting the most dilemmatic state (see Eq. (3)), switching to the next best action and greedily growing until a candidate solution has been reached.

DFS vs Random state selection (RSS): RSS consists in selecting a state at random from the set of explored states, regardless of their dilemma estimator. In other words, RSS only differs from DFS in that it randomizes the order of the dilemma queue. RSS requires more iterations on average to find $\hat{S}^*$ than selecting the dilemma state based on the optimal S_α (see Corollary 1). Therefore, we investigate RSS as a comparison to DFS in the experimental section.

V. KNAPSACK PROBLEM

In this section, we explain how DFS can be used to search for solutions of the 0-1 Knapsack problem efficiently. The 0-1 Knapsack problem is formulated as follows. Given n items $x_i = (w_i, v_i) \in X$, where w_i is the weight of item x_i and v_i its value, we would like to select a subset of items $X_s \in X$, such that the total profit $\sum_{v_j \in X_s} v_j$ is maximized and the total weight $\sum_{w_j \in X_s} w_j < W$ does not exceed the knapsack capacity W. For this classical NP-hard problem, a pseudo-polynomial time dynamic programming solution [29] and a Branch and Bound solution [14] exist. This section serves as a toy example to explain how DFS is applied.

Greedy approach First, we define a decision heuristic which will help choosing the items. From any current state S, the greedy approach iteratively adds an item which is not in the knapsack, while fitting. We use the heuristic [19]:

$$f(A_i|S) = \frac{v_i}{w_i} \text{ if } w(S) + w_i \leq W, \text{ 0 otherwise} \quad (4)$$

where A_i corresponds to adding item x_i to the Knapsack. The greedy approach is to always choose the decision with the maximum heuristic score, until no more item can be added.

Dilemma First Search for Knapsack In addition to the heuristic $f(A_i|S)$ (see Eq. (4)), we specify an energy function $E(\hat{S}) = v(\hat{S})$ (where $v(\hat{S})$ is the knapsack value, i.e. the sum of the values of items in the Knapsack), and a dilemma estimator $\delta(S)$. The dilemma estimator $\delta(S)$ is calculated following Eq. (3) where $L_{best}(S_\alpha)$ is estimated as $f(A_1|S)$ and $L'_{best}(S_\alpha)$ as $f(A_2|S)$, $f(A_i|S)$ being defined as in Eq. (4).

VI. DECISION TREE OPTIMIZATION

In this section, we detail how Dilemma First Search (DFS) helps in decision tree learning. First, we give a brief overview of the ID3 algorithm in our notation. Then, we explain how to optimize the initial greedy solution.

The decision tree to be created is used to classify data vectors $\vec{x} = (x_1, \ldots, x_n)$. Without loss of generality, let each x_j with $j = 1 \ldots n$ be a categorical value from the finite set of attributes $\mathcal{X}_j$. Each node in the tree partitions the space according to one of the attributes $x_j \in \mathcal{X}_j$. Finally, a leaf node signals the classification result. In order to establish a decision tree, the set of training data $\mathcal{T} = \{(\vec{x}; \text{class}(\vec{x})), \ldots\}$ is used, each component $\vec{x}$ being labelled with $\text{class}(\vec{x}) \in \text{classes}(\mathcal{T})$.

Greedy approach We re-use the heuristic from the ID3 algorithm [23]. Based on the current state $S = (A_1, \ldots, A_{i-1})$ which corresponds to a partially established tree with $(i-1)$ nodes, the action A_i is selecting the attribute for the decision of the next i^{th} node. To determine the attribute $j := A_i$, the information gain is used as heuristics

$$f(A_i|S) = H(\mathcal{T}) - \sum_{x \in \mathcal{X}_j} \frac{|\mathcal{T}_j(x)|}{|\mathcal{T}|} H(\mathcal{T}_j(x)), \qquad (5)$$

where $\mathcal{T}$ is the training data available for the node, $\mathcal{T}_j(x)$ is the subset of $\mathcal{T}$ with the jth attribute being equal to x, and H the class entropy defined by the proportion p of occurrences of class c in $\mathcal{T}$:

$$H(\mathcal{T}) = - \sum_{c \in \text{classes}(\mathcal{T})} p(c|\mathcal{T}) \cdot \log_2 p(c|\mathcal{T}). \qquad (6)$$

After j is determined, the child nodes with all $x \in \mathcal{X}_j$ are explored using Eq. (5) with the filtered training data $\mathcal{T}' = \mathcal{T}_j(x)$. The algorithm finishes when all the attributes have been used, a maximum depth is reached, or when the leaf nodes have a small enough entropy.

Dilemma First Search for ID3 Having defined the ID3 heuristic $f(A|S)$ in Eq. (5), we now specify the energy function $E(\hat{S})$ and the dilemma estimator $\delta(S|\mathcal{A}_S)$ according to Section 13. To prevent overfitting, we define the energy E as the number of mis-classifications over a validation set $\mathcal{V}$ which is disjoint to the training data as

$$E(\hat{S}) = |\{\vec{x} \in \mathcal{V} | d_S(\vec{x}) \neq \text{class}(\vec{x})\}|, \qquad (7)$$

where $d_S(\vec{x})$ is the classification of the data vector $\vec{x}$ carried out with the tree specified by the candidate solution S. The dilemma estimator $\delta(S)$ is calculated following Eq. (3) where $L_{\text{best}}(S_\alpha)$ and $L'_{\text{best}}(S_\alpha)$ are estimated as $f(A_i|S)$, $f(A_i|S)$ being defined as in Eq. (5).

VII. Experimental Evaluation

We considered four standard datasets for the 0-1 Knapsack problem, as well as two diverse datasets for decision tree optimization. For both Knapsack and Decision Tree, each iteration takes less than 100 ms (unoptimized code). Note that the runtime depends on the times for computing (1) the energy of a candidate solution, (2) the decision heuristics, (3) the sorted dilemma queue. (1+2) are the same than for the greedy algorithm while (3) can be efficiently achieved in $\mathcal{O}(\sqrt{log(n)})$ time with the Fusion tree algorithm by [8]. As discussed above, we note that it is relevant to compare DFS against random state selection. If DFS is an efficient informed search, we should notice that DFS outperforms random state selection on average (from Corollary 1).

A. Knapsack Results

As a first experiment, we analyze the 0-1 Knapsack problem. The greedy solution selects the items in order of best value per weight as described in Section V. This provides an good initial solution. See Figure 2 for results where the greedy

solution is the 0th iteration. We evaluate on four standard datasets [16], [19] of various sizes (24A/E with 24, medium with 100, and large01 with 10000 items). In all cases, DFS is able to improve over the greedy solution in few iterations, although the search space is extremely large. Further we consistently outperform random state selection (averaged over 50 random runs). The random state selection picks a random state and regrows a greedy solution from that state. Therefore, the only difference to DFS occurs in line 15 of the DFS algorithm. Not only is DFS better than random state selection on average for all datasets, it also consistently outperforms the best random state selection result.

B. Decision Tree Results

As a second experiment, we optimize the construction of decision trees for classification. The greedy solution is the one of the ID3 algorithm, which greedily selects the most discriminating attribute for classification. We evaluate our proposed search optimization and its effects on two different datasets. The MNIST dataset [18] is a text character recognition dataset, which consists of size-normalized handwritten digits to be classified. It consists of a 60,000 instance training set and of a 10,000 instance test set. For this dataset, the raw binary image is simply the input to the decision tree. Next, we evaluate DFS optimization on the semantic segmentation dataset ECP2011 [28]. It comprises 104 images and seven semantic classes, where the goal is to classify each pixel into one of the semantic classes. The RGB values of each pixel are quantized into 10 bins each, and each bin combination constitutes a binary attribute. As the protocol suggests, we split the dataset into two disjoint sets containing 70% data for training and validation, and 30% for testing. To analyze the performance of DFS, we evaluate the energy of the testing data according to Eq. (7). For both decision trees, we enforce a maximal depth of eight to prevent the tree from overfitting. This reduction in depth also results in improved classification time, as the entire decision tree classifier is smaller. Please note, a limited depth makes state selection even more crucial.

Figure 2 shows results for the two datasets, where the greedy solution is the 0th iteration in all graphs. Starting from a high mis-classification rate, our method improves the classification rate more efficiently than a random state selection − in ECP2011 by 2.6% and in MNIST by 9.7%. The random state selection achieved a reduction by only 1.9% and 7.6% on average, respectively. Furthermore, DFS is almost always converging faster than any of the random state selection runs. For this problem, the depth of the search tree being fixed, we evaluated over a range for the constant in Cor. 1 according to lower/upper bounds of the estimated probabilities. There was no significant change in the results, confirming that a preference to reuse large parts of the solution is not required.

VIII. Conclusion

In this work, we introduced a theoretical framework for search algorithms, leading to a novel search algorithm called Dilemma First Search (DFS). The main idea is to guide the search by revisiting harder-to-make decisions first and hence increase the probability for a better solution. Dilemma First Search (DFS) is very efficient and principled, yet also very simple to implement. These are useful properties especially
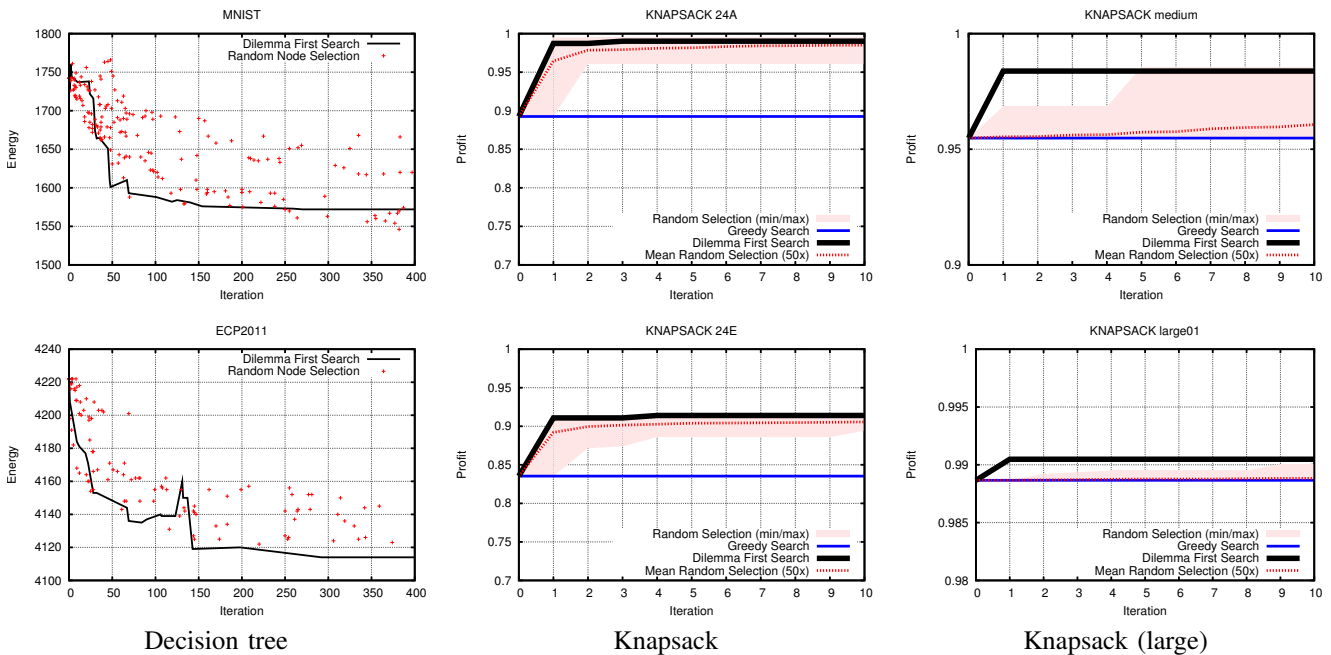
Fig. 2. Comparison on Decision tree (left) and Knapsack optimizations (right) between DFS (black) and Random Node Selection (RNS) (red, as a point cloud for Decision trees and averaged over 50 trials for Knapsack). The 0th iteration corresponds to the greedy solution. RNS picks a state at random and explores the next best candidate solution according to a given heuristic. DFS picks the most dilemmatic node and explores the next best candidate solution according to a given heuristic. Picking the most dilemmatic state results in almost systematic improvement in convergence speed over RNS.

when little information about the problem is known and exact inference is not a viable option. We demonstrated on two distinct problems that search optimization by DFS leads to an overall performance improvement. Future work entails automatically refining the decision heuristic while searching, e.g. by using backpropagation and learning. We also plan to explore much larger data for example from applications in surface fitting [3] or semantic segmentation [26].

References

[1] Y. Amit, G. August, and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 1996.

[2] K. Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, 1994.

[3] A. Bodis-Szomoru, H. Riemenschneider, and L. Van Gool. Superpixel Meshes for Edge-Preserving Surface Reconstruction. In *CVPR*, 2015.

[4] R. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *ML Research*, 2003.

[5] L. Breiman. Random forests. *Machine Learning*, 2001.

[6] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[7] P. Brucker, B. Bernd Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 1994.

[8] M. Fredman and D. Willard. Blasting through the information theoretic barrier with fusion trees. In *ACM Symp. on Theory of Computing*, 1990.

[9] F. Glover. Tabu search – Part I. *Journal on Computing*, 1989.

[10] H. Hans, U. Pferschy, and D. Pisinger. *Knapsack problems*. 2004.

[11] W. Harvey and M. Ginsberg. Limited Discrepancy Search. In *International Joint Conference on Artificial Intelligence*, 1995.

[12] J. Hendler. *Artificial Intelligence Planning Systems*. 1992.

[13] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *ECML*. 2006.

[14] P. Kolesar. A branch and bound algorithm for the knapsack problem. *Management Science*, 1967.

[15] R. Korf. Algorithms and theory of computation handbook. 1999.

[16] D. Kreher and D. Simpson. *Combinatorial Algorithms*. CRC, 1998.

[17] M. Kretowski and M. Grześ. Global learning of decision trees by an evolutionary algorithm. In *Information Processing and Security Systems*. 2005.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *IEEE Speech and Image Processing*, 1998.

[19] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.

[20] M. Norouzi, M. Collins, M. Johnson, D. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. In *NIPS*, 2015.

[21] L. Otten and R. Dechter. Anytime and/or depth-first search for combinatorial optimization. *AI Communications*, 2012.

[22] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *PAMI*, 2010.

[23] J. Quinlan. Induction of decision trees. *Machine Learning*, 1986.

[24] J. Quinlan. *C4.5: Programs for Machine Learning*. 1993.

[25] S. Ren, X. Cao, Y. Wei, and J. Sun. Global refinement of random forest. In *CVPR*, 2015.

[26] H. Riemenschneider, A. Bodis-Szomoru, J. Weissenberg, and L. Van Gool. Learning Where To Classify In Multi-View Semantic Segmentation. In *ECCV*, 2014.

[27] D. Sontag. *Approximate inference in graphical models using LP relaxations*. PhD thesis, MIT, 2010.

[28] O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios. Procedural modeling and image-based 3d reconstruction of complex architectures through random walks. In *IJCV*, 2010.

[29] P. Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 1980.

[30] W. Zhang. Branch-and-bound search algorithms and their computational complexity. Technical report, University of southern California, 1996.