Pedagogical Possibilities for the N-Puzzle Problem

Zdravko Markov[1], Ingrid Russell [2], Todd Neller[3], and Neli Zlatareva[4]

Abstract - **In this paper we present work on a project funded by the National Science Foundation with a goal of unifying the Artificial Intelligence (AI) course around the theme of machine learning. Our work involves the development and testing of an adaptable framework for the presentation of core AI topics that emphasizes the relationship between AI and computer science. Several hands-on laboratory projects that can be closely integrated into an introductory AI course have been developed. We present an overview of one of the projects and describe the associated curricular materials that have been developed. The project uses machine learning as a theme to unify core AI topics in the context of the N-puzzle game. Games provide a rich framework to introduce students to search fundamentals and other core AI concepts. The paper presents several pedagogical possibilities for the N-puzzle game, the rich challenge it offers, and summarizes our experiences using it.**

Index Terms – Artificial Intelligence Education, Games, Machine Learning, N-Puzzle Problem, Search.

INTRODUCTION

Search is a fundamental concept to the Computer Science data structures and algorithms courses as well as to the artificial intelligence course. In AI the concept of search is usually introduced by using the idea of state space representation of games. Games provide a rich framework to introduce students to search fundamentals in a motivating and entertaining way. Simple games with large state space serve best this purpose because they illustrate the huge computational cost of problem solving that humans can easily do.

The N-puzzle game is among the classical games that have been used extensively in this area. Along with its basic role as an illustrative example, the N-puzzle game can be used for lab experiments and student projects as well. Its simple representation and the possibility to change the size of the board (the parameter N) allow students to easily implement various approaches to solve the game at different levels of complexity. While using this game in our AI courses, we have found an interesting application of the N-puzzle framework in teaching machine learning [7]. This is a classical situation in AI, where changing the representation brings new insights into a well known problem. Instead of a state-space representation, we consider the Explanation-Based Learning (EBL) setting with a domain theory defined as a set of facts describing the basic moves of the game. 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 Explanation-Based Generalization (EBG) 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 learned and then integrated into the search algorithm to achieve better performance. In this way we illustrate the basics of EBL, theory revision, and other concepts related to analytical learning approaches.

We present a project that uses machine learning as a theme to unify core AI topics typically covered in the AI course using the N-puzzle game. Further in the paper we present several pedagogical possibilities for the game, the rich challenges it offers, and summarize our experiences using it.

This work is part of a larger project Machine Learning Laboratory Experiences for Introducing Undergraduates to Artificial Intelligence (MLExAI). An overview of this NSF-funded work and samples of other course materials developed under this grant are published in [2, 3] and are available at the project website at http://uhaweb.hartford.edu/compsci/ccli. We will first present an introduction to project MLExAI. This will be followed by a presentation of the N-puzzle project and how machine learning is used as a theme to tie together the core AI concepts of search and knowledge representation and reasoning that are typically covered in an introductory AI course. Our experiences using the N-puzzle project in our introductory AI course and its role in enhancing student learning experience are also discussed.

PROJECT MLEXAI

Project MLExAI is funded by the National Science Foundation with a goal of unifying the Artificial Intelligence (AI) course around the theme of machine learning. Our work involves the development and testing of an adaptable framework for the presentation of core AI topics that emphasizes the relationship between AI and computer science. A suite of adaptable hands-on laboratory projects that can be closely integrated into an introductory AI course is being developed.

Our work incorporates machine learning as a unifying theme for the AI course. Machine learning is inherently connected with the AI core topics and provides methodology and technology to enhance real-world applications within many of

[1] Zdravko Markov, Central Connecticut State University, markovz@ccsu.edu
[2] Ingrid Russell, University of Hartford, irussell@hartford.edu
[3] Todd Neller, Gettysburg College, tneller@gettysburg.edu
[4] Neli Zlatareva, Department of Computer Science, Central Connecticut State University

1-4244-0257-3/06/$20.00 © 2006 IEEE **October 28 – 31, 2006, San Diego, CA**

these topics. Machine learning also provides a bridge between AI technology and modern software engineering. In his article, Mitchell discusses the increasingly important role that machine learning plays in the software world and identifies three important areas: data mining, difficult-to-program applications, and customized software applications [6].

We have developed a suite of adaptable, hands-on laboratory projects that can be closely integrated into an introductory AI course. Each project involves the design and implementation of a learning system which will enhance a particular commonly-deployed application. The goal is to enhance the student learning experience in the introductory artificial intelligence course by (1) introducing machine learning elements into the course, (2) implementing a set of unifying machine learning laboratory projects to tie together the core AI topics, and (3) developing, applying, and testing an adaptable framework for the presentation of core AI topics which emphasizes the important relationship between AI and computer science in general, and software development in particular. The N-puzzle game, presented here, provides a good framework for illustrating conceptual AI search in an interesting and motivating way. In [7], we presented how the N-puzzle problem can be used to introduce machine learning concepts into the undergraduate introductory AI course. Here, the objective of this project is to use Analytical (Explanation-Based) Learning as a theme to tie together core AI concepts of search and knowledge representation and reasoning that are typically covered in an introductory AI course. Hands-on experiments with search algorithms combined with an Explanation Based Learning (EBL) component give students a deep, experiential understanding of the basics of EBL and also of core AI concepts typically covered in such a course.

THE N-PUZZLE GAME

In its most popular 8-puzzle version the game consists of a 3x3 board with 8 tiles and an empty square. One may move any tile into an orthogonally adjacent empty square (no diagonal moves or moves outside the board are allowed). The problem is to find a sequence of moves that transform an initial board configuration into a goal configuration. Below is an example of an initial and a goal configuration.

Initial configuration

1	6	2
4	5	3
7	0	8

Goal configuration

1	2	3
4	5	6
7	8	0

The solution is a sequence of 7 moves (swaps between a tile and the empty tile 0): 5-0, 6-0, 0-2, 0-3, 6-0, 0-5, 0-8.

This simple game provides a rich framework for defining challenging problems and small projects, which can be used in the undergraduate artificial intelligence course and other computer science courses, which include the topic of search. In the next sections we discuss possible problems and example solutions, which can be used in these courses.

PROLOG IMPLEMENTATION

In the context of the AI course the most popular implementation language for search algorithms (and for other techniques too) is LISP. However our preference for this project is Prolog due to the following reasons:

- The Prolog code is very concise and easily understood by students because of the language declarative nature. The authors' experience also shows that Prolog representations and algorithms can be used by students at query level without the need of going into programming details.
- By simple queries students can experiment with basic components of AI algorithms and easily combine them in more complex ones without the need of programming. This feature of Prolog will be illustrated in this paper.
- Prolog suits very well other important AI topics such as first order logic and reasoning because it is based on two techniques fundamental to these topics – unification and resolution.
- The Prolog meta-programming features (run time modification of the database) allow straightforward implementation of the Explanation-Based Generalization techniques that we use in our analytical learning framework illustrated with the N-puzzle problem.
- There exists an excellent implementation of the language – SWI-Prolog [10], which is simple, efficient and free, has a very good documentation and comes with many additional useful modules and libraries.

Further in the paper we shall illustrate some of these advantages of using Prolog for teaching search in AI. We have made available a number of Prolog programs that we have developed to accompany the AI course [4]. An introduction to Prolog can be found in [5]. Prolog implementations of major AI algorithms including search are found in [1].

STATE SPACE REPRESENTATION

For brevity hereafter we discuss a downsized version of the game – 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, the structure `s(0,1,2,3,4,5)` corresponds to the following board:

0	1	2
3	4	5

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:

A	B	C
D	E	F

For example, moving the empty tile from position A to position B is represented as transforming state `s(A,B,C,D,E,F)` into state `s(B,A,C,D,E,F)`, where all variables except for A (which holds the 0) can take tile numbers form 1 to 5 (all different). This generalized transition represents 5! actual transitions between game states. The constraint for the empty tile is represented by using 0 instead of a variable. For each position of the empty tile (0) we have two or three transitions. We show below 5 Prolog facts representing the transitions when the empty tile is in position A and position B (we use the name arc, because these are actually arcs in the state space graph):

```
% empty tile in position A
arc(s(0,B,C,D,E,F),s(B,0,C,D,E,F)).
arc(s(0,B,C,D,E,F),s(D,B,C,0,E,F)).

% empty tile in position B
arc(s(A,0,C,D,E,F),s(0,A,C,D,E,F)).
arc(s(A,0,C,D,E,F),s(A,C,0,D,E,F)).
arc(s(A,0,C,D,E,F),s(A,E,C,D,0,F)).
```

By querying the Prolog database (after the facts above are loaded) we can easily see how the transitions work. For example,

```
?- arc(s(0,1,2,3,4,5),X).
X = s(1, 0, 2, 3, 4, 5) ;
X = s(3, 1, 2, 0, 4, 5)
```

These are the two possible moves if the empty tile is in position A. We can easily generate 2 or more step transitions by extending the query. All alternative solutions can be printed by adding `fail` at the end. For example, the following query prints all states that can be reached from `s(0,4,5,1,2,3)` by two transitions.

```
?- arc(s(0,4,5,1,2,3),X),arc(X,Y),
   writeln(Y),fail.
s(0, 4, 5, 1, 2, 3)
s(4, 5, 0, 1, 2, 3)
s(4, 2, 5, 1, 0, 3)
s(0, 4, 5, 1, 2, 3)
s(1, 4, 5, 2, 0, 3)
```

Other types of query-based experiments include searching for paths between two states. For example,

```
?- arc(s(0,4,5,1,2,3),X),arc(X,Y),
   arc(Y,s(4,2,5,1,0,3)).

No
```

The answer "No" means that there is no path with 3 transitions between states `s(0,4,5,1,2,3)` and `s(4,2,5,1,0,3)`. However, if we add one more arc then such a path exists.

```
?- arc(s(0,4,5,1,2,3),X),arc(X,Y),
   arc(Y,Z),arc(Z,s(4,2,5,1,0,3)).

X = s(4, 0, 5, 1, 2, 3)
Y = s(0, 4, 5, 1, 2, 3)
Z = s(4, 0, 5, 1, 2, 3)
```

This kind of experiment actually solves the puzzle however with a predefined number of moves. It is also a natural way to introduce the general recursive approach to search, which in Prolog is straightforward – we need to add the following simple definition to the database:

```
path(X,Y,[]) :- arc(X,Y).
path(X,Y,[Z|P]) :- arc(X,Z), path(Z,Y,P).
```

This definition is an excellent illustration of the declarative programming style of Prolog. It reads: "There is a path between state X and state Y, if (the symbol ":-" stands for logical "if" or implication "←") there is an arc between them, or if there is an arc to another state Z, such that there is a path from Z to Y." The third argument is a list that holds the states between the start and end states. Its explanation can be omitted from class discussions because it involves some list processing details which are not needed for the purposes of this project.

Now the previous queries can be answered by using the path predicate.

```
?- path(s(0,4,5,1,2,3),s(4,2,5,1,0,3),P).

P = [s(4,0,5,1,2,3)] ;

P = [s(4,0,5,1,2,3),s(0,4,5,1,2,3),
     s(4,0,5,1,2,3)] ;
…
```

Note that the search for alternatives (entering a semicolon after the answer) may continue forever. This is a nice way to illustrate loops in search and to show ways how loops can be avoided (see the repeated state in the list).

At this point students should be able to understand the basic idea of the state-space representation and to use simple Prolog queries to experiment with it. Also they should understand the basic idea of recursion and how it helps searching state space. We would suggest the following additional experiments, questions and assignments for independent work:

1. Complete the set of facts to implement all possible transitions from all possible current states (positions of the empty tile). How many are needed?
2. Change the representation accordingly and implement the state space representation of the 8-puzzle problem.
3. Investigate the state space by experimenting with more state space transitions at query level. What is the branching factor? Are there repeated states and how many?
4. Use the path predicate to find paths between states. Investigate when and why infinite loops occur. Suggest ideas to avoid loops.
5. What kind of search does the path predicate implement – depth-first or breadth-first?

SEARCH ALGORITHMS

The next step is to introduce students to search algorithms – *depth-first*, *breadth-first* and *iterative deepening*. Again there is no need to go into the details of the Prolog implementation

of these algorithms. The only thing students need to know is the parameters of the corresponding Prolog predicates so that they can be used at query level. For example, a description such as the following one can be found as a comment in the file search1.pl, which includes the Prolog code for uninformed search algorithms [5].

```
breadth_first(+[[Start]],+Goal,-Path,
              -ExploredNodes)
```

The plus sign (+) indicates input parameters and the minus (-) – output ones. The input parameters have to be instantiated, while the output ones should be free variables. Below is an example of solving the puzzle with breadth-first search.

```
?- breadth_first([[s(4,5,3,0,1,2)]],
   s(1,2,3,4,5,0),P,N),length(P,L).

P = [s(1,2,3,4,5,0), s(1,2,3,4,0,5),
     s(1,0,3,4,2,5), s(0,1,3,4,2,5),
     s(4,1,3,0,2,5), s(4,1,3,2,0,5),
     s(4,1,3,2,5,0), s(4,1,0,2,5,3),...]
N = 1197
L = 19
```

For brevity Prolog does not print long answers and uses "…" instead. If one wants to see the whole path it should be included in a direct output primitive, like `writeln(P).`

The next step is to discuss informed (heuristic search) algorithms – *best-first*, *A-star* and *beam search*. The advantage of using heuristics is measured in terms of *time* and *space complexity*. These two measures are in fact reported by the search algorithms. The returned value of the `ExploredNodes` parameter is an indication of time complexity. The variable called "`NewQueue`", which can be found in the code, holds the size of the queue used by the algorithms. This size shows the space complexity. By inserting "`length(NewQueue,N),writeln(N)`" in the code the queue size can be monitored.

Suggested student projects related to uninformed and informed search that use our N-Puzzle include:

1. Download uninformed and informed search algorithms (depth-first, breadth-first, iterative deepening, best-first, A-star, beam search) from [4] and test the uninformed search algorithms with the transition `s(4,5,3,0,1,2) => s(1,2,3,4,5,0).`
2. Use uninformed search to solve the 8-puzzle problem with initial state `s(2,3,5,0,1,4,6,7,8)` and goal state `s(0,1,2,3,4,5,6,7,8)`:
    - Compare breadth-first, iterative deepening and depth-first.
    - Explain why depth-first fails.
    - Figure out an approach to find game states that can be solved.
3. Implement a heuristic function for the informed search algorithms (see [8], Chapter 4) and solve the 8-puzzle with initial state `s(2,3,5,0,1,4,6,7,8)` and goal state `s(0,1,2,3,4,5,6,7,8).`
4. Use best-first, a-star and beam search. For beam search try n = 100, 10, 1. What changes? Explain why beam search fails with n=1?
5. Compare the results with depth-first, breadth-first and iterative deepening.
6. Collect data about the time and space complexity of solving the above problems with uninformed and informed search algorithms. Analyze the results.

EXPLANATION-BASED LEARNING

There are two major approaches to learning – inductive and deductive. The inductive learning algorithms find regularities in data and create descriptions or predict values for the target concept. Deductive learning systems use domain knowledge and have some ability to solve problems. The objective of deductive 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. Explanation-Based Learning (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 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 solve the same or similar examples. The basic components of EBL are the following [7, 8]:

- *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 the training example as 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 out setting this is the language of first order logic and the constraints associated with the domain theory (e.g. the empty tile 0).

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 an EBL system 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 (applicable to fewer examples) than a theory itself. All this however does not undermine EBL as a Machine 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 impractical. An EBL system, given well chosen training examples, will not add anything new to the rules of playing chess, 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.

EBL IN THE N-PUZZLE DOMAIN

In this part of the project, students are asked to incorporate Explanation-Based Learning (EBL) into the N-puzzle problem. This allows them to better understand the concept of analytical learning, and to see how learning improves performance of search algorithms. The goal is to introduce the student to Analytical (Explanation-Based) Learning using the classical AI framework of search. Hands-on experiments with search algorithms combined with an EBL component give the student a deep, experiential understanding of the basics of EBL and the role it plays in improving the performance of search algorithms in particular and problem solving approaches in general. The problem solving component in our setting for EBL is an uninformed search algorithm that is able to find the shortest path in a graph. As the goal of EBL is to improve the efficiency, the algorithm can be simple and not necessarily efficient. Iterative deepening and breadth-first search are good choices, because they have high computational complexity. Thus after applying EBL the speed-up would be easily measured by the reduction of the size of the path between initial and goal states and the run time and memory usage.

First we specify a *training example*, as a pair of start and end states. Let us consider the transition from state `s(4,5,3,0,1,2)` to `s(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:

```
?- breadth_first([[s(4,5,3,0,1,2)]],
   s(5,1,3,4,2,0),P,N).

P = [s(5,1,3,4,2,0), s(5,1,3,4,0,2),
     s(5,0,3,4,1,2), s(0,5,3,4,1,2),
     s(4,5,3,0,1,2)]
N = 13
```

The next step is *Explanation-Based Generalization (EBG)*. In our setting, EBG is simply substituting constants for variables. Following the representation adopted here, this results in a new generalized transition from state `s(A,B,C,0,E,F)` to state `s(B,E,C,A,F,0)`, where the following substitutions apply: `A=4, B=5, C=3, E=1, F=2`. Note that instead of a variable in position D we use the constant 0. This is needed to keep the constraint of the empty tile (as we explained in the state space representation).

The objective of EBL is improving the domain theory. This is achieved by adding the new target concept definition to the domain theory. In the particular case this means adding a new arc to the database of facts that will allow the search algorithm to use the new generalized state transition.

```
arc(s(A,B,C,0,E,F), s(B,E,C,A,F,0)).
```

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 fact in the beginning of the database. To preserve the completeness of the algorithm (in EBL terms, completeness of the theory), the new transitions should not replace the original ones (one-tile moves). Rather, it should be just added, thus expanding the search space with new transitions.

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 (the new fact in the database) 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 standard breadth-first search.

```
?- breadth_first([[s(4,5,3,0,1,2)]],
   s(1,2,3,4,5,0),P,N),length(P,L).

P = [s(1,2,3,4,5,0), s(4,1,3,0,2,5),
     s(4,1,3,2,0,5), s(4,1,3,2,5,0),
     s(4,1,0,2,5,3), s(4,0,1,2,5,3),
     s(4,5,1,2,0,3), s(4,5,1,0,2,3),...]
N = 647
L = 13
```

Now the path has only 13 states, which means that the new transition is used twice during the search. Note also that the number of explored nodes is reduced from 1197 to 647, which is an indication of improvement in time complexity.

For the EBL phase, we suggest the following student assignments:

1. Identify useful search heuristics and generate and verify the corresponding EBL training examples.
2. Perform experiments with training examples and update the state transition database manually. Then measure the improvement in terms of time and space complexity after the EBL step.
3. Implement automatic update of the theory given a training example. This includes verifying the example, EBL generalization and incorporating the new generalized

transition into the search algorithm (adding an arc to the database).
4. Evaluate the effect of learning if too many or bad examples are supplied.

KNOWLEDGE REPRESENTATION AND REASONING

As previously stated, the goal of project MLExAI is to use machine learning as a theme to tie together core AI concepts in an effort to enhance student experiences in the introductory AI course. In addition to search techniques, knowledge representation and reasoning are considered to be core AI topics in the course. The N-puzzle project has been used to introduce students to some basic notions of knowledge representation and reasoning. The EBL framework used in the project can provide the intuition for this. In EBL, the transitions between game states are considered as domain knowledge (theory) and the graph search algorithm as a deductive reasoning system. In these terms a solution of the game is an example that logically follows from the theory and the EBG step is a theory specialization (refinement). Further, the notion of incomplete theory can be illustrated by unreachable game states (if some state transitions are removed).

THE N-PUZZLE GAME IN DATA STRUCTURES

Most of the components of the N-puzzle project can be used in the data structures course. The game representation is a good exercise for dynamic data structures such as records or objects. Various aspects of memory optimization can be discussed in this context. The other important topic is trees and graphs, which are used for the implementation of the N-puzzle state space. For this topic basic graph algorithms as depth-first search and breadth-first search can be studied and their implementations based on queues can be discussed. Heuristics for evaluating the game states can be used to introduce the A* algorithm. Because of its large state space the N-puzzle game is suitable for discussing the computational complexity of graph search algorithms and various ways to improve their efficiency.

DISCUSSION AND EXPERIENCES

Over the last two years, we have used the curricular materials relating to uninformed and informed search in the data structures and algorithms course with positive results. The complete N-puzzle project presented here along with other projects developed as part of project MLExAI have been tested in the introductory AI course over the last two years.

Evaluation forms filled out by students as well as feedback from students revealed a high level of satisfaction with the course. Students in the AI course liked being able to apply the problem solving techniques to a "real" situation and to see how they worked. Their experience with the course heightened their awareness of the importance of AI as well as their ability to see a variety of situations in which it could be used. They felt that they had a good understanding of both artificial intelligence and machine learning as a result of taking this course. Students also stated that they would like to learn more about both areas. They felt that they had gained a good grasp of AI problem solving techniques and wanted to have more opportunities to apply them. Students indicated that they had a very positive experience in the courses using this material.

These curricular materials as well as the modules developed under the MLExAI project have been revised based on our experiences. Further testing of the material is currently underway. Modifications and enhancement to the projects are ongoing as we continue to use and test the material. A comprehensive evaluation of the project is also being planned.

CONCLUSION

We presented curricular material that incorporates machine learning as a unifying theme to teach fundamental concepts typically covered in the introductory artificial intelligence courses. This was done in the context of the N-puzzle game. Additional pedagogical possibilities for the N-puzzle were also presented. Our experiences using the material are also discussed. The projects were well received by the students. By using projects involving games, we provided additional motivation for students. While illustrating core concepts, the projects introduced students to an important area in computer science, machine learning, thus motivating further study.

ACKNOWLEDGEMENT

This work is supported in part by National Science Foundation grant *DUE CCLI-A&I Award Number 0409497*.

REFERENCES

[1] Bratko, I. Prolog Programming for Artificial Intelligence (Third Edition), Addison-Wesley, 2000.

[2] Kumar, A., Kumar D., I. Russell, "Non-Traditional Projects in the Undergraduate AI Course", Proceedings of the SIGCSE 2006 Conference, ACM Press, March 2006.

[3] Markov, Z., I. Russell, T. Neller, Enhancing Undergraduate AI Courses through Machine Learning Projects", *Proceedings of the Frontiers in Education Conference*, IEEE Press, November 2005

[4] Markov, Z., Artificial Intelligence Course, www.cs.ccsu.edu/~markov/ccsu_courses/ArtificialIntelligence.html

[5] Markov, Z., *Quick Introduction to Prolog*, www.cs.ccsu.edu/~markov/ccsu_courses/prolog.txt

[6] Mitchell, T., *Machine Learning*, McGraw Hill, 1997.

[7] Russell, I., Z. Markov, N. Zlaterava, "Introducing Machine Learning from an AI Perspective", *Proceedings of the 13th International Conference on Artificial Neural Networks*, June 2003.

[8] Russell, S. and P. Norvig, *Artificial Intelligence: A Modern Approach (Second Edition)*, Prentice Hall, 2003.

[9] Russell, S. and P. Norvig, Artificial Intelligence: A Modern Approach, Author's Website, http://aima.cs.berkeley.edu/.

[10] *SWI-Prolog home page*, http://www.swi-prolog.org/