Goal-Oriented Conceptualization of Procedural Knowledge

Martin Možina, Matej Guid, Aleksander Sadikov, Vida Groznik, Ivan Bratko

Faculty of Computer and Information Science, University of Ljubljana, Slovenia
{martin.mozina,matej.guid,aleksander.sadikov,vida.groznik,ivan.bratko}
@fri.uni-lj.si

Abstract. Conceptualizing procedural knowledge is one of the most challenging tasks of building systems for intelligent tutoring. We present an algorithm that enables teachers to accomplish this task semi automatically. We used the algorithm on a difficult king, bishop, and knight versus the lone king (KBNK) chess endgame, and obtained concepts that could serve as textbook instructions. A pilot experiment with students and a separate evaluation of the instructions by experienced chess trainers were deemed very positive.

Keywords: domain conceptualization, procedural knowledge, goal-oriented rule learning, argument-based machine learning, chess

1 Introduction

Domain conceptualization lies at the very core of building an intelligent tutoring system (ITS) [7],[10]. This involves the structuring of the domain and creating a vocabulary or ontology of key concepts. Domain conceptualization consists of declarative knowledge and procedural knowledge, which generally speaking is the knowledge exercised in the performance at some task. Procedural knowledge is usually implicit and not easily articulated by the individual. Due to its tacit nature this kind of knowledge is often very hard to conceptualize.

In this paper, we will consider symbolic problem solving domains where problem solving is based on reasoning with symbolic descriptions (like in physics, mathematics, or games like chess). A particular domain is defined with a basic domain theory (*e.g.*, the rules of chess) and a solution to be achieved (*e.g.*, checkmate the opponent in chess). The task is to find a sequence of steps that bring us from the starting state of the problem to the goal state.

The basic domain theory (or basic declarative knowledge of the domain) is usually simple and easy to remember. It is, in principle, sufficient for solving problems (*e.g.*, knowing rules of chess could in theory enable optimal play). However, finding a solution using only declarative knowledge would require far too extensive searching for a human. A human student is incapable of searching very deeply, therefore we need to teach him also the procedural knowledge – how to solve problems. The "complete" procedural knowledge would be a function

mapping from each problem state to a corresponding action that leads to the solution of the problem. In chess, such complete knowledge (called "tablebases") is computed for some endgames. Tablebases effectively specify best moves for all possible positions. They logically follow from the rules of the game and can be viewed as a compilation of the rules into an extensive form. Tablebases can be used easily because they only require trivial amount of search. But now the problem is the space complexity – it is impossible for humans to memorize such tablebases that typically contain millions of positions.

There is a way, however, that enables humans to solve problems in such chess endgames quite comfortably. The key is that humans use some intermediate representation of the problem that lies between the rules of the game (or the corresponding tablebases) and solutions. We call such an intermediate representation a "conceptualized domain." Powerful conceptualizations are sufficiently "small" so they can be memorized by a human, and they contain concepts that enable fast derivation of solutions. Such a domain conceptualization enables effective reasoning about problems and solutions [8].

In this paper, we propose a goal-oriented conceptualization of domains and explore how to semi-automatically construct such a conceptualization that can be effectively used in teaching problem-solving. To this end, we used argument-based machine learning (ABML) [6], an approach that combines learning from examples with learning from domain knowledge. Such a combination can be particularly useful in the problem of domain conceptualization, as it is consistent with data (accurate) and at the same time consistent with expert's knowledge (understandable) [4]. A similar idea, however with a different goal, was explored in a system called *SimStudent* [2], where learning from examples and learning by tutored problem solving was interweaved. Another interesting and somewhat similar work comes from Tecuci et al. [9] who developed a series of systems called *Disciple* that combine different types of learning, such as learning from explanations provided by users or by generalizing learning examples.

2 Goal-Oriented Rules

A goal-oriented rule has the following structure:

IF preconditions THEN goal (depth)

The rule's preconditions and goal are expressed in terms of attributes used for describing states. The *preconditions* is a conjunction of simple conditions specifying the required value of an attribute. For example, preconditions could contain $kdist = 3$ (kdist being distance between kings in chess), 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.*, $kdist = 3$) or any of the four possible qualitative changes of an attribute given the initial value: decrease, increase, not decrease, not increase or its optimization: minimize, maximize; *e.g.*, a subgoal can be "decrease kdist" (decrease distance between kings). The depth property of a rule specifies the maximum allowed number of steps in

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

```
GOAL-ORIENTED RULE LEARNING (examples ES, depth)
let allRules be an empty list
while ES is not empty do
   let seedExample be FindBestSeed(ES, ruleList)
   let goals be DiscoverGoals(ES, seedExample, ruleList, depth)
   if goals is empty then
      remove seedExample from ES and return to the beginning of while sentence
   end if
   let rule be LearnRule(ES, goals, ruleList)
   add rule to allRules
   remove examples from ES covered by rule
end while
return allRules
```

achieving the goal. It corresponds to the level of conceptualization, where higher depths lead to simpler rules with less conditions and less subgoals, however, these goals are more difficult to solve, because they require more search.

The complete proposed conceptualization of procedural knowledge is a decision list of ordered goal-oriented rules. In an ordered set of rules, the first rule that "triggers" is applied. Note the difference between goal-oriented rules and classical if-then rules. An if-then rule triggers for a particular state if the preconditions are true, while a goal-based rule triggers when the preconditions are true *and* the goal is achievable. For example, consider a rule: *IF edist* > 1 *THEN decrease kdist*. The correct interpretation of this rule is: "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 the kings*."

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

The quality of a rule R is directly related to the quality of its goal on states 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 the Laplacian rule of succession: $q(R) = (p + 1)/(n + 2)$.

3 Goal-Oriented Rule Learning Algorithm

The task of learning goal-oriented rules is stated as: given a set of problem solving states each labeled with a distance-to-solution, learn an ordered set of goal-oriented rules. As these states act as learning examples, we will use this term in the description of the algorithm. As mentioned above, each learning example is described with a set of attributes.

The pseudo code of our goal-oriented rule learning method is shown in Algorithm 1. It accepts two parameters; ES are the learning examples and *depth* is the maximum allowed search depth for achieving goals.

The learning loop starts by selecting a seed example, which is used in the following calls to procedures $DiscoverGoals$ and $LearnRule$. The $DiscoverGoals$ procedure finds *good* goals for the seed example and then $LearnRule$ induces a rule covering this example. The idea of seed examples and learning rules from them was adopted from the AQ series of rule-learners developed by Michalski[3], and is especially useful here, since discovering a goal is a time consuming step. A 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.

The $FindBestSeed$ procedure selects as the seed example the one with the lowest distance-to-solution. The $DiscoverGoals$ procedure searches for best goals in a given example. It starts with an empty goal and iteratively adds subgoals (selecting from all possible subgoals, see section 2) until we find a *good* goal. If there are several *good* goals having the same number of subgoals, then the method returns all *good* goals. The $LearnRule$ procedure creates for each provided goal a data set containing all examples from ES, where this goal is achievable. Each example in the new data set is labeled as either a *good* goal or as a *bad* goal. Afterwards, $LearnRule$ procedure learns a single rule from each data set and selects the best among them. We use the CN2 algorithm to learn a rule.

We extended the above algorithm with the capability to use *arguments* as in argument-based machine learning (ABML)[6]. Arguments are provided by an expert to explain single learning examples – we call such examples *argumented examples*. The task in ABML is to find a hypothesis that is consistent with learning examples and arguments. In goal-oriented rule learning, an argument has the following structure: "$argGoal$ because $argConditions$," where an expert expresses his or her opinion that the goal $argGoal$ is *good* in the selected state, because the conditions $argConditions$ hold.

We developed an iterative loop that asks the expert to explain only critical examples, *i.e.*, examples not covered by any sufficiently good rules. Such loop significantly decreased the required effort of the expert; he needed to explain only a few examples instead of all. Due to space limitations, we only presented an overview of the ABML extension (see [1] and [5] for more details).

4 Evaluation

We used our algorithm for the conceptualization of procedural knowledge required to deliver checkmate in the KBNK chess endgame. KBNK (king, bishop, and knight vs. a lone king) is regarded as the most difficult of the elementary chess endgames. The stronger side can always checkmate the opponent, but even optimal play may take as many as 33 moves. There are many recorded cases when strong players, including grandmasters, failed to win this endgame. In an interactive procedure between a chess teacher (a FIDE master of chess) and the

computer, we derived instructions in the form of goals for delivering checkmate from any given KBNK position (see [1] for details). The result of this procedure was an ordered set of eleven rules.

The rules were used to compile *teaching materials* for playing KBNK: textbook instructions, supplemented with five example games.[1] They were presented to three chess teachers (among them a selector of Slovenian women's squad and a selector of Slovenian youth squad) to evaluate their appropriateness for teaching chess-players. They all agreed on the usefulness of the presented concepts and found the teaching materials suitable for educational purposes. Among the reasons to support this assessment was that the instructions "clearly demonstrate the intermediate subgoals of delivering checkmate." [1]

We further assessed the teaching materials with the following pilot experiment with three students – chess beginners of slightly different levels – who played several KBNK games against a computer. The computer was defending "optimally," *i.e.*, randomly choosing among moves with the longest distance to mate (using chess tablebases). The time limit was 10 minutes per game. Each game started from a different starting position, all mate-in-30-moves or more. The moves and times spent for each move were recorded automatically.

At the beginning of the experiment, each student played three games against the computer, and they always failed to deliver checkmate. They clearly lacked procedural knowledge for successfully delivering checkmate in this endgame before seeing the teaching materials.

Next, the students were presented with the teaching materials. They were reading the instructions and observing the example games until they felt they are ready to challenge the computer once again. None of them spent more than 30 minutes at this second stage.

In the final stage of the experiment, the students were again trying to checkmate the optimally defending computer. The textbook instructions and example games were not accessible to the students during the games. Only if a game ended in a draw, the student was again granted the access to the teaching materials for up to ten minutes before starting a new game. While the first student (a slightly stronger chess player than the other two) successfully checkmated in the second game already, the other two students checkmated in games 5 and 6, respectively. Once they achieved the win the students had no problems at all achieving it again, even with the white bishop being placed on the opposite square color than in all previous games.

Although the goal of the conceptualized procedural knowledge included in the textbook instructions is not to teach students how to play "optimally," but merely to enable them to achieve a step-by-step progress towards delivering checkmate, it is particularly interesting that the second student in his third game of the final stage of the experiment played 22(!) optimal moves in a row – an achievement that a chess grandmaster could be proud of. Moreover, it happened in less than an hour after he was first given access to the textbook instructions and example games. This result would be very hard or even impos-

[1] The teaching materials are available at `http://www.ailab.si/matej/KBNK`.

sible to achieve without an effective way of memorizing particular concepts of procedural knowledge required in order to master this difficult endgame.

5 Conclusions

We presented a novel algorithm for semi-automated conceptualization of procedural knowledge based on goal-oriented rules in symbolic domains. We applied the algorithm to the challenging KBNK chess endgame, and carried out a pilot experiment to evaluate whether the obtained concepts (instructions) could serve as a teaching tool. Somewhat surprisingly, even the beginner-level chess players were able to quickly grasp the concepts, and learn to deliver checkmate. A separate, subjective evaluation of the instructions by experienced chess trainers was also positive.

A more rigorous evaluation is an obvious task for further work. Apart from other domains, it should be evaluated whether the derived concepts could serve as the knowledge base of an ITS. To this end we plan to build such a system, and conduct an experiment on a much larger number of students.

References

1. Guid, M., Možina, M., Sadikov, A., Bratko, I.: Deriving concepts and strategies from chess tablebases. In: ACG. pp. 195–207 (2009)
2. Matsuda, N., Keiser, V., Raizada, R., Tu, A., Stylianides, G., Cohen, W., Koedinger, K.: Learning by teaching simstudent: Technical accomplishments and an initial use with students. In: Aleven, V., Kay, J., Mostow, J. (eds.) Intelligent Tutoring Systems, pp. 317–326. Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2010)
3. Michalski, R.S.: A theory and methodology of inductive learning. Artificial Intelligence 20(2), 111–161 (1983)
4. Možina, M., Guid, M., Krivec, J., Sadikov, A., Bratko, I.: Fighting knowledge acquisition bottleneck with argument based machine learning. In: Patras, G. (ed.) Proceedings of 18th European Conference on Artificial Intelligence (ECAI 2008). pp. 234–238. IOS Press, Patras, Greece (2008)
5. Možina, M., Guid, M., Sadikov, A., Groznik, V., Krivec, J., Bratko, I.: Conceptualizing procedural knowledge targeted at students with different skill levels, http://www.ailab.si/martin/abml/gorules.pdf, 2010, unpublished
6. Možina, M., Žabkar, J., Bratko, I.: Argument based machine learning. Artificial Intelligence 171(10/15), 922–937 (2007)
7. Murray, T.: Authoring intelligent tutoring systems: An analysis of the state of the art. International Journal of Artificial Intelligence in Education (IJAIED) 10, 98–129 (1999)
8. Tadepalli, P.: Learning to solve problems from exercises. Computational Intelligence 24(4), 257–291 (2008)
9. Tecuci, G., Boicu, M., Boicu, C., Marcu, D., Stanescu, B., Barbulescu, M.: The disciple-RKF learning and reasoning agent. Computational Intelligence 21(4), 462–479 (2005)
10. Woolf, B.P.: Building Intelligent Interactive Tutors: Student-centered strategies for revolutionizing e-learning. Elsevier & Morgan Kaufmann, Burlington, MA (2008)