Front. Comput. Sci., 2018, 12(5): 939–949
https://doi.org/10.1007/s11704-017-6315-2

RESEARCH ARTICLE

Achieving data-driven actionability by combining learning and planning

Qiang LV(✉)[1], Yixin CHEN[2], Zhaorong LI[1], Zhicheng CUI[2], Ling CHEN[1], Xing ZHANG[3], Haihua SHEN (✉)[4]

1 College of Information Engineering, Yangzhou University, Yangzhou 225127, China
2 Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis MO 63130, USA
3 School of Management, Fudan University, Shanghai 200433, China
4 School of Computer and Control Engineering, University of Chinese Academy of Science, Beijing 100049, China

Abstract A main focus of machine learning research has been improving the generalization accuracy and efficiency of prediction models. However, what emerges as missing in many applications is actionability, i.e., the ability to turn prediction results into actions. Existing effort in deriving such actionable knowledge is few and limited to simple action models while in many real applications those models are often more complex and harder to extract an optimal solution.

In this paper, we propose a novel approach that achieves actionability by combining learning with planning, two core areas of AI. In particular, we propose a framework to extract actionable knowledge from random forest, one of the most widely used and best off-the-shelf classifiers. We formulate the actionability problem to a sub-optimal action planning (SOAP) problem, which is to find a plan to alter certain features of a given input so that the random forest would yield a desirable output, while minimizing the total costs of actions. Technically, the SOAP problem is formulated in the SAS+ planning formalism, and solved using a Max-SAT based approach. Our experimental results demonstrate the effectiveness and efficiency of the proposed approach on a personal credit dataset and other benchmarks. Our work represents a new application of automated planning on an emerging and challenging machine learning paradigm.

Received June 15, 2016; accepted December 29, 2016

E-mail: lvqiang@yzu.edu.cn, shenhh@ucas.ac.cn

Keywords actionable knowledge extraction, machine learning, planning, random forest

1 Introduction

Research on machine learning has achieved great success on enhancing the models' accuracy and efficiency. Successful models such as support vector machines (SVMs), random forests, and deep neural nets have been applied to vast industrial applications [1]. However, in many applications, users need not only a prediction model, but also suggestions on courses of actions to achieve desirable goals. For practitioners, a complex model such as a random forest is often not very useful even if its accuracy is high because of its lack of actionability. Given a learning model, extraction of actionable knowledge entails finding a set of actions to change the input features of a given instance so that it achieves a desired output from the learning model. We elaborate this problem using one example.

Example 1 In a credit card company, a key task is to decide on promotion strategies to maximize the long-term profit. The customer relationship management (CRM) department collects data about customers, such as customer education, age, card type, the channel of initiating the card, the number and effect of different kinds of promotions, the number and time of phone contacts, etc.

For data scientists, they need to build models to predict the profit brought by customers. In a real case, a company builds a random forest involving 35 customer features. The model predicts the profit (with probability) for each customer. In addition, a more important task is to extract actionable knowledge to revert "negative profit" customers and retain "positive profit" customers. In general, it is much cheaper to maintain existing "positive profit" customers than to revert "negative profit" ones. It is especially valuable to retain high profit, large, enterprise-level customers.

There are certain actions that the company can take, such as making phone contacts and sending promotional coupons. Each action can change the value of one or multiple attributes of a customer. Obviously, such actions incur costs for the company. For instance, there are seven different kinds of promotions and each promotion associates with two features, the number and the accumulation effect of sending this kind of promotion. When performing an action of "sending promotion_amt_N", it will change features "nbr_promotion_amt_N" and "s_amt_N", the number and the accumulation effect of sending the sales promotion, respectively. For a customer with "negative profit", the goal is to extract a sequence of actions that change the customer profile so that the model gives a "positive profit" prediction while minimizing the total action costs. For a customer with "positive profit", the goal is to find actions so that the customer has a "positive profit" prediction with a higher prediction probability.

Besides the example above, ample needs for learning actionability have been reported, such as suggesting medical interventions to avert imminent patient deterioration [2] and changing high-school students' behavior for educational goals [3].

Research on extracting actionability from machine learning models is still limited in machine learning comunity [4–7] and marketing science [8–14]. Most of the approaches are not suitable for the problems studied in this paper due to two major drawbacks. First, they cannot provide customized actionable knowledge for each individual since the rules or rankings are derived from the entire population of training data. Second, they did not consider the action costs while building the rules or rankings. For example, a low income housewife may be more sensitive to sales promotion driven by consumption target, while a social housewife may be more interested in promotions related to social networks. Thus, these rule-based and ranking algorithms cannot tackle these problems very well since they are not personalized for each customer.

Another related work is extracting actionable knowledge from decision tree and additive tree models by bounded tree search and integer linear programming [15–17]. A limitation of these works is that the actions are assumed to change only one attribute each time. As we discussed above, actions like "sending promotion_amt_N" may change multiple features, such as "nbr_promotion_amt_N" and "s_amt_N". Moreover, Yang's greedy method is fast but cannot give optimal solution [15], and Cui's optimization method is optimal but very slow [17].

In order to address these challenges, we propose a novel approach to extract actionable knowledge from random forests, one of the most popular learning models. Our approach leverages planning, one of the core and extensively researched areas of AI. We first rigorously formulate the knowledge extracting problem to a sub-optimal actionable planning (SOAP) problem which is defined as finding a sequence of actions transferring a given input to a desirable goal while minimizing the total action costs. Then, our approach consists of two phases. In the offline preprocessing phase, we use an anytime state-space search on an action graph to find a preferred goal for each instance in the training dataset and store the results in a database. In the online phase, for any given input, we translate the SOAP problem into a SAS+ (multi-valued State vAriableS representation) planning problem. The SAS+ planning problem is solved by an efficient MaxSAT-based (Max SATisfiability) approach capable of optimizing plan metrics.

We perform empirical studies to evaluate our approach. We use a real-world credit card company dataset obtained through an industrial research collaboration. We also evaluate some other standard benchmark datasets. We compare the quality and efficiency of our method to several other state-of-the-art methods. The experimental results show that our method achieves a near-optimal quality and real-time online search as compared to other existing methods.

2 Preliminaries

2.1 Random forest

Random forest is a popular model for classification, one of the main tasks of learning. The reasons why we choose random forest are: 1) In addition to superior classification/regression performance, random forest enjoys many appealing properties many other models lack [18], including the support for multi-class classification and natural handling of missing values and data of mixed types. 2) Often referred to as one of the best off-the-shelf classifier [18], random forest

has been widely deployed in many industrial products such as Kinect [19] and face detection in camera [20], and is the popular method for some competitions such as Web search ranking [21].

Note that random forest is a special case of the Additive Tree Models (ATMs). Our approach can be easily expanded to other additive tree models [22], such as adaboost [23], gradient boosting trees [24]. Thus, the proposed action extraction algorithm has very wide applications.

Consider a dataset $\{X, Y\}$, where $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$ is the set of training samples and $Y = \{y^1, y^2, \ldots, y^N\}$ is the set of classification labels. Each vector $\mathbf{x}^i = (x_1^i, x_2^i, \ldots, x_M^i)$ consists of M attributes, where each attribute x_j can be either categorical or numerical and has a finite or infinite domain $Dom(x_j)$. Note that we use $\mathbf{x} = (x_1, x_2, \ldots, x_M)$ to represent $\mathbf{x}^i = (x_1^i, x_2^i, \ldots, x_M^i)$ when there is no confusion. All labels y^i have the same finite categorical domain $Dom(Y)$.

A random forest contains D decision trees where each decision tree d takes an input $\mathbf{x}$ and outputs a label $y \in Dom(Y)$, denoted as $o_d(\mathbf{x}) = y$. For any label $c \in Dom(Y)$, the probability of output c is

$$p\big(y = c \mid \mathbf{x}\big) = \frac{\sum_{d=1}^{D} w_d I(o_d(\mathbf{x}) = c)}{\sum_{d=1}^{D} w_d}, \tag{1}$$

where $w_d \in \mathcal{R}$ are weights of decision trees, $I(o_d(\mathbf{x}) = c)$ is an indicator function which evaluates to 1 if $o_d(\mathbf{x}) = c$ and 0 otherwise. The overall output predicted label is

$$H(\mathbf{x}) = \underset{c \in Dom(Y)}{\operatorname{argmax}} \; p(y = c|\mathbf{x}). \tag{2}$$

A random forest is generated as follows [25]. For $d = 1, 2, \ldots, D$,

1) Sample n_k ($0 < n_k < N$) instances from the dataset with replacement.
2) Train an un-pruned decision tree on the n_k sampled instances. At each node, choose the split point from a number of randomly selected features rather than all features.

2.2 SAS+ formalism

In classical planning, there are two popular formalisms, STRIPS (STanford Research Institute Problem Solver) and PDDL (Planning Domain Definition Language) [26]. In recent years, another indirect formalism, SAS+, has attracted increasing uses due to its many favorable features, such as compact encoding with multi-valued variables, natural support for invariants, associated domain transition graphs (DTGs) and causal graphs (CGs) which capture vital structural information [27–29].

In SAS+ formalism, a planning problem is defined over a set of multi-valued *state variables* $\mathcal{X} = \{x_1, x_2, \ldots, x_{|\mathcal{X}|}\}$. Each variable $x \in \mathcal{X}$ has a finite domain $Dom(x)$. A state s is a full assignment of all the variables. If a variable x is assigned to $f \in Dom(x)$ at a state s, we denote it as $s(x) = f$. We use $\mathcal{S}$ to represent the set of all states.

Definition 1 (Transition) Given a multi-valued state variable $x \in \mathcal{X}$ with a domain $Dom(x)$, a transition is defined as a tuple $\mathcal{T} = (x, f, g)$, where $f, g \in Dom(x)$, written as $\delta^x_{f \to g}$. A transition $\delta^x_{f \to g}$ is applicable to a state s if and only if $s(x) = f$. We use $\oplus$ to represent applying a transition to a state. Let $s' = s \oplus \delta^x_{f \to g}$ be the state after applying the transition to s, we have $s'(x) = g$. We also simplify the notation $\delta^x_{f \to g}$ as δ^x or δ when there is no confusion.

A transition $\delta^x_{f \to g}$ is a *regular transition* if $f \neq g$ or a *prevailing transition* if $f = g$. In addition, $\delta^x_{* \to g}$ denotes a *mechanical transition*, which can be applied to any state s and changes the value of x to g.

For a variable x, we denote the set of all transitions that affect x as $\mathcal{T}(x)$, i.e., $\mathcal{T}(x) = \{\delta^x_{f \to g}\} \cup \{\delta^x_{* \to g}\}$ for all $f, g \in Dom(x)$. We also denote the set of all transitions as $\mathcal{T}$, i.e., $\mathcal{T} = \bigcup_{x \in \mathcal{X}} \mathcal{T}(x)$.

Definition 2 (Transition mutex) For two different transitions $\delta^x_{f \to g}$ and $\delta^x_{f' \to g'}$, if at least one of them is a mechanical transition and $g = g'$, they are compatible; otherwise, they are mutually exclusive (mutex).

Definition 3 (Action) An action a is a set of transitions $\{\delta_1, \delta_2, \ldots, \delta_{|a|}\}$, where there do not exist two transitions $\delta_i, \delta_j \in a$ that are mutually exclusive. An action a is *applicable* to a state s if and only if all transitions in a are applicable to s. Each action has a cost $\pi(a) > 0$.

Definition 4 (SAS+ planning) A SAS+ planning problem is a tuple $\Pi_{sas} = (\mathcal{X}, \mathcal{O}, s_I, S_G)$ defined as follows

- $\mathcal{X} = \{x_1, x_2, \ldots, x_{|\mathcal{X}|}\}$ is a set of state variables.
- $\mathcal{O}$ is a set of actions.
- $s_I \in \mathcal{S}$ is the initial state.
- S_G is a set of goal conditions, where each goal condition $s_G \in S_G$ is a partial assignment of some state variables. A state s is a goal state if there exists $s_G \in S_G$ such that s agrees with every variable assignment in s_G.

Note that we made a slight generalization of original SAS+

planning, in which S_G includes only one goal condition. For a state s with an applicable action a, we use $s' = s \oplus a$ to denote the resulting state after applying all the transitions in a to s (in an arbitrary order since they are mutex free).

Definition 5 (Action mutex) Two different actions a_1 and a_2 are mutually exclusive if and only if at least one of the following conditions is satisfied:

- There exists a non-prevailing transition δ such that $\delta \in a_1$ and $\delta \in a_2$.
- There exist two transitions $\delta_1 \in a_1$ and $\delta_2 \in a_2$ such that δ_1 and δ_2 are mutually exclusive.

A set of actions P is applicable to s if each action $a \in P$ is applicable to s and no two actions in P are mutex. We denote the resulting state after applying a set of actions P to s as $s' = s \oplus P$.

Definition 6 (Solution plan) For a SAS+ problem $\Pi_{sas} = (\mathcal{X}, O, s_I, S_G)$, a solution plan is a sequence $\mathcal{P} = (P_1, P_2, \ldots, P_L)$, where each P_t, $t = 1, 2, \ldots, L$ is a set of actions, and there exists $s_G \in S_G$, $s_G \subseteq s_I \oplus P_1 \oplus P_2 \oplus \cdots P_L$.

Note that in a solution plan, multiple non-mutex actions can be applied at the same time step. $s \oplus P_t$ means applying all actions in P_t in any order to state s. In this work, we want to find a solution plan that minimizes a quality metric, the *total action cost* $\sum_{P_t \in \mathcal{P}} \sum_{a \in P_t} \pi(a)$.

3 Sub-optimal actionable plan (SOAP) problem

We first give an intuitive description of the SOAP problem. Given a random forest and an input $\mathbf{x}$, the SOAP problem is to find a sequence of actions that, when applied to $\mathbf{x}$, changes it to a new instance which has a desirable output label from the random forest. Since each action incurs a cost, it also needs to minimize the total action costs. In general, the actions and their costs are determined by domain experts. For example, analysts in a credit card company can decide which actions they can perform and how much each action costs.

There are two kinds of features, *soft attributes* which can be changed with reasonable costs and *hard attributes* which cannot be changed with a reasonable cost, such as gender [15]. We only consider actions that change soft attributes.

Definition 7 (SOAP problem) A SOAP problem is a tuple $\Pi_{soap} = (H, \mathbf{x}^I, c, O)$, where H is a random forest, $\mathbf{x}^I$ is a given input, $c \in Dom(Y)$ is a class label, and O is a set of actions. The goal is to find a sequence of actions $A = (a_1, a_2, \ldots, a_n), a_i \in O$, to solve:

$$\min_{A \subseteq O} \; F(A) = \sum_{a_i \in A} \pi(a_i), \tag{3}$$

$$\text{subject to: } \; p(y = c|\tilde{\mathbf{x}}) \geqslant z, \tag{4}$$

where $\pi(a) > 0$ is the cost of action a, $0 < z \leqslant 1$ is a constant, $p(y = c|\tilde{\mathbf{x}})$ is the output of H as defined in Eq. (1), and $\tilde{\mathbf{x}} = \mathbf{x}^I \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_n$ is the new instance after applying the actions in A to $\mathbf{x}^I$.

Example 2 A random forest H with two trees and three features is shown in Fig. 1. x_1 is a hard attribute, x_2 and x_3 are soft attributes. Given H and an input $\mathbf{x} = (\text{male}, 2, 500)$, the output from H is 0. The goal is to change $\mathbf{x}$ to a new instance that has an output of 1 from H. For example, two actions changing x_2 from 2 to 5 and x_3 from 500 to 1500 is a plan and the new instance is $(\text{male}, 5, 1500)$. □

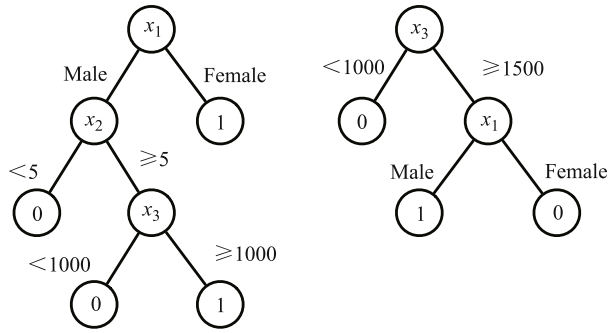


Fig. 1 An illustration of a random forest

4 A planning approach to SOAP

The SOAP problem is proven to be an NP-hard problem, even when an action can change only one feature [17]. Therefore, we cannot expect any efficient algorithm for optimally solving it. We propose a planning-based approach to solve the SOAP problem. Our approach consists of an offline preprocessing phase that only needs to be run once for a given random forest, and an online SAS+ planning phase that is used to solve each SOAP problem instance.

4.1 Offline preprocessing phase

Since there are typically prohibitively high number of possible instances in the feature space, it is too expensive and unnecessary to explore the entire space. We reason that the training dataset for building the random forest gives a repre-

sentative distribution of the instances. Therefore, in the offline preprocessing, we form an action graph and identify a preferred goal state for each training sample.

Definition 8 (Feature partitions) Given a random forest H, we split the domain of each feature x_i ($i = 1, 2, \ldots, M$) into a number of partitions according to the following rules.

1) x_i is split into n partitions if x_i is categorical and has n categories.
2) x_i is split into $n+1$ partitions if x_i is numerical and has n *branching nodes* in all the decision trees in H. Suppose the branching nodes are $(b_1, b_2, \ldots, b_n)$, the partitions are $\{(-\infty, b_1), [b_1, b_2), \ldots, [b_n, +\infty)\}$.

In Example 2, x_1 is splited into {male, female}, x_2 and x_3 are splited into $\{(-\infty, 5), [5, +\infty)\}$ and $\{(-\infty, 1000), [1000, 1500), [1500, +\infty)\}$, respectively.

Definition 9 (State transformation) For a given instance $\mathbf{x} = (x_1, x_2, \ldots, x_M)$, let n_i be the number of partitions and p_i the partition index for feature x_i, we transform it to a SAS+ state $s(\mathbf{x}) = (z_1, z_2, \ldots, z_M)$, where $|Dom(z_i)| = n_i$ and $s(z_i) = p_i, i = 1, 2, \ldots, M$.

For simplicity, we use s to represent $s(\mathbf{x})$ when there is no confusion. Note that if two instances $\mathbf{x}_1$ and $\mathbf{x}_2$ transform to the same state s, then they have the same output from the random forest since they fall within the same partition for every feature. In that case, we can use $p(y = c|s)$ in place of $p(y = c|\mathbf{x}_1)$ and $p(y = c|\mathbf{x}_2)$.

Given the states, we can define SAS+ transitions and actions according to Definitions 1 and 3. For Example 2, $\mathbf{x} = (x_1, x_2, x_3)$ can be transformed to state $s = (x_1, x_2, x_3)$, $Dom(x_1) = \{0, 1\}, Dom(x_2) = \{0, 1\}, Dom(x_3) = \{0, 1, 2\}$. For an input $\mathbf{x} = (\text{male}, 2, 500)$, the corresponding state is $s = (0, 0, 0)$. The action a changing x_2 from 2 to 5 can be represented as $\delta_{0\to1}^{x_2}$. Thus, the resulting state of applying a is $s \oplus a = (0, 1, 0)$.

Definition 10 (Action graph) Given a SOAP problem $\Pi_{soap} = (H, \mathbf{x}^I, c, O)$, the action graph is a graph $G = (\mathcal{F}, E)$ where $\mathcal{F}$ is the set of transformed states and an edge $(s_{i-1}, s_i) \in E$ if and only if there is an action $a \in O$ such that $s_{i-1} \oplus a = s_i$. The weight for this edge is $w(s_{i-1}, s_i) = \pi(a)$.

The SOAP problem in Definition 7 is equivalent to finding the shortest path on the state space graph $G = (\mathcal{F}, E)$ from a given state s_I to a goal state. A node s is a goal state if $p(y = c|s) \geqslant z$. Given the training data $\{X, Y\}$, we use a heuristic search to find a **preferred goal** state for each $\mathbf{x} \in X$ that $p(y = c|\mathbf{x}) < z$. For each of such $\mathbf{x}$, we find a path in the action graph from $s(\mathbf{x})$ to a state s^* such that $p(y = c|s^*) \geqslant z$ while minimizing the cost of the path.

The heuristic search algorithm shows as follows. The search uses a standard evaluation function $f(s) = g(s) + h(s)$. $g(s)$ is the cost of the path leading up to s. Let the path be $s_0 = s_I, s_1, s_2, \ldots, s_m = s$, and $s_{i-1} \oplus a_i = s_i$ for $i = 1, 2, \ldots, m$, we have $g(s) = \sum_{i=1}^{m} \pi(a_i)$. We define the *heuristic function* as $h(s) = \alpha(z - p(y = c|s))$ if $p(y = c|s) < z$, otherwise $h(s) = 0$.

Algorithm Heuristic search (Input: $G = (\mathcal{F}, E)$, s_I)

```
1: N_es ← 0, s* ← NULL, g* ← ∞
2: MinHeap.push(< s_I, f(s_I) >), ClosedList ← {}
3: while MinHeap is not empty do
4:    < s, f(s) >← MinHeap.pop()
5:    if p(y = c|s) ⩾ z and g(s) < g* then
6:       N_es ← |ClosedList|, s* ← s, g* ← g(s)
7:    end if
8:    if |ClosedList| − N_es > Δ then return s*
9:    if s ∉ ClosedList and p(y = c|s) < z then
10:      ClosedList=ClosedList ∪{s}
11:      for each (s, s′) ∈ E do
12:         MinHeap.push(< s′, f(s′) >)
13:      end for
14:   end if
15: end while
16: end return s*
```

For any state $s = s_I \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_m$ satisfying $p(y = c|s) < z$, $f(s) = g(s) + h(s) = \sum_{i=1}^{m} \pi(a_i) + \alpha(z - p(y = c|s))$. Since the goal is to achieve $p(y = c|s) \geqslant z$, $h(s)$ measures how far s is from the goal. α is a controlling parameter. In our experiments, α is set to the mean of all the action costs.

The heuristic search algorithm maintains two data structures, a min heap and a closed list, and performs the following main steps:

1) Initialize N_{es}, s^*, and g^* where N_{es} represents the number of expanded states, s^* is the best goal state ever found, and g^* records the cost of the path leading up to s^*. Add the initial state s_I to the min heap (lines 1 and 2).
2) Pop the state s from the heap with the smallest $f(s)$ (line 4).
3) If $p(y = c|s) \geqslant z$ and $g(s) < g^*$, update g^*, N_{es}, and the best goal state s^* (lines 5 and 6).
4) If the termination condition ($|\text{ClosedList}| - N_{es} > \Delta$) is met, stop the search and return s^* (line 8).

5) Add s to the closed list and for each edge $(s, s') \in E$, add s' to the min heap if s is not in the closed list and not a goal state (lines 10–12).

6) Repeat from Step 2.

The closed list is implemented as a set with highly efficient hashing-based duplicate detection. The search terminates when the search has not found a better plan for a long time ($|\text{ClosedList}| - N_{es} > \Delta$). We set a large value ($\Delta = 10^7$) in our experiments. Note that the heuristic search algorithm does not have to search all states since it will stop the search once a state $\mathbf{s}$ satisfies the termination condition (line 8).

By the end of the offline phase, for each $\mathbf{x} \in X$ and the corresponding state $s(\mathbf{x})$, we find a preferred goal state $s^*(\mathbf{x})$. For an input $\mathbf{x} = (\text{male}, 2, 500)$ in Example 2, the corresponding initial state is $s = (0, 0, 0)$. An optimal solution is $P = (a_1, a_2, a_3)$ where $a_1 = \delta^{x_2}_{0\to 1}$, $a_2 = \delta^{x_3}_{0\to 1}$, $a_3 = \delta^{x_3}_{1\to 2}$, and the preferred goal state is $s = (0, 1, 2)$.

4.2 Online SAS+ planning phase

Once the offline phase is done, the results can be used to repeatedly solve SOAP instances. In the online phase, for any given input, we translate the SOAP problem into a SAS+ planning problem based on the preferred goal states found in the offline phase and then solve it by an efficient MaxSAT-based approach capable of minimizing total action costs.

We now describe how to handle a new instance $\mathbf{x}^I$ and find the actionable plan. In online SAS+ planning, we will find a number of closest states of $s(\mathbf{x}^I)$ and use the combination of their goals to construct the goal $s^*(\mathbf{x}^I)$. This is inspired by the idea of similarity-based learning methods such as k-nearest-neighbor (kNN). We first define the similarity between two states.

Definition 11 (Feature similarity) Given two states $s(x_1, x_2, \ldots, x_M)$ and $s'(x'_1, x'_2, \ldots, x'_M)$, the similarity of the ith feature variable is defined as:

- If the ith feature is categorical, $\xi_i(s, s') = 1$ if $x_i = x'_i$, otherwise $\xi_i(s, s') = 0$.
- If the ith feature is numerical, $\xi_i(s, s') = 1 - |p_i - p'_i|/(n_i - 1)$, where p_i and p'_i are the partition index of features x_i and x'_i, and n_i is the number of partitions of the ith feature.

Note that $\xi_i(s, s') \in [0, 1]$. $\xi_i(s, s') = 1$ means they are in the same partition, while $\xi_i(s, s') = 0$ means they are totally different.

Definition 12 (State similarity) The similarity between two states $s(x_1, x_2, \ldots, x_M)$ and $s'(x'_1, x'_2, \ldots, x'_M)$ is 0 if there exists $i \in [1, M]$, x_i is a hard attribute and x_i and x'_i are not in the same partition. Otherwise, the similarity is

$$sim(s, s') = \frac{\sum_{i=1}^{M} \phi_i \xi_i(s, s')}{\sum_{i=1}^{M} \phi_i}, \tag{5}$$

where ϕ_i is the feature weight in the random forest.

Note that $sim(s, s') \in [0, 1]$. A larger $sim(s, s')$ means higher similarity. Given two vectors $\mathbf{x} = (\text{male}, 2, 500)$ and $\mathbf{x}' = (\text{male}, 6, 800)$ in Example 2, the corresponding states are $\mathbf{s} = (0, 0, 0)$ and $\mathbf{s}' = (0, 1, 0)$. Their feature similarities are $\xi_0(s, s') = 1$, $\xi_1(s, s') = 0$, and $\xi_2(s, s') = 1$. Suppose $\phi_i = 1/3$, then $sim(s, s') = 2/3$.

Given two vectors $\mathbf{x} = (\text{male}, 2, 500)$ and $\mathbf{x}' = (\text{female}, 2, 500)$, the corresponding states are $\mathbf{s} = (0, 0, 0)$ and $\mathbf{s}' = (1, 0, 0)$. Since x_1 is a hard attribute and x_1, x'_1 are not in the same parition, $sim(s, s') = 0$.

SAS+ formulation Given a SOAP problem $\Pi_{soap} = (H, \mathbf{x}^I, c, O)$, we define a SAS+ problem $\Pi_{sas} = (\mathcal{X}, O, s_I, S_G)$ as follows:

- $\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$ is a set of state variables. Each variable x_i has a finite domain $Dom(x_i) = n_i$ where n_i is the number of partitions of the i-th feature of $\mathbf{x}$.
- O is a set of SAS+ actions directly mapped from O in Π_{soap}.
- s_I is transformed from $\mathbf{x}^I$ according to Definition 9.
- Let $(s_1, s_2, \ldots, s_K)$ be the K nearest neighbors of s_I ranked by $sim(s, s_j)$, and their corresponding preferred goal states be $(s^*_1, s^*_2, \ldots, s^*_K)$, the goal in SAS+ is $S_G = \{s^*_1, s^*_2, \ldots, s^*_K\}$. $K > 0$ is a user-defined integer.

In Example 2, if we preprocessed three initial states $s_1 = (0, 0, 0)$, $s_2 = (0, 1, 0)$, $s_3 = (0, 1, 1)$, then three preferred goal states $s^*_1 = (0, 1, 2)$, $s^*_2 = (0, 1, 2)$, and $s^*_3 = (0, 1, 2)$ will be found in the offline phase. In the online phase, given a new input $\mathbf{x}^I = (\text{male}, 2, 1200)$, the corresponding state is $s_I = (0, 0, 1)$. Suppose $\phi_i = 1/3$, then $sim(s_I, s_1) = 5/6$, $sim(s_I, s_2) = 1/2$, and $sim(s_I, s_3) = 2/3$. If $K = 2$, the 2 nearest neighbors of s_I are s_1 and s_3, and the goal of the SAS+ problem is $S_G = \{s^*_1, s^*_3\}$.

In the online phase, for a given $\mathbf{x}^I$, we solve a SAS+ instance defined above. In addition to classical SAS+ planning, we also want to minimize the total action costs. Since some existing classical planners do not perform well in optimizing the plan quality, we employ a SAT-based method.

Our method follows the bounded SAT solving strategy, originally proposed in SATPlan [30] and Graphplan [31]. It starts from a lower bound of makespan (L=1), encodes the SAS+ problem as a weighted partial Max-SAT (WPMax-SAT) instance [32], and either proves it unsatisfiable or finds a plan while trying to minimize total action costs at the same time.

For a SAS+ problem $\Pi_{sas} = (\mathcal{X}, \mathcal{O}, s_I, S_G)$, given a makespan L, we define a WPMax-SAT problem Ψ with the following variable set U and clause set C. The variable set includes three types of variables:

- Transition variables: $U_{\delta,t}$, $\forall \delta \in \mathcal{T}$ and $t \in [1, L]$.
- Action variables: $U_{a,t}$, $\forall a \in \mathcal{O}$ and $t \in [1, L]$.
- Goal variables: U_{s^*}, $\forall s^* \in S_G$.

Each variable in U represents the assignment of a transition or an action at time t, or a goal condition s^*.

The clause set C has two types of clauses: soft clauses and hard clauses. The soft clause set C^s is constructed as: $C^s = \{\neg U_{a,t} | \forall a \in \mathcal{O} \text{ and } t \in [1, L]\}$. For each clause $c = \neg U_{a,t} \in C^s$, its weight is defined as $w(c) = \pi(a)$. For each clause in the hard clause set C^h, its weight is $\sum_{c \in C^s} w(c)$ so that it must be true. C^h has the following hard clauses:

- Initial state: $\forall x, s_I(x) = f$, $\bigvee_{\forall \delta^x_{f \to g} \in \mathcal{T}(x)} U_{\delta^x_{f \to g}, 1}$.
- Goal state: $\bigvee_{\forall s* \in S_G} U_{s^*}$. It means at leat one goal condition s^* must be true.
- Goal condition: $\forall s^* \in S_G$, $\forall x, s^*(x) = g$, $U_{s^*} \to \bigvee_{\forall \delta^x_{f \to g} \in \mathcal{T}(x)} U_{\delta^x_{f \to g}, L}$. If U_{s^*} is true, then for each assignment $s^*(x) = g$, at least one transition changing variable x to value g must be true at time L.
- Progression: $\forall \delta^x_{f \to g} \in \mathcal{T}(x)$ and $t \in [1, L-1]$, $U_{\delta^x_{f \to g}, t} \to \bigvee_{\forall \delta^x_{g \to h} \in \mathcal{T}(x)} U_{\delta^x_{g \to h}, t+1}$.
- Regression: $\forall \delta^x_{f \to g} \in \mathcal{T}(x)$ and $t \in [2, L]$, $U_{\delta^x_{f \to g}, t} \to \bigvee_{\forall \delta^x_{h \to f} \in \mathcal{T}(x)} U_{\delta^x_{h \to f}, t+1}$.
- Mutually exclusive transitions: for each mutually exclusive transitions pair (δ_1, δ_2), $t \in [1, L]$, $\overline{U_{\delta_1,t}} \bigvee \overline{U_{\delta_2,t}}$.
- Mutually exclusive actions: for each mutually exclusive actions pair (a_1, a_2), $t \in [1, L]$, $\overline{U_{a_1,t}} \bigvee \overline{U_{a_2,t}}$.
- Composition of actions: $\forall a \in \mathcal{O}$ and $t \in [1, L-1]$, $U_{a,t} \to \bigwedge_{\forall \delta \in M(a)} U_{\delta,t}$.
- Action existence: for each non-prevailing transition $\delta \in \mathcal{T}$, $U_{\delta,t} \to \bigvee_{\forall a, \delta \in M(a)} U_{a,t}$.

There are three main differences between our approach and a related work, SASE (SAS+ based Encoding) [33,34]. First, our encoding transforms the SAS+ problem to a WPMax-SAT problem aiming at finding a plan with minimal total action costs while SASE transforms it to a SAT problem which only tries to find a satisfiable plan. Second, besides transition and action variables, our encoding has extra goal variables since the goal definition of our SAS+ problem is a combination of several goal states while in SASE it is a partial assignment of some variables. Third, the goal clauses of our encoding contain two kinds of clauses while SASE has only one since the goal definition of ours is more complicated than SASE.

We can solve the above encoding using any of the MaxSAT solvers, which are extensively studied. Using soft clauses to optimize the plan in our WPMax-SAT encoding is similar to Balyo's work [35] which uses a MAXSAT based approach for plan optimization (removing redundant actions).

5 Related work

Knowledge extraction has been studied in marketing science. Most of them are focused on extracting certain rules [8, 9], models [10], or ranking informations [11, 13, 14]. Research on actionable knowledge discovery is still very limited in the machine learning community. Early work includes using similarity analysis to prune and summarize the learnt rules [4, 5], domain-driven mining [6, 7], and combining meta-synthetic ubiquitous intelligence and several types of other frameworks into the mining process [7].

Some other related works are post-analysis techniques proposed to extract actionable knowledge on decision tree and additive tree models [15–17]. Yang et al. focus on finding optimal strategies by using a greedy strategy to search on one or multiple decision trees [15, 16]. Cui et al. use an integer linear programming (ILP) method to find actions changing sample membership on an ensemble of trees [17]. However, both only consider actions changing one attribute each time. Our SAS+ formulation is more general as it can naturally model actions changing multiple attributes using SAS+ actions. Another work supporting action changing multipel attributes is a sub-optimal state space search method which is very close to our offline preprocessing algorithm [22]. The main difference is that we use a much larger termination parameter Δ^u. By combining offline state space search and online fast SAS+ planning, our algorithm achieves a much higher search efficiency than [22] while mainting a better plan qualities.

6 Experimental results

To test the proposed approach (denoted as "Planning"), in the

offline preprocess, Δ in the heuristic search algorithm is set to 10^7. In the online search, we set neighborhood size $K = 3$ and use WPM-2014-in to solve the encoded WPMax-SAT instances. For comparison, we also implement three solvers: 1) An iterative greedy algorithm, denoted as "Greedy" which chooses one action in each iteration that increases $p(y = c|s)$ while minimizes the total action costs. It keeps iterating until there is no more variables to change. 2) A sub-optimal state space method denoted as "NS" [22]. 3) An integer linear programming (ILP) method [17], one of the state-of-the-art algorithms for solving the SOAP problem. ILP gives exact optimal solutions.

We test these algorithms on a real-world credit card company dataset ("Credit") and other nine benchmark datasets from the UCI repository and the LibSVM website used in ILP's original experiments [17]. Information of the datasets is listed in Table 1. N, D, and C are the number of instances, features, and classes, respectively. A random forest is built on the training set using the Random Trees library in OpenCV 2.4.9. GNU C++ 4.8.4 and Python 2.7 run-time systems are used.

Table 1 Datasets information and offline preprocess results

Dataset	N	D	C	T/s	#S	$\sum$T/d
Credit	17714	14	2	1.22	3.40E+09	4.81E+01
A1a	32561	123	2	365.25	1.68E+07	7.09E+01
Australian	690	14	2	0.06	1.14E+08	7.34E-02
Breast	683	10	2	2.43	7.07E+07	1.99E+00
Dna scale	2000	180	3	161.89	3.36E+07	6.29E+01
Heart	270	13	2	0.35	2.07E+08	8.37E-01
Ionosphere scale	351	34	2	64.06	8.39E+06	6.22E+00
Liver disorders	345	6	2	0.05	2.33E+05	1.40E-04
Mushrooms	8124	112	2	0.01	2.05E+03	1.80E-07
Vowel	990	10	11	0.15	5.96E+08	1.06E+00

In the offline preprocess, we generate all possible initial states and use the heuristic search algorithm to find a preferred goal state for each initial state. For each dataset, we generate problems with the same parameter settings as in ILP experiments. Specifically, we use a weighted Euclidean distance as the action cost function. For action a which changes state $\mathbf{s} = (x_1, x_2, \ldots, x_M)$ to $\mathbf{s}' = (x'_1, x'_2, \ldots, x'_M)$, the cost is

$$\pi(a) = \sum_{j=1}^{M} \beta_j (x_j - x'_j)^2, \tag{6}$$

where β_j is the cost weight on variable j, randomly generated in $[1, 100]$. Since the offline preprocess works are totally independent, we can parallelly solve them in a large number of workstation nodes. We run the offline preprocess parallelly on a workstation with 125 computational nodes. Each node has a 2.50GHz processor with 8 cores and 64GB memory. For each instance, the time limit is set to 1800 seconds. If the preprocess search does not finish in 1800 seconds, we record the best solution found in terms of net profit and the total search time (1800 seconds).

We show the average preprocessing time (T) on each dataset in seconds and the total number of possible initial states (#S) in Table 1. $\sum$T shows how many days it costs to finish all preprocess works by parallelly solving in 1000 cores. We can see that even though the total number of preprocessed states are very large, the total preprocess time can be extensively reduced to an acceptable range by parallelly solving.

In the offline preprocess, the percentage of actual preprocessed states out of all possible initial states in the transformed state space is a key feature of determing the online search quality. For each preprocessing percentage $r \in (0, 100]$, we randomly sample $r * \#S$ instances from all possible initial states and use the heuristic search algorithm to find preferred goals. Then, in the online search, we randomly sample 100 instances from the test set and generate 100 problems based on these preferred goals. We report the online search time in seconds and total action costs of the solutions, averaged over 100 runs. From Figs. 2(a) and 2(c), we can see that the total offline preprocessing time linearly increases with the percentage and the average total action costs almost linearly decrease with the percentage. Actually, considering the almost unlimited offline preprocessing time, we can always increase the preprocessing percentage and eventually reach 100%.

Table 2 shows a comprehensive comparison in terms of the average search time, the solution quality measured by the total action costs, the action number of solutions, and the memory usage under the preprocessing percentage 100%. We report the search time (T) in seconds, total action costs of the solutions (Cost), action number of solutions (L), and the memory usage (GB), averaged over 100 runs.

From Table 2 and Fig. 2(b), we can see that even though our method spends quite a lot of time on offline processing, its online search is very fast. Since our method finds near optimal plans for all training samples, its solution quality is much better than Greedy while spending almost the same search time. Comparing against NP, our method is much faster in online search and maintains better solution qualities in a1a and ionosphere scale and equal solution qualities in other eight datasets. Comparing against ILP, our method is much faster in online search with the cost of losing optimality. Typically a trained random forest model will be used for long time. Since

Table 2 Comparison of four SOAP algorithms on ten datasets. ILP is optimal and others are suboptimal

Dataset	Greedy				NS				Planning				ILP			
	T /s	Cost	L	M /GB	T /s	Cost	L	M /GB	T /s	Cost	L	M /GB	T /s	Cost	L	M /GB
Credit	1.06	525.61	12.07	0.01	1.65	33.20	3.37	0.05	**0.08**	**33.20**	**3.17**	**15.21**	6.59	33.20	3.37	1.13
A1a	1.24	462.07	8.47	0.01	6.56	68.07	3.10	0.11	**0.05**	**62.17**	**3.40**	**3.85**	7.56	60.60	3.33	1.10
Australian	0.04	215.10	9.30	0.01	0.06	6.03	1.37	0.01	**0.03**	**6.03**	**1.37**	**2.98**	108.89	6.03	1.37	2.12
Breast	0.02	375.70	16.77	0.01	0.65	74.97	11.70	0.01	**0.11**	**74.97**	**11.70**	**1.20**	30.58	74.97	11.70	1.45
Dna scale	0.11	775.26	16.68	0.01	4.59	75.30	3.00	0.08	**0.05**	**75.30**	**3.00**	**11.26**	34.54	75.30	3.00	1.31
Heart	0.02	569.07	9.13	0.01	0.05	83.37	2.03	0.01	**0.04**	**83.37**	**2.03**	**5.03**	5.54	83.37	2.03	1.07
Ionosphere scale	0.04	1219.12	25.62	0.01	62.33	460.33	12.23	0.52	**0.10**	**445.40**	**12.17**	**0.54**	47.97	444.90	12.17	1.80
Liver disorders	0.04	212.67	4.90	0.01	0.07	83.17	2.50	0.01	**0.04**	**83.17**	**2.50**	**0.01**	30.47	83.17	2.50	1.52
Mushrooms	0.00	58.71	1.00	0.01	0.01	30.27	1.13	0.01	**0.03**	**30.27**	**1.13**	**0.01**	3.74	30.27	1.13	1.07
Vowel	0.02	425.29	9.83	0.01	0.49	61.63	4.20	0.01	**0.06**	**61.63**	**4.20**	**11.11**	66.92	61.63	4.20	1.82

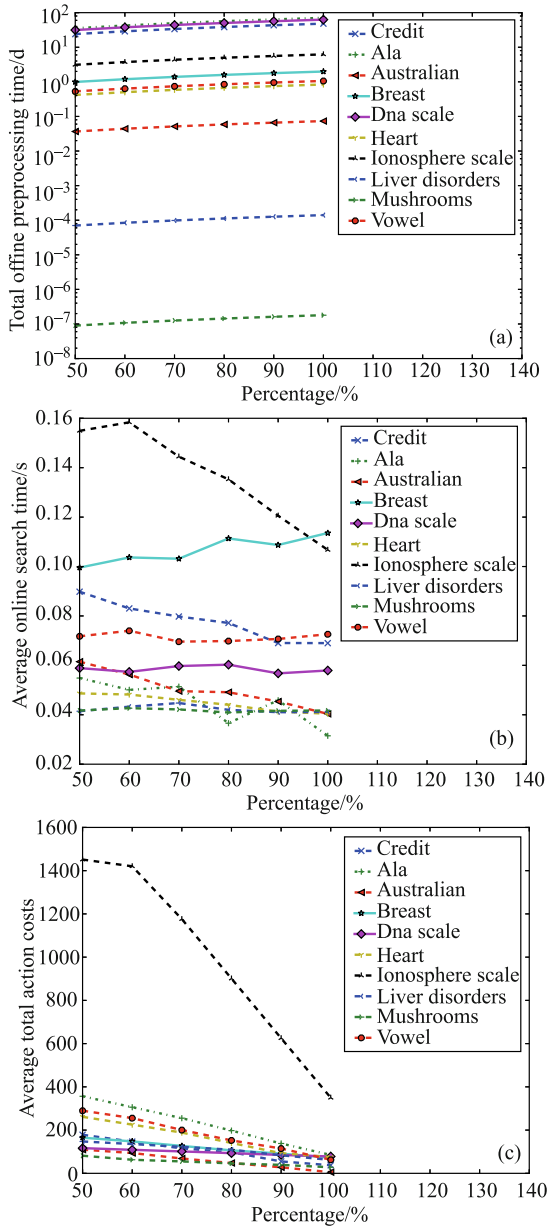


Fig. 2 Experimental results of offline preprocess for different preprocessing percentages. (a) Total offline preprocess time; (b) average online searching time; (c) average total action costs

our offline preprocessing only needs to be run once, its cost is well amortized over large number of repeated uses of the online search. In short, our planning approach gives a good quality-efficiency tradeoff: it achieves a near-optimal quality using search time close to greedy search. Note that since we need to store all preprocessed states and their preferred goal states in the online phase, the memory usage of our method is much larger than greedy and NS approaches.

7 Conclusions

We have studied the problem of extracting actionable knowledge from random forest, one of the most widely used and best off-the-shelf classifiers. We have formulated the suboptimal actionable plan (SOAP) problem, which aims to find an action sequence that can change an input instance's prediction label to a desired one with the minimum total action costs. We have then proposed a SAS+ planning approach to solve the SOAP problem. In an offline phase, we construct an action graph and identify a preferred goal for each input instance in the training dataset. In the online planning phase, for each given input, we formulate the SOAP problem as a SAS+ planning instance based on a nearest neighborhood search on the preferred goals, encode the SAS+ problem to a WPMax-SAT instance, and solve it by calling a WPMax-SAT solver.

Our approach is heuristic and suboptimal, but we have leveraged SAS+ planning and carefully engineered the system so that it gives good performance. Empirical results on a credit card company dateset and other nine benchmarks have shown that our algorithm achieves a near-optimal solution quality and is ultra-efficient, representing a much better quality-efficiency tradeoff than some other methods.

With the great advancements in data science, an ultimate goal of extracting patterns from data is to facilitate decision making. We envision that machine learning models will be

part of larger AI systems that make rational decisions. The support for actionability by these models will be crucial. Our work represents a novel and deep integration of machine learning and planning, two core areas of AI. We believe that such integration will have broad impacts in the future.

In our SOAP formulation, we only consider actions having deterministic effects. However, in many realistic applications, we may have to tackle some nondeterministic actions. For instance, pushing a promotional coupon may only have a certain probability to increase the accumulation effect since people do not always accept the coupon. We will consider to add nondeterministic actions to our model in the near future.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61502412, 61379066, and 61402395), Natural Science Foundation of the Jiangsu Province (BK20150459, BK20151314, and BK20140492), Natural Science Foundation of the Jiangsu Higher Education Institutions (15KJB520036), United States NSF grants (IIS-0534699, IIS-0713109, CNS-1017701), Microsoft Research New Faculty Fellowship, and the Research Innovation Program for Graduate Student in Jiangsu Province (KYLX16_1390).

References

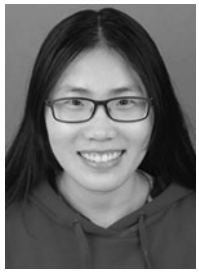
1. Mitchell T M. Machine learning and data mining. Communications of the ACM, 1999, 42(11): 30–36
2. Bailey T C, Chen Y X, Mao Y, Lu C Y, Hackmann G, Micek S T, Heard K M, Faulkner K M, Kollef M H. A trial of a real-time alert for clinical deterioration in patients hospitalized on general medical wards. Journal of Hospital Medicine, 2013, 8: 236–242
3. Johnson R A, Gong R, Greatorex-Voith S, Anand A, Fritzler A. A data-driven framework for identifying high school students at risk of not graduating on time. Bloomberg Data for Good Exchange, 2015
4. Liu B, Hsu W. Post-analysis of learned rules. In: Proceedings of the AAAI Conference on Artificial Intelligence. 1996, 828–834
5. Liu B, Hsu W, Ma Y M. Pruning and summarizing the discovered associations. In: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1999, 125–134
6. Cao L B, Zhang C Q. Domain-driven, actionable knowledge discovery. IEEE Intelligent Systems, 2007, 22(4): 78–88
7. Cao L B, Zhao Y C, Zhang H F, Luo D, Zhang C Q, Park E K. Flexible frameworks for actionable knowledge discovery. IEEE Transactions on Knowledge and Data Engineering, 2010, 22(9): 1299–1312
8. DeSarbo W S, Ramaswamy V. Crisp: customer response based iterative segmentation procedures for response modeling in direct marketing. Journal of Direct Marketing, 1994, 8(3): 7–20
9. Levin N, Zahavi J. Segmentation analysis with managerial judgment. Journal of Direct Marketing, 1996, 10(3): 28–47
10. Moro S, Cortez P, Rita P. A data-driven approach to predict the success of bank telemarketing. Decision Support Systems, 2014, 62: 22–31
11. Hilderman R J, Hamilton H J. Applying objective interestingness measures in data mining systems. In: Proceedings of European Conference of Principles of Data Mining and Knowledge Discovery. 2000, 432–439
12. Cao L B, Luo D, Zhang C Q. Knowledge actionability: satisfying technical and business interestingness. International Journal of Business Intelligence and Data Mining, 2007, 2(4): 496–514
13. Cortez P, Embrechts M J. Using sensitivity analysis and visualization techniques to open black box data mining models. Information Sciences, 2013, 225: 1–17
14. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. In: Proceedings of the International Conference on Learning Representations. 2014
15. Yang Q, Yin J, Ling C, Chen T. Postprocessing decision trees to extract actionable knowledge. In: Proceedings of the 3rd IEEE International Conference on Data Mining. 2003, 685–688
16. Yang Q, Yin J, Ling C, Pan R. Extracting actionable knowledge from decision trees. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(1): 43–56
17. Cui Z C, Chen W L, He Y J, Chen Y X. Optimal action extraction for random forests and boosted trees. In: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2015, 179–188
18. Friedman J, Hastie T, Tibshirani R. The Elements of Statistical Learning, Vol 1. New York: Springer-Verlag, 2001
19. Shotton J, Sharp T, Kipman A, Fitzgibbon A, Finocchio M, Blake A, Cook M, Moore R. Real-time human pose recognition in parts from single depth images. Communications of the ACM, 2013, 56(1): 116–124
20. Viola P, Jones M J. Robust real-time face detection. International Journal of Computer Vision, 2004, 57(2): 137–154
21. Mohan A, Chen Z, Weinberger K. Web-search ranking with initialized gradient boosted regression trees. Journal of Machine Learning Research, 2011, 14: 77–89
22. Lu Q, Cui Z C, Chen Y X, Chen X P. Extracting optimal actionable plans from additive tree models. Frontiers of Computer Science, 2017, 11(1): 160–173
23. Freund Y, Schapire R E. A decision-theoretic generalization of online learning and an application to boosting. Journal of Computer and System Sciences, 1997, 55: 119–139
24. Friedman J H. Greedy function approximation: a gradient boosting machine. The Annals of Statistics, 2001, 29: 1189–1232
25. Breiman L. Random forests. Machine Learning, 2001, 45(1): 5–32
26. Fox M, Long D. PDDL2.1: an extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, 2003, 20: 61–124
27. Bäckström C, Nebel B. Complexity results for SAS+ planning. Computational Intelligence, 1995, 11(4): 625–655
28. Jonsson P, Bäckström C. State-variable planning under structural restrictions: algorithms and complexity. Artificial Intelligence, 1998, 100(1–2): 125–176
29. Helmert M. The fast downward planning system. Journal of Artificial Intelligence Research, 2006, 26: 191–246
30. Kautz H A, Selman B. Planning as satisfiability. In: Proceedings of European Conference on Artificial Intelligence. 1992, 359–363
31. Blum A, Furst M L. Fast planning through planning graph analysis. Artificial Intelligence, 1997, 90(1–2): 281–300
32. Lu Q, Huang R Y, Chen Y X, Xu Y, Zhang W X, Chen G L. A SAT-based approach to cost-sensitive temporally expressive planning. ACM Transactions on Intelligent Systems and Technology, 2014, 5(1): 18
33. Huang R Y, Chen Y X, Zhang W X. A novel transition based encoding scheme for planning as satisfiability. In: Proceedings of the AAAI

Conference on Artificial Intelligence. 2010, 89–94

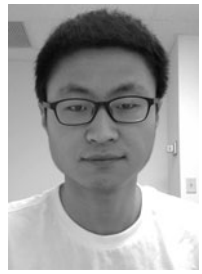
34. Huang R Y, Chen Y X, Zhang W X. SAS+ planning as satisfiability. Journal of Artificial Intelligence Research, 2012, 43: 293–328
35. Balyo T, Chrpa L, Kilani A. On different strategies for eliminating redundant actions from plans. In: Proceedings of the 7th Annual Symposium on Combinatorial Search. 2014, 10–18

Qiang Lv is currently an assistant professor in College of Information Engineering, Yangzhou University, China. He received the BE and PhD degrees from the School of Computer Science and Technology, University of Science and Technology of China, China in 2007 and 2012, respectively. His research interests include data mining, automated planning and scheduling. He has published more than ten papers in journals and conference proceedings, including ACM TIST, IEEE TSC, EAAI, FCS, AAAI'13, ICAPS'11, CloudCom'11, and IPC'11. He is a member of the ACM and the CCF.

Yixin Chen is a professor of computer science at the Washington University in St. Louis, USA. He received the PhD degree in computer science from the University of Illinois at Urbana-Champaign, USA in 2005. His research interests include nonlinear optimization, constrained search, planning and scheduling, data mining, and data warehousing. His work on planning has won First-Class Prizes in the International Planning Competitions (2004 and 2006), the Best Paper Award in AAAI (2010) and ICTAI (2005), and Best Paper nomination at KDD (2009). He has received an Early Career Principal Investigator Award from the Department of Energy (2006) and a Microsoft Research New Faculty Fellowship (2007). Dr. Chen is a senior member of IEEE. He serves as an associate editor of IEEE Transactions on Knowledge and Data Engineering, and ACM Transactions on Intelligent Systems and Technology.



Zhaorong Li is currently a graduate student in the College of Information Engineering, Yangzhou University (YZU), China. She received the Bachelor's degree from the College of Guangling at YZU. Her research interests include data mining, machine learning and artificial intelligence. She has published two papers in Journal of Chinese Computer Systems and CCDM. She is a student member of the CCF.



Zhicheng Cui received his BE degree in computer science from University of Science and Technology of China, China in 2014. He is now a PhD candidate in the Department of Computer Science and Engineering, Washington University in St. Louis (WUSTL), USA, supervised by Prof. Yixin Chen. His research interests are data mining and machine learning, in the area of large scale time series analysis.

Ling Chen is currently a professor in the College of Information Engineering, Yangzhou University, China. His research interests include bioinformatics, data mining and computational intelligence. He has co-edited six books/proceedings, and published more than 300 research papers including over 120 journal papers. He has received many awards from government and agencies. He has organized several academic conferences and workshops and has also served as a program committee chair or member for several major international conferences. He is a member of IEEE CS society and ACM, and a senior member of the Chinese Computer Society.

Xing Zhang is an assistant professor of marketing at the School of Management, Fudan University, China. She received the PhD degree in marketing from Washington University in St. Louis, USA in 2013. Her research interests are in empirical modeling consumer behavior and firm competition using econometric methodologies. She has conducted various research projects in the domain of marketing and economics. Her research about consumer information search and firm pricing has been published in Management Science.

Haihua Shen is an associate professor with the School of Computer and Control Engineering, University of Chinese Academy of Sciences, China. She received the PhD degree in computer science and technology from Tsinghua University, China in 2002. Her research interests include computer architecture, artificial intelligence, VLSI design & verification and hardware security. She has published more than 30 technical papers, and holds 20 patents.