Speedup Learning

Alan Fern
School of Electrical Engineering and Computer Science
Oregon State University

July 2, 2009

Definition

Speedup learning is a branch of machine learning that studies learning mechanisms for speeding up problem solvers based on problem solving experience. The input to a speedup learner typically consists of observations of prior problem-solving experience, which may include traces of the problem solver's operations and/or solutions to solved problems. The output is knowledge that the problem solver can exploit to find solutions more quickly than before learning without seriously effecting solution quality. The most distinctive feature of speedup learning, compared to most branches of machine learning, is that the learned knowledge does not provide the problem solver with the ability to solve new problem instances. Rather, the learned knowledge is intended solely to facilitate faster solution times compared to the solver without the knowledge.

Motivation and Background

Much of the work in computer science and especially artificial intelligence aims at developing practically efficient problem solvers for combinatorially hard problem classes such as automated planning, logical and probabilistic reasoning, game playing, constraint satisfaction, and combinatorial optimization. While it is often straightforward to develop optimal problem solvers for these problems using brute-force, exponential-time search procedures, it is generally much more difficult to develop solvers that are efficient across

a wide range of problem instances. The main motivation behind speedup learning is to create adaptive problem solvers that can learn patterns from problem solving experience that can be exploited for efficiency gains. Such adaptive solvers have the potential to significantly outperform traditional static solvers by specializing their behavior to the characteristics of a single problem instance or to an entire class of related problem instances. The exact form of knowledge and learning mechanism is tightly tied to the problem class and the problem-solver architecture.

Most branches of machine learning, such as supervised classification, aim to learn fundamentally new problem solving capabilities that are not easily programmed by hand even when ignoring efficiency issues—for example, learning to recognize hand-written digits. Speedup learning is distinct in that it is typically applied in situations where hand-coding an optimal, but inefficient, problem solver is straightforward—for example, solving satisfiability problems. Rather, learning is aimed exclusively at finding solutions in a more practical time frame.

Work in speedup learning grew out of various subfields of artificial intelligence and more generally computer science. An early example, from automated planning involved learning knowledge for speeding up the original STRIPS planner [3] via the learning of triangle tables or macros that could later be exploited by the problem solver. Throughout the 80's and early 90's there was a great deal of additional work on speedup learning in the area of automated planning as overviewed in [9, 16].

Another major source of speedup learning research has originated from the areas of AI search and constraint satisfaction. Many of the intelligent backtracking mechanisms from these areas, which are critical to performance, can be viewed as speedup learning techniques [5], where knowledge is learned, while solving a problem instance, that better informs later search decisions. Such methods have also come out of the area of logic programming [7] where search efficiency plays a central role.

In addition, various branches of AI have developed speedup-learning approaches based on learning improved heuristic evaluation functions. Samuel's checker player [11] was one such early example, where learned evaluation functions allowed for the performance of deep game tree search to be approximated by shallower, less expensive, search.

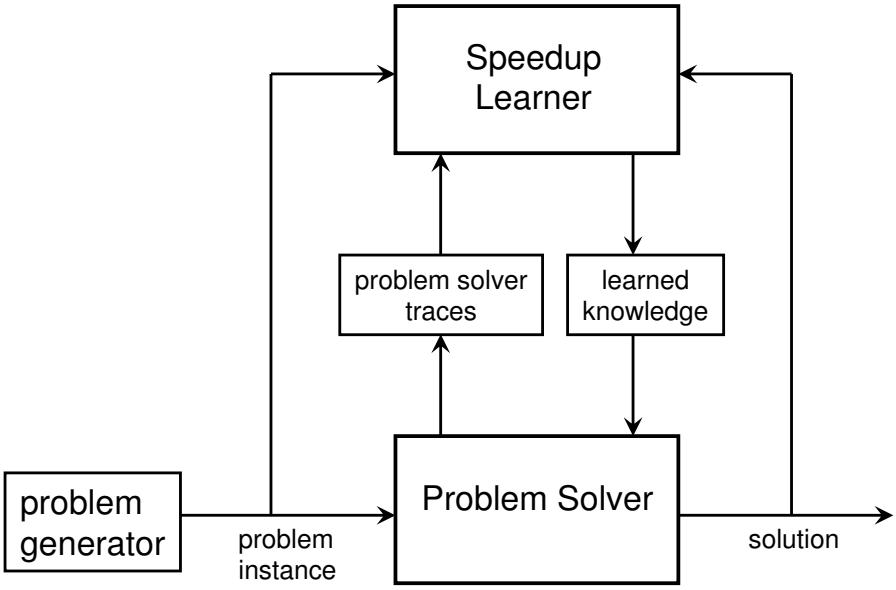

Figure 1: Schematic diagram of a speedup learning system. The problem solver receives problem instances from a problem generator and produces solutions. The speedup learner can observe the input problem instances, traces of the problem solver while solving the problem instances, and sometimes also the solutions to previously solved problem instances. The speedup learner outputs knowledge that can be used by the problem solver to speedup its solution time either on the current problem instance (intra-problem speedup) and/or future related instances (inter-problem speedup).

Structure of Learning System

Figure 1 shows a generic diagram of a speedup learning system. The main components are the problem solver and the speedup learner. The role of the problem solver is to receive problem instances from a problem generator and to produce solutions for those instances. Example problem solvers might include constraint-satisfaction engines, automated planners, or A^* search. The role of the speedup learner is to produce knowledge that the problem solver can use to improve its solution time. The input to the speedup learner, which is analyzed in order to produce the knowledge, can include one or more

of the following data sources: 1) the input problem instances, 2) traces of the problem solver's decisions while solving the input problems, and 3) solutions to solved problems.

Clearly there is a large space of possible speedup learning systems that result from different problem solvers, forms of learned knowledge, learning methods, and intended mode of applicability. Below we first describe some of the main dimensions along which speedup learning approaches can be characterized. Next we provide examples of typical learners that span this space, noting that the examples are far from an exhaustive list.

Dimensions of Speedup Learning

Intra-Problem versus Inter-Problem Speedup. Intra-problem speedup learning is when knowledge is learned during the solution of the current problem instance and is only applicable to speeding up the solution of the current instance. After a solution is found, the knowledge is discarded as it is not applicable to future instances. Inter-problem speedup learning is when the learned knowledge is applicable not only to the problem(s) it was learned on but also to new problems encountered in the future. In this sense, the learned knowledge can be viewed as generalize knowledge about how to find solutions more quickly for an entire class of problems.

Typically in inter-problem learning, the problem generator produces instances that are related in some way and hence share common structure that can be learned from earlier instances and exploited when solving later instances. Rather intra-problem speedup learners treat each problem instance as completely distinct from all others. Also note that inter-problem learners have the potential to benefit from the analysis of solutions to previous problems instances. Rather, intra-problem learners are unable to use this source of information since once the current problem is solved, no further learning is warranted.

Types of Learned Knowledge. Most problem solvers can be viewed as search procedures, which is the view that we will take when characterizing the various forms of learned knowledge in speedup learning. Below we list four types of commonly used knowledge, noting that this is far from an exhaustive list. First, *pruning constraints* are sets of constraints on search nodes that signal when a certain branch of the search space can be safely pruned. Second, *macros operators* are sequences of search operators that are typically useful when executed in order. Problem solvers can often utilize

macros in order to decrease the effective solution depth of the search space by treating macros as additional search operators. It is important that the decrease in effective depth is enough to compensate for the increase in number of operators, which increases the search complexity. Third, *search-control rules* are sets of rules that typically test the current problem solving state and suggest problem-solving actions such as rejecting, selecting, or preferring a particular search operator. In the extreme case, learned search control rules can completely remove the need for search. Fourth, *heuristic evaluation functions* are used to measure the quality of a particular search node. Learning improved heuristics can result in better directed search behavior.

Deductive versus Inductive Learning. Deductive learning refers to a learning process for which the learned knowledge can be deductively proven to be correct. For example, in the case of learned pruning constraints, a deductive learning mechanism would provide a guarantee that the pruning was sound in the sense that the optimality of the problem solver would be unaffected. Inductive learning mechanisms rather are statistical in nature and typically do not produce knowledge with associated deductive guarantees. Rather, inductive methods focus on finding statistical regularities that are typically useful, though perhaps not correct in all cases. For example, an inductive learner may discover patterns that are strongly correlated to pruning opportunities, though these patterns may have a small probability of leading to unsound pruning.

In cases where one must guarantee a sound and complete problem solver, deductive learning approaches are always applicable, though their utility depends on the particular application. In certain cases, inductively learned knowledge can also be utilized in a way that does not effect the correctness of the problem solver. For example, inductively learned search-control rules that assert preferences, rather than prune nodes from the search, do not lead to incompleteness. Traditionally, the primary disadvantage of deductive learning, compared to inductive learning, is that inductive methods typically produce knowledge that generalizes to a wider range of situations than deductive methods. In addition, deductive learning methods are often more costly in terms of learning time as they rely on expensive deductive reasoning mechanisms. Naturally, a number of speedup learning systems exist that utilize a combination of inductive and deductive learning techniques.

Examples of Intra-Problem Speedup Learning

Much of the speedup learning work arising from research in AI search and constraint satisfaction falls into the intra-problem paradigm. The most common forms of learning are deductive and are based on computing explanations of "search failures" that occur during the solution of a particular problem. Here a search failure typically corresponds to a point where the problem solver must backtrack. By computing and forming such failure explanations the problem solver is typically able to avoid similar types of failures in the future by detecting that a search path will lead to failure without fully exploring that path. Nogood learning is a very successful, and commonly used, example of the general failure-explanation approach [13]. Nogoods are combinations of variable values that lead to search failures. By computing and recording nogoods, it is possible to immediately prune search states that consider those value combinations. There are many variations of nogood learning, with different techniques utilizing different approaches to analyzing search failures in order to extract general nogoods.

Another example of the failure-explanation approach, which is commonly utilized in satisfiability solvers, is clause learning. The idea is similar to nogood learning. When a failure occurs during systematic search, a proof of the failure is constructed and analyzed to extract implied constraints, or clauses, that the solution must satisfy. These learned clauses are then added to the set of clauses of the original satisfiability problem and in later search trigger early pruning when they, or their consequences, are violated. Efficient implementations of this idea have lead to huge gains in satisfiability solvers. In addition, it has been shown theoretically that clause learning can improve solution times by an exponential factor [1].

Inductive techniques for learning heuristic evaluation functions have also been investigated in the intra-problem speedup paradigm. Here we discuss just two such approaches, where in both cases the key idea is to observe the problem solver and extract training examples that can be used to learn an accurate evaluation function. A particularly successful example of this approach is the STAGE system [2] for solving combinatorial optimization problems such as traveling salesman and circuit layout. The problem solving architecture used by STAGE is based on repeated random restarts of a fast hill-climbing local optimizer, which when given an initial configuration of the combinatorial object, performs a greedy search to a local minimum configuration. The speedup learning mechanism for STAGE is to learn an

approximate function that maps initial configurations to the performance of the local optimizer when started at that configuration. Note that on each restart of the problem solver the learning component gets a training example that can be used to improve the function. The problem solver uses the learned function in order to select promising configurations from which to restart, rather than choosing randomly. In particular, STAGE attempts to restart from a configuration that optimizes the learned function, which is the predicted best starting point for the hill-climber. This overall approach has shown impressive performance gains in a number of combinatorial optimization domains.

As a second example of inductive learning of heuristics in the intra-problem paradigm, there has been work within the more traditional problem solving paradigm of best-first search [12]. Here the speedup learner observes the sequence of search nodes traversed by the problem solver. For any pair of nodes observed to be on the same search path the learner creates a training example in an attempt to train a heuristic to better predict the distance between those two nodes. Ideally this updated heuristic function will better reflect the distance from nodes in the search queue to the goal node of the current problem instance and hence result in improved search performance.

Examples of Inter-Problem Speedup Learning

Much of the work on inter-problem speedup learning came out of AI planning research, where researchers have long studied learning approaches for speeding up planners. We focus on speedup in planning here, noting that similar ideas have also been pursued in other research areas such as constraint satisfaction. For a collection and survey of work on speedup in planning see [9] and [16]. Typically in this work, one is interested in learning knowledge for an entire planning domain, which is a collection of problems that share the same set of actions. The Blocksworld is a classic example of such a planning domain. After experiencing and solving a number of problems from a target domain, such as the Blocksworld, the learned knowledge is then used to speed up performance on new problems from the same domain.

There have been a number of deductive learning approaches to speedup learning in planning, which are traditionally cited as explanation-based learning (EBL) approaches [10]. EBL for AI planning is strongly related to the failure-explanation approaches developed for CSPs as characterized nicely by [5]. There are two main differences between the inter-problem EBL work in plan-

ning and the intra-problem EBL approaches for CSPs. First, EBL approaches in planning produce more general explanations that are applicable not only in the problem in which they were learned, but also new problems. This is often made possible by introducing variables in the place of specific objects into the explanations derived from a particular problem. This allows the explanations to apply to contexts in new problems that share similar structure but involve different objects. The second difference is that inter-problem EBL approaches in planning often produce explanations of successes and not just of failures. These positive explanations are not possible in the context of intra-problem speedup since the intra-problem learner is only interested in solving a single problem.

Despite the relatively large effort invested in inter-problem EBL research, the best approaches typically did not consistently lead to significant gains, and even hurt performance in many cases. A primary way that EBL can hurt performance is by learning too many explanations, which results in the problem solver spending too much time simply evaluating the explanations at the cost of reducing the number of search nodes considered. This problem is commonly referred to as the EBL utility problem [8] as it is difficult to determine which explanations have high enough utility to be worth keeping.

In addition to EBL, there has also been work on inductive mechanisms for acquiring search-control rules to speedup AI planners. Typically, statistical learning mechanisms are used to find common patterns that can distinguish between good and bad search decisions. As one example, Huang et. al. learn action-rejection and selection rules based on the solutions to planning problems from a common domain [4]. The learned rules were then added as constraints to the constraint satisfaction engine, which served to guide the solver to solution plans more quickly. Another approach, which has been studied at a theoretical and empirical level, is to learn heuristic functions to guide a bounded search process [15], in particular, bread-first beam search. Results in a number of planning domains demonstrate significant improvements over planners that do not incorporate a learning component. One other class of approach is based on attempting to learn knowledge that removes the need for a problem solver altogether. In particular, to learn a reactive policy for quickly selecting actions in any given state of the environment. Such policies can be learned via statistical techniques by simply trying to learn an efficient function that maps planning states to the actions selected by the planner. Despite its simplicity, this approach has demonstrated considerable success [6] and has also been characterized at a theoretical level [14].

Additional Definitions

Title: Supervised Classification
Synonyms: Classifier Learning
Definition: Supervised classification is an important class of machine learning problems where the goal is to learn a function that maps inputs to one of a finite set of class labels. The input to the learner is a supervised training set containing example inputs with the correct class labels. The output is a function that can be used to classify new examples.

Title: Intelligent Backtracking
Synonyms: Dependency Directed Backtracking
Definition: Intelligent backtracking is a general class of techniques used to enhance search and constraint satisfaction algorithms. Backtracking is a general mechanism in search where a problem solver encounters an unsolvable search state and backtracks to a previous search state that might be solvable. Intelligent backtracking mechanisms provide various ways of selecting the backtracking point based on past experience in a way that is likely to be fruitful.

Title: Deductive Learning
Synonyms: Analytical Learning
Definition: Deductive learning is a subclass of machine learning that studies algorithms for learning provably correct knowledge. Typically such methods are used to speedup problem solvers by adding knowledge to them that is deductively entailed by existing knowledge, but that may result in faster solutions.

Title: Inductive Learning
Synonyms: Statistical Learning
Definition: Inductive learning is a subclass of machine learning that studies algorithms for learning knowledge based on statistical regularities. The learned knowledge typically has no deductive guarantees of correctness, though there may be statistical forms of guarantees.

Title: Nogood Learning
Synonyms:
Definition: Nogood learning is a deductive learning technique used for the purpose of intelligent backtracking in constraint satisfaction. The approach analyzes failures at backtracking points and derives sets of variable bindings, or nogoods, that will never lead to a solution. These nogood constraints can then be used to prune later search nodes.

Title: Clause Learning
Synonyms:
Definition: Clause learning is a deductive learning technique used for the purpose of intelligent backtracking in satisfiability solvers. The approach analyzes failures at backtracking points and derives clauses that must be satisfied by the solution. The clauses are added to the set of clauses from the original satisfiability problem and serve to prune new search nodes that violate them.

References

[1] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[2] J. A. Boyan and A. W. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *National Conference on Artificial Intelligence*, pages 3–10, 1998.

[3] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(1–3):251–288, 1972.

[4] Y.-C. Huang, B. Selman, and H. Kautz. Learning declarative control rules for constraint-based planning. In *International Conference on Machine Learning*, pages 415–422, 2000.

[5] S. Kambhampati. On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. *Artificial Intelligence*, 105(1-2):161–208, 1998.

[6] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113(1-2):125–148, 1999.

[7] V. Kumar and Y. Lin. A data-dependency based intelligent backtracking scheme for prolog. *The Journal of Logic Programming*, 5(2):165–181, 1988.

[8] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *National Conference on Artificial Intelligence*, 1988.

[9] S. Minton, editor. *Machine Learning Methods for Planning*. Morgan Kaufmann, 1993.

[10] S. Minton, J. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[11] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[12] S. Sarkar, P. Chakrabarti, and S. Ghose. Learning while solving problems in best first search. *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, 28(4), 1998.

[13] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, 3(2):187–207, 1994.

[14] P. Tadepalli and B. Natarajan. A formal framework for speedup learning from problems and solutions. *Journal of Artificial Intelligence Research*, 4:445–475, 1996.

[15] Y. Xu and A. Fern. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research*, 10:1349–1388, 2009.

[16] T. Zimmerman and S. Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2)(2):73–96, 2003.