Goal-Based Rule Learning
Technical Report, March 2009

Martin Možina, Matej Guid, Aleksander Sadikov, and Ivan Bratko

Faculty of Computer and Information Science, University of Ljubljana, Slovenia

1 Introduction

Goal-Based Rule Learning is used to derive a meaningful strategy in domains that require search and are usually solved by a computer better than by a human (*e.g.* 8-puzzle game, scheduling, game playing, etc.). A strategy is an ordered list of goals that lead to the solution of the problem, when they are achieved in exact order. These goals can then be used to teach a human, who is incapable of extensive searching, how to act in these domains and still be able to solve these problems simply by following suggested goals.

The most natural application of Goal-Based Rule Learning is in game playing, especially chess. Nowadays, computer programs can beat the world champion in chess mainly due to the capability of searching very deep. However, are there any goals, patterns, or concepts in the computer's play that humans are not aware of them yet? Would knowing these goals improve humans play? Another possible application is in problem solving that requires complex search. For example, in the classic task of finding a way out of a maze, the question would be: what is the best strategy that would, on average, quickly bring me to the exist of the maze without searching exhaustively.

We start with a definition of Goal-Based Rule Learning, and in the following section describe our Gabriel algorithm. In the fourth section, we shall extend the basic Gabriel algorithm with some of the ideas from argument-based machine learning (ABML), the new method is called AB-Gabriel. AB-Gabriel enables efficient elicitation of expert knowledge and uses this knowledge to improve the quality of induced strategies.

2 Goal-Based Rule Learning

Goal-based rules learning can be used to extract strategies in specific type of problems that can be represented with a graph of states, often called state space. A particular problem is defined with a start state S_0 and a goal condition. The task is to find a plan that brings you from the start state to a state where the goal is achieved. A solution path is a list of states $\{S_0, \ldots, S_n\}$, where the goal must be achieved in S_n and transition between succeeding states is possible by a single move.

Sometimes searching exhaustively for a solution is too complex even for a computer. In such cases, we need a heuristic function h that can approximately order states by their "closeness" to the solution state. We shall assume that h assigns a value to each state and that lower values characterise states closer to the solution.

We define learning goal-based rules as:

- Given (some) solution paths and a heuristics h
- Induce an ordered set of rules of form:

$$\text{IF } \textit{preconditions} \text{ THEN } \textit{goal}$$

The complete learning data for machine learning is therefore a set of states, where each of these states occured in one of the provided solution paths. Each state is acting as a learning example and is described with a vector of attributes that correspond to some well-known domain concepts. For example, in chess, we often use attributes like *kdist* (distance between both kings) or *edist* (distance of black king from nearest edge).

The rule's *preconditions* and *goal* are both expressed in terms of these attributes. Term *preconditions* is a conjunction of simple conditions, in rule learning often referred as selectors, that specify the required value of an attribute, for example $kdist = 3$ or a threshold on an attribute value, e.g. $kdist > 3$. Similarly, a *goal* is a conjunction of subgoals, where a subgoal can specify the desired value of an attribute (e.g. $kdist = 3$), any of four possible qualitative changes of an attribute given the initial position: *decrease, increase, not decrease, not increase* or its optimisation: *minimise, maximise.* For example, a subgoal can be " decrease *kdist*" (decrease distance between kings). In case of a discrete attribute, a subgoal can only specify the target value of this attribute.

The resulting rules are defining a procedure for finding a solution in the selected problem. An example rule could be:

$$\text{IF } edist > 1 \text{ THEN } \text{ decrease } kdist$$

The rule can be interpreted as: "if black king's distance from the edge is larger than 1 and a decrease in distance between kings is possible, then reach this goal: decrease the distance between kings." Note that there is an important difference between semantics of goal-based rules and classical if-then rules. A normal if-then rule triggers if the preconditions are true, while a goal-based rule triggers only if the goal is actually achievable (exact definition of when a goal is achievable is described in the following section). In other words, even if all preconditions are true for a position, it is not necessary that this rule will *cover* the position. The new redefined *cover* relation is: a goal-based rule R covers a state s if:

- the preconditions of rule R are true for s, and
- goal of R is achievable in s.

2.1 Hierarchy of Goals and Achieving a Goal

In the definition of *covering* relation, we mentioned that a goal must be achievable. We say that a goal G is achievable in a selected example, if we can execute the goal G or any of the relevant goals above G in the rule set, where a goal is relevant if its rule preconditions are true for the particular learning example. Such a hierarchical structure assumes that attaining a goal higher on hierarchy is always better than attaining a goal positioned lower. For example, in chess endings, if the goal is to push the defender's king towards the edge and the defender resists the goal by allowing the opponent to deliver checkmate (that would not be achievable without the opponent's help), player should be content by mating the opponent, as the mate goal is higher on hierarchy than decreasing edge distance.

In the basic type of search problems, where the task is to find a path from a start node to a goal node, the hierarchy is not relevant. I a higher goal can be achieved, the the current one will not even be considered. However, in problems where a disjunction between goals is required (*e.g.* AND/OR graphs or 2-player games, where minimax is used), the higher and the current goals by itself only might not be achievable, however they are, when applied together.

2.2 Evaluation of a Goal

If a goal is achievable, we would like to know how good it is in a given position. We evaluate the goal by its worst possible realization in terms of heuristic evaluation of the final state in the search tree. Formally, goal's quality $q(g, s)$ in state s is defined as the difference between starting heuristic evaluation and heuristic evaluation in the worst realization of the goal: $q(g, s) = h(s_{worst}) - h(s)$. We say that a goal is *good* for a state s if its worst realization reduces the distance to solution, i.e. if $q(g, p) < 0$; otherwise the goal is *bad*.

2.3 Evaluation of a Rule

The quality of a rule R is directly related to the quality of its goal on learning examples covered by the rule. Let p be the number of covered examples where the goal is *good* and n number of all covered examples. Then, the quality is computed using Laplacian formula of probability:

$$q(R) = \frac{p+1}{n+1} \tag{1}$$

2.4 Special Types of Heuristic Functions

In certain domains, we can search the complete state space and compute for each state its minimal distance to the goal state. For example, such "heuristics" is available for many chess endgames in the form of tablebases. Tablebases enable optimal play in the sense of shortest win against best defence. However, the resulting play using tablebases is difficult or occasionally impossible to understand by humans. Our approach can detect subgoals in the optimal play and hence explain the human optimal game with some basic concepts.

Algorithm 1 Pseudo code of the Gabriel goal-based rule learning method.

```
Procedure GABRIEL(Examples ES)

Let allRules be an empty list.
while $ES \neq \emptyset$ do
    Let seedState be FindBestSeed($ES, ruleList$).
    Let goals be DiscoverGoals($seedState, ES, ruleList$).
    Let rule be LearnRule($goals, seedState, ES, ruleList$).
    Add rule to allRules.
    Remove examples from ES covered by rule.
end while
Return allRules.
```

Another type of heuristics is derived from solution paths only. This is useful when devising a heuristic function is relatively hard, however a human can reach a solution quite easily. Here, the heuristic value of a state is defined as the number of moves that was needed to reach the goal state in the given solution path. If a state is not mentioned in any of the solution paths, then its heuristic value can simply be the maximum possible value. Here, the search mechanism for evaluating goals will always try to reach one of the "known" positions in solution paths and the resulting rules will describe what goals did the human have in mind while solving this problem.

3 The GABRIEL Algorithm

The pseudo code of our goal-based rule learning method Gabriel (a mnemonic for Goal-Based Rule Learner) is shown in Alg. 1. The learning loop starts by selecting a seed state, which is then used in the following calls to procedures *DiscoverGoals* and *LearnRule*. The *DiscoverGoals* procedure finds most favorable goals for the seed state and then *LearnRule* induces one rule for each possible goal and returns the most appropriate rule.[1] The idea of seed examples and learning rules from them was adopted from the AQ series of rule-learners developed by Michalski[4], and is especially useful here, since discovering a goal is a time consuming step. Learned rule is afterwards added to the list of all rules *allRules* and all examples covered by this rule are removed from the learning examples. The loop is stopped when all learning examples have been covered.

3.1 Procedure: FindBestSeed

As mentioned above, a goal is always evaluated together with goals that are higher on the hierarchy, therefore we desire to have goals sorted by importance; the most important goal should be the first in the hierarchy. We speculate that

---

[1] The selection of the most appropriate rule will be explained later within the explanation of *LearnRule* procedure.

important goals come into place in later stages of the solution plan, and hence we select as the best seed example the one with lowest heuristic value in the current examples.

3.2 Procedure: DiscoverGoals

Search for the best goal in a given position follows the well known star-search strategy, which is often used in separate-and-conquer rule learning algorithms [2]. It starts with an empty goal and sequentially adds subgoals until we find a *good* goal. A goal can have a maximum of five subgoals, and if the algorithm reaches this limit, while goal is still *bad*, then this procedure returns “goal not found”. If there are several good goals having the same number of subgoals, then the method returns all good goals.

3.3 Procedure: LearnRule

LearnRule procedure first creates for each provided goal a new data set containing all positions from ES, where this goal is achievable. Each position in the new data set is labeled as either *good* goal or as *bad* goal. If several goals were provided, the best goal is selected according to the following criteria:

$$i(g) = \frac{\texttt{num. of examples with good goal}}{[max(h) - h(seedState)]^2} \tag{2}$$

where max(h) is the maximum heuristic value of examples where goal is achievable. Afterwards, *LearnRule* procedure learns a rule for the best goal. Its preconditions separate states where a goal is *good* from those where it is not. We use CN2 [1] to learn one rule only, where its conditions must cover the $seedState$.

4 Argument-Based Extension of Gabriel Algorithm

Argument Based Machine Learning (ABML) [3] is machine learning extended with certain concepts from argumentation. Arguments in ABML are a way to enable domain experts to provide their prior knowledge about a specific learning example that seems relevant for this case and does not have to be valid for the whole domain. This approach greatly reduces the difficulty that experts face when they try to articulate their background knowledge.

An argument can be regarded as partial explanation of a single learning example. A learning example explained with an argument is called an argumented example. An ABML method is then required to induce a theory that uses given arguments to explain the argumented examples. Thus, arguments constrain the combinatorial search among possible hypotheses, and also direct the search towards hypotheses that are more comprehensible in the sense of expert’s background knowledge. When an ABML method is used without arguments, it acts as a normal machine learning method.

In goal-based rule learning, ABML is a favorable approach because of three reasons:

- Arguments will prevent learning rules describing some strange effects occurring only in the given training data set.
- Induced rules will suggest goals more in line with the experts's thinking.
- Experts find it much easier to provide goals for *particular* problem states than give exact rules applicable to the complete task.

The structure of an argument depends on the learning problem. In goal-based rule learning, an argument has the following structure: "Goal g is good in state s because *conditions*". For example, a concrete argument in a chess-ending could be: "In this position, a good goal is to decrease distance between kings, because black king is sufficiently constrained".

In ABML, the method should induce such a hypothesis that uses given arguments while explaining particular examples. A rule in goal-based rule learning explaining an argumented problem state should therefore also use its arguments, which is achieved by having a goal and conditions from the argument as a part of the final rule. This can be achieved by a redefinition of the rule *covering* relation. Let a learning example in AB goal-based rule learning be a triple: (*attributes*, *argGoal*, *argConditions*). Then, a rule R AB-covers an example if:

- the preconditions of rule R are true for the example,
- goal of R is achievable in state s,
- *argGoal* is a subset of rule R's goal, and
- *argConditions* is a subset of rule's preconditions.

Along to the new covering relation, in order to enable efficient learning with arguments, we implemented two other changes in the algorithm:

- If *seedState* has an argument, then *DiscoverGoal* starts with *argGoal* and not an empty goal.
- If *seedState* does not have an argument, then *DiscoverGoal* first tries to apply any of the goals from arguments given to other examples. If none of the goals is good, then it searches for a goal the same as original GABRIEL.
- If *seedState* has an argument, the *LearnRule* procedures uses ABCN2 [3] instead of classical CN2, where *argConditions* are used as reasons.

The ABCN2 method used in *LearnRule* procedure is an argument-based extension of the CN2 method [1] that learns a set of unordered probabilistic rules from argumented examples. In ABCN2, the theory (a set of rules) is said to explain the examples using given arguments, when there exists at least one rule for each argumented example that contains at least one positive argument in the condition part. Since the method here learns only a single rule at a time, the requirement is that the final rule needs to contain *argConditions* in its preconditions.

5 ABML Refinement Loop: Selection of Critical Examples

In argument-based approach, experts provide their prior knowledge in the form of arguments for some of the learning examples. Asking them to give arguments to the whole learning set is not likely to be feasible, because it would require too much time and effort. The following loop describes the skeleton of the procedure that picks out critical learning examples that should, when explained, the most improve the quality of learned rules:

1. Learn a set of rules with GABRIEL.
2. Find the most critical state and present it to the expert.
3. Expert explains the example; the explanation is encoded in arguments and attached to the learning example.
4. Return to step 1 if critical position was found.

With respect to the second step in the loop, there are two types of critical states: *type A* states are critical states, where rules do not suggest any goals (no rules have triggered) and *type B* are states, where rules suggest a *bad* goal.

5.1 Explaining Type A Critical States

The explanation of a type A critical state contains the following five steps:

Step 1: Explaining critical state. Expert provides an argument "Goal because Reasons". This argument is then added to the critical state.

Step 2: Evaluating goal in argument. Method first classifies the goal in the argument into *bad*, *good*, or *not-achievable.*

- If goal is *not-achievable*, expert is asked to change the argument. Return to Step 1.
- If goal is *good*, we proceed to Step 3.
- If goal is *bad*, then the method shows to the expert the critical sequence of moves that achieves the goal and at the same time increases heuristic value. The expert can accept the sequence of moves as *good*, although heuristic value is increased (continue to Step 3), or improves the initial goal and returns to Step 1.

Step 3: Discovering counter states. GABRIEL learns a rule " IF *cond* THEN $Goal = g$", where the critical state is used as a *seedState*. *Counter states* are positions covered by the rule, where the learned goal is classified as *bad.* If there are no such positions, we move to Step 4. Otherwise, the counter state s with highest $q(g, s)$ (see evaluation of goal section) is denoted as a *key counter state.* This state is presented to the expert.

Step 4: Improving argument. If a key counter example is found, the expert has three possible solutions:

- If he or she thinks that the goal in the key counter state is admissible, then we automatically accept also all other counter examples and move to Step 5.
- If the goal in the key counter position is not acceptable, the expert can improve the initial argument. Return to Step 2.
- If the goal in the key counter state is not acceptable, but the expert cannot improve the initial argument, then we provide the next counter example to the expert.

Step 5: Validating learned rule. We show the learned rule to the expert. If the rule seems good, we conclude the explanation process of the example. If there is a condition in preconditions of the rule that expert finds irrelevant or even wrong, then we can prevent adding this condition to the preconditions of the rule, when this example is used as a *seedState*. Return to Step 2.

5.2 Explaining Type B Critical States

In a Type B critical state the rules suggest a *bad* goal. Here, the expert is first asked whether the current goal is acceptable or should it be changed. If the current goal is acceptable, we continue to the next critical state, otherwise the critical state is explained in the same way as a Type A critical state.

References

1. Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In *Machine Learning - Proceeding of the Fifth Europen Conference (EWSL-91)*, pages 151–163, Berlin, 1991.
2. Johannes Fürnkranz and Peter A. Flach. Roc 'n' rule learning – towards a better understanding of covering algorithms. *Machine Learning*, 58(1):39–77, January 2005.
3. Martin Možina, Jure Žabkar, and Ivan Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10/15):922–937, 2007.
4. J. Wnek and R. S. Michalski. Hypothesis-driven constructive induction in aq17-hci: A method and experiments. *Machine Learning*, 14(2):139–168, 1994.