INTRODUCING MACHINE LEARNING FROM AN AI PERSPECTIVE

Ingrid Russell[1], Zdravko Markov[2], and Neli Zlatareva[3]

Abstract – *This paper presents our approach of introducing Machine Learning from an AI perspective. We present an AI course with a Machine Learning component. We also discuss some of the examples and projects we used to introduce various search algorithms and show how they can be extended into projects that incorporate ML techniques. The N-puzzle problem is used as a theme throughout the various implementations, from the more conventional search algorithms to the implementation of a simple learning technique utilizing Explanation-Based Learning.*

I. INTRODUCTION

While many undergraduate computer science and computer engineering programs do not offer a course in Machine Learning (ML), a number of them do offer a course in Artificial Intelligence (AI) and in some cases require such a course. The course is intended to provide students with basic knowledge of the theory and practice of Artificial Intelligence as a discipline.

One of the goals of the field of AI is to provide tools and techniques for solving problems that have been difficult to solve by other methods. Several of the problems that AI deals with involve searching for a solution in a large search space, and storing and manipulating significant amounts of data. Hence search methods and formalisms for knowledge representation and reasoning are common themes in AI problems and therefore are important topics to be covered in an AI course. While several flavors of undergraduate AI courses exist with a variety of foci, these topics are present in all.

A common approach to teaching an AI course is to include a major component that covers search techniques and knowledge representation and reasoning, and then provide a short introduction to several sub-fields of AI such as natural language processing, vision, robotics, and others. However, our experience using this approach has been less than ideal. Given the time limitation, such an introduction to several areas of AI is generally brief and presented hastily, and in turn does not allow students to get an appreciation of any of the fields presented nor does it give any degree of in-depth coverage of any one area.

An alternative that we have successfully implemented is to cover one area with some degree of depth. We chose to present ML for two main reasons. First, learning is becoming an increasingly important area of computer science that is playing a major role in a wide range of applications. Second, the coverage of search algorithms in an AI course provides an ideal setting allowing us to easily expand such coverage to machine learning algorithms. Machine learning algorithms also provide excellent examples of heuristic approximation algorithms. The goal is to introduce students to the basic concepts and techniques of ML, and to allow them to implement a simple learning system.

We will present such an AI course along with the ML module. We will discuss some of the examples and projects we used to introduce various search algorithms, and show how they can be extended into projects that incorporate ML techniques. The N-puzzle problem is used as a theme throughout the various implementations, from the more conventional search algorithms to the implementation of a simple learning technique utilizing Explanation-Based Learning.

II. AI COURSE OUTLINE

The AI course provides students with basic knowledge of the theory and practice of Artificial Intelligence as a discipline concerning intelligent agents capable of deciding what to do and doing it. The course introduces the fundamental problem solving techniques and knowledge representation paradigms in AI.

The beginning of the course covers a brief introduction to the LISP programming language. This is followed by problem solving techniques including problem spaces, uninformed as well as informed search techniques, and the role of heuristics. Two player games and constraint satisfaction problems are covered next along with planning techniques. The course then covers knowledge representation schemes including predicate logic, non-monotonic inference, probabilistic reasoning, production systems, semantic nets and frames. The last part, which in other courses typically consists of exposure to several AI fields, consists of approximately three weeks of coverage of machine learning concepts and algorithms. Students are also expected to write a paper on an AI topic not covered in the course and present it in class. This is their opportunity to research an AI area of interest and gain exposure to other AI fields. With the exception of the learning module, the course is based on [6].

The following sections describe the projects assigned in class. The first project is on conceptual search utilizing the N-puzzle problem. Its goal is a study of the performances of basic search algorithms. The second project is on Explanation-Based Learning (EBL) and its application to the

[1] Ingrid Russell, University of Hartford, Department of Computer Science, West Hartford, CT 06117, irussell@hartford.edu, 860-768-4191.
[2] Zdravko Markov, Central Connecticut State University, Department of Computer Science, New Britain, CT 06050, markovz@ccsu.edu, 860-832-2723
[3] Neli Zlatareva, Central Connecticut State University, Department of Computer Science, New Britain, CT 06050, zlatareva@ccsu.edu, 860-832- 2711

N-puzzle problem. It aims to show the students how EBL helps search algorithms generate more efficient solutions.

III. SEARCH ALGORITHMS AND THE N-PUZZLE PROBLEM

A classic problem, the N-puzzle problem, serves as a good application for illustrating conceptual AI search in an interesting and motivating way. In the 8-puzzle version, a 3×3 board consists of 8 tiles numbered 1 through 8 and an empty tile (marked as 0). One may move any tile into an orthogonally adjacent empty square, but may not move outside the board or diagonally. The problem is to find a sequence of moves that transforms an initial board configuration into a specified goal configuration. The following are examples of an initial and goal configurations:

Initial configuration	Goal configuration
1 6 2	1 2 3
5 7 3	4 5 6
0 4 8	7 8 0

The N-puzzle project is divided into several parts tackled by students in the following sequence:

- Implementation of a function which given a state (current configuration) generates all new states reachable from that state, and its incorporation into a conceptual search program.
- Empirical study of uninformed search algorithms (breadth-first, depth-first, depth-limited, iterative deepening).
- Design and implementation of suitable heuristic functions for the N-puzzle problem (number of tiles in proper places and Manhattan distance among them).
- Empirical study of performance of informed (best-first) search with different heuristic functions.
- Comparison of performance of informed and uninformed search.

Students can do various kinds of experimentation. Students are asked to compare the performance of search algorithms based on:

- Size of the search space.
- Length of the solution an algorithm returns.
- Number of times an algorithm backtracks.
- Number of states examined.
- Elapsed time.

While interesting initial configurations can be given to students, we chose to ask students to come up with meaningful initial states by starting with the goal state and simulating backtracking.

A. N-Puzzle and Uninformed Search Algorithms

Students are given the LISP source code of the depth-first and breadth-first searches for the 8-puzzle and 5-puzzle problems. The code is available at the course web site [7]. Students are then asked to compare the performance of depth-first and breadth-first searches. They are advised to use the implementation of the 5-puzzle instead of the 8-puzzle since it has a smaller search space. By experimenting with these algorithms, students see, in the case of breadth-first search, how quickly they can run out of memory due to the large search space associated with the problem (even for the 5-puzzle version).

Students also discover how the depth-first search memory requirement is less than for breadth-first search (its space complexity is polynomial versus exponential space complexity for the breadth-first search). On the other hand, depth-first search can lead to a dead-end or continue infinitely. In addition, it is not guaranteed to find a solution even if one exists, and if it finds a solution, the solution is not guaranteed to be optimal. However, the polynomial space complexity of the depth-first search makes it a practical choice in larger applications. Breadth-first search, on the other hand, has a large memory requirement since it requires the entire search space to be stored in memory. However, it is guaranteed to find a solution that is nearest to the initial state (the shortest path), if one exists.

Another uninformed search that students study in this project is the depth-limited search. It is a variation of the depth-first search, where a maximum depth is set in advance. Like depth-first search, it expands along one path at a time, but switches to a different path when the preset depth limit is reached. Depth-limited search avoids the main problem of the depth-first search – following a given path infinitely, which makes it complete, but not optimal. Another variation of it, iterative deepening search, is also studied as part of this project.

These exercises provide for an excellent complexity analysis and illustrate the tradeoff between time efficiency, space efficiency, and quality of the solution found.

B. N- Puzzle and Informed Search: the Role of Heuristics

As a result of the first part of the project, students discover the limitations of uninformed searches, and hence the need for domain specific knowledge (heuristics) to guide the search. A good heuristic function can considerably increase the quality of the search. It evaluates the promise of each possible state and guides the search along the path of the most promising states toward a solution. While a good heuristic function helps significantly in the search, it does not guarantee an optimal solution.

Several types of heuristics for the N-puzzle problem are presented in class. Students are given a revised source code implementing one of these heuristics, namely the number of tiles in place, and are asked to implement the Manhattan

distance. Students are asked to do various kinds of experimentation and to identify convergences. Finally, students are asked to compare the performance of informed and uninformed searches, and write a report with their findings.

C. Learning and the N-Puzzle Problem

Planning algorithms and machine learning techniques are important in several areas of AI and hence their in-depth coverage is important in such a course. While at times an agent may be able to react immediately, there are times where planning and evaluating potential actions is important. Learning, therefore, is a particularly important concern when building intelligent systems. In a similar way, computer systems and programs are limited by the designer/programmer's limitations. Learning allows a system to adapt and improve its performance based on experience. Such applications are widespread in areas such as natural language processing, computer vision, robotics, among others.

A variety of machine learning systems are available for students to explore. Mitchell has a neural network computer vision project [5], Dietterich has a hyphenation project involving learning to hyphenate English words [1], and document (Web page) classification is available at [3], etc. In addition, simulations of various machine learning algorithms as well as applications of these algorithms are also available [2].

In our course, students are asked to incorporate Explanation-Based Learning (EBL) into the N-puzzle problem. This allows them to better understand the concepts of learning, and to see how learning improves the performance of search algorithms.

While the N-puzzle problem may not be the ideal domain for EBL, we selected it for several reasons. First, the N-puzzle problem is an interesting and relatively simple example to illustrate the concepts of EBL, and how learning can help convergence to a solution. In addition to its use in search, EBL is used in another main area covered in the AI course, knowledge-based systems, for knowledge base refinement.

The following sections cover the basics of EBL as covered in class. Students are asked to implement the concepts learned and apply EBL learning to the N-puzzle problem to achieve better performance. Students see how their previous implementations can benefit from incorporating machine learning.

IV. Basics of Explanation-Based Learning

Many learning algorithms (usually considered as inductive learning) generalize on the basis of regularities in training data. These algorithms are often referred to as similarity based, i.e. generalization is primarily determined by the syntactical structure of the training examples and the use of domain knowledge is limited to specifying the hypothesis language and exploring the hierarchy of the attribute values. Typically a learning system uses domain knowledge and is expected to have some ability to solve problems. Then the objective of learning is to improve the system's knowledge or system's performance using that knowledge. This task could be seen as knowledge reformulation or theory revision.

EBL uses a domain theory to construct an explanation of the training example, usually a proof that the example logically follows from the theory. Using this proof the system filters the noise, selects only the aspects of the domain theory relevant to the proof, and organizes the training data into a systematic structure. This makes the system more efficient in later attempts to deal with the same or similar examples. The basic components in EBL, as introduced in [4] are the following:

- *Target concept*. The task of the learning system is to find an effective definition of this concept. Depending on the specific application the target concept could be a classification rule, theorem to be proven, a plan for achieving goal, or heuristic to make a problem solver more efficient (e.g. a state space search heuristic).
- *Training example*. This is an instance of the target concept. For example, this may be a good (efficient) solution in a state space search.
- *Domain theory*. Usually this is a set of rules and facts representing domain knowledge. They are used to explain how the training example is an instance of the target concept.
- *Operationality criteria*. Some means to specify the form of the concept definition. In other words this is the language of expressing the target concept definition, which is usually a part of the language used in the domain theory.

In the form outlined above, EBL can be seen as partial evaluation. In terms of theorem-proving, this technique is also called unfolding, i.e. replacing body goals with the bodies of the rules they match, following the order in which goals are reduced (depth-first). Hence in its pure form EBL doesn't learn anything new, i.e. all the rules inferred belong to the deductive closure of the domain theory. This means that these rules can be inferred from the theory without using the training example at all. The role of the training example is only to focus the theorem prover on relevant aspects of the problem domain. Therefore EBL is often viewed as a form of speed-up learning or knowledge reformulation. Consequently EBL can be viewed not as a form of generalization, but rather as specialization, because the rule produced is more specific than a theory itself (the EBL rule is applicable to only one example). All this however does not undermine EBL as a Machine Learning approach. There are many reasons to support this. Hereafter we discuss the one that shows the advantages of EBL as a speed-up learning approach.

There are small and well defined theories, however practically inapplicable. For example, consider the game of chess. The rules of chess combined with an ability to perform unlimited look-ahead on the board states will allow a system to play well. Unfortunately this approach is not practically useful. An EBL system, given well chosen training examples, will not add anything new to the rules of chess playing, but will actually learn some heuristics to apply these rules, which might be practically useful. The N-puzzle domain is another typical example of this approach. As the search space is huge, any practical solution requires heuristics. And the role of EBL is to learn such heuristics from examples of successful searches.

V. EBL IN THE N-PUZZLE DOMAIN

A. Basic idea

Consider the domain theory of the 8-puzzle problem. It can be expressed by a set of facts describing state transitions, and a search engine that can be used to find paths between initial and goal states. Given a pair of an initial and a goal state (a training example), the search algorithm finds the shortest path between them (explanation or proof). Then applying the EBL techniques, the path is generalized so that it can be used later to match other initial states and bring the search algorithm directly to the goal state, without the resource-consuming exploration of the huge state space of the game. With carefully chosen training examples, useful rules for typical moves can be learnt and then integrated into the search algorithm to achieve better performance.

B. Game representation

Hereafter we discuss the 5-puzzle problem. In this representation tiles are numbered 1, 2, 3, 4, 5. The empty square (no tile) is represented by 0. The state of the game is represented by a list of tiles (including 0), where their position in the list corresponds to their board position. For example (1,2,3,4,5,0) correspond to the following board:

```
-------------
| 1 | 2 | 3 |
-------------
| 4 | 5 | 0 |
-------------
```

C. State transitions

The state transitions are represented by reordering tiles in the list. For this purpose we use variables, so that the number of transitions that have to be described is minimized. Positions are mapped to variables, which hold the actual tile numbers as follows:

```
-------------------------
| first | second | third |
-------------------------
| fourth| fifth  | sixth |
-------------------------
```

For example, moving the empty tile from position 1 to position 2 is represented as transforming state

`(first,second,third,fourth,fifth,sixth)`

into state

`(second,first,third,fourth,fifth,sixth)`,

where all list elements except for `first` (which holds the 0) can take actual tile numbers form 1 to 5 (all different). Thus, this generalized transition represents 5! actual transitions between game states.

Depending on the position of the empty tile (0), we may have two or three possible transitions of the type discussed above. In LISP, this is implemented as a function, which adds the new states in the beginning of the list of current states. For example, the function below extends the currents list of states `new-list` with two new states, which are the two possible transitions from `state`.

```
(defun move-1 (state)
 (setf new-states (cons(append
    (list (second state))(list (first state))
    (nthcdr 2 state)) new-states))
 (setf new-states (cons(append
    (list (fourth state))(list (second state))
    (list (third state)) (list (first state))
    (nthcdr 4 state)) new-states)))
```

Here is an example of extending state `(0 1 2 3 4 5)` by using the `move-1` function:

```
> (setf new-states (list (list 0 1 2 3 4 5)))
((0 1 2 3 4 5))
> (move-1 (list 0 1 2 3 4 5))
((3 1 2 0 4 5) (1 0 2 3 4 5) (0 1 2 3 4 5))
```

D. Search algorithm

Any uninformed search algorithm that is able to find the shortest path in a graph can be used here. As the goal of EBL is to improve the efficiency, the algorithm can be a simple one. *Iterative deepening* and *breadth-first search* are good choices, because they have high computational complexity. Thus after EBL the speed-up would be easily measured by the reduction of the size of the path between initial and goal states and run time and memory usage. Here is an example of solving the 5-puzzle with breadth-first search (the print shows the path between `start` and `finish` found by the algorithm):

```
> (setf start '(4 5 3 0 1 2))
> (setf  finish '(1 2 3 4 5 0))
> (breadth-first start finish)
((4 5 3 0 1 2) (0 5 3 4 1 2) (5 0 3 4 1 2)
 (5 1 3 4 0 2) (5 1 3 4 2 0) (5 1 0 4 2 3)
 (5 0 1 4 2 3) (0 5 1 4 2 3) (4 5 1 0 2 3)
 (4 5 1 2 0 3) (4 0 1 2 5 3) (4 1 0 2 5 3)
 (4 1 3 2 5 0) (4 1 3 2 0 5) (4 1 3 0 2 5)
 (0 1 3 4 2 5) (1 0 3 4 2 5) (1 2 3 4 0 5)
 (1 2 3 4 5 0))
```

E. Training example and target concept

First we specify a *training example*, as a pair of start and finish states. Let us consider the transition from state `(4 5 3 0 1 2)` to `(5 1 3 4 2 0)`. According to the EBL principles, the training example is an instance of the target concept. So, we have to run the algorithm in order to verify that this is a *correct training example*, i.e. it is an instance of a *correct target concept*:

```
> (setf start '(4 5 3 0 1 2))
> (setf finish '(5 1 3 4 2 0 ))
> (breadth-first start finish)
((4 5 3 0 1 2) (0 5 3 4 1 2) (5 0 3 4 1 2)
 (5 1 3 4 0 2) (5 1 3 4 2 0))
```

F. EBL generalization

In our setting, EBL generalization is simply substituting constants for variables. Following the representation adopted here, we get a new generalized transition from state

`(first,second,third,fourth,fifth,sixth)`

to state

`(second,fifth,third,first,sixth,fourth),`

where the following substitutions apply:

```
first  = 4 second = 5 third  = 3
fourth = 0 fifth  = 1 sixth  = 2
```

G. Improving the search (in EBL terms: improving the domain theory)

The last step in EBL is to add to new target concept definition to the domain theory. In the particular example, this means defining a new function that will allow the search algorithm to use the new state transition. In our LISP implementation we have to modify the `move-4` function in order to add the new state to the resulting list.

```
(defun move-4 (state)
 (setf new-states (cons (append
    (list (second state))(list (fifth state))
    (list (third state))(list (first state))
    (list (sixth state))(list (fourth state)))
     new-states))
 (setf new-states (cons (append
    (list (fourth state))(list (second state))
    (list (third state))(list (first state))
    (nthcdr 4 state)) new-states))
 (setf new-states (cons (append
    (butlast state 3)(list (fifth state))
    (list (fourth state))(last state))
     new-states)))
```

It is important to note that the new state transition generated by EBL should be used first by the search algorithm. We achieve this by adding the new state as a first element of the path extension (the new state is added by the first `setf`).

To preserve the completeness of the algorithm (in EBL terms: *completeness of the theory*), the new transitions should not replace the original basic ones (one-tile move). Rather, it should be just added, thus expanding the search space with new transitions. This is implemented in our `move-4` function too, as it returns also the state transitions based on one-tile moves.

The newly learned EBL state transition may represent useful search heuristics. To achieve this, however, the training examples have to be carefully chosen. They should represent expert strategies to solve the game or at least pieces of such strategies. In fact, our training example was chosen with this idea in mind. Thus, the newly learnt concept (incorporated in `move-4`) improves the efficiency of the algorithm. This can be shown with the same pair of start and finish states that produced a path of 19 states with the standard breadth-first search. Now the path has only 13 states, which means that the new transition is used twice during the search.

```
> (setf start '(4 5 3 0 1 2))
> (setf  finish '(1 2 3 4 5 0))
> (breadth-first start finish)
((4 5 3 0 1 2) (5 1 3 4 2 0) (5 1 0 4 2 3)
 (5 0 1 4 2 3) (0 5 1 4 2 3) (4 5 1 0 2 3)
 (4 5 1 2 0 3) (4 0 1 2 5 3) (4 1 0 2 5 3)
 (4 1 3 2 5 0) (4 1 3 2 0 5) (4 1 3 0 2 5)
 (1 2 3 4 5 0))
```

H. EBL Project Description

- Students are asked to identify useful search heuristics and generate and verify the corresponding EBL training examples.
- Next, students are asked to perform experiments with training examples and measure the improvement in terms of run time and memory requirements. They also are asked to measure the effect of learning if too many or bad examples are supplied.

VI. CONCLUSION

We presented our experiences integrating Machine Learning techniques into an AI course. Several projects that students worked on throughout the course were presented. Overall, student experiences were very positive. While covering the main AI topics, the course provided students with an introduction to and an appreciation of an increasingly important area in AI, Machine Learning. Using a unified example, the N-puzzle problem, throughout the course proved to be helpful and motivating for the students. Students saw how simple search programs evolve into more interesting ones, and finally into a learning framework with interesting theoretical and practical properties.

REFERENCES

[1] Diettrich, T., English Words Hyphenation, **http://cs.oregonstate.edu/~tgd/classes/534/**

[2] Georgiopoulos, M., CRCD in Machine Learning, **http://www.seecs.ucf.edu/ml/**

[3] McCallum, A., Document Classification, **http://www.cs.cmu.edu/~mccallum/bow/rainbow/**

[4] Mitchell, T. M., R. M. Keller and S. T. Kedar-Cabelli, *Explanation-Based Generalization: A Unifying view*, *Machine Learning* 1, Kluwer, Boston, 1986, pp. 47-80.

[5] Mitchell, T., Neural Network Computer Vision Project, **http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html**

[6] Russell S.J., and Norvig P., *Artificial Intelligence: a modern approach*, Upper Saddle River, NJ: Prentice-Hall, 1995.

[7] Zlatareva, N., Source Code for 8-Puzzle Problem, **http://www.cs.ccsu.edu/~neli/cs462/cs462Sp03.html**