

Searching for Intelligent Behavior

Jarosław Arabas and Paweł Cichosz

Institute of Electronic Systems, Warsaw University of Technology, Warsaw, Poland
{jarabas, pcichosz}@elka.pw.edu.pl

Abstract

This article attempts to revive the semi-obvious thesis about close relationships between learning and optimization as two major artificial intelligence tasks. After adopting some taxonomies of most common variants of these tasks we review several connections between them, both on a more general level of task formulation and on a more specific level of algorithms. We show that optimization and learning tasks have much in common and can be combined to mutual benefit. On one hand, learning involves at least some implicit optimization and making it more explicit permits applying a wider range of optimization techniques than those hard-wired into existing learning algorithms. On the other hand, optimization techniques can be augmented by learning applied to operations they need to perform to generate and evaluate candidate solutions. Whereas several examples of such combinations can be found in the literature, we believe there is space for many more techniques that could boost optimization by learning or boost learning by optimization.

Keywords: machine learning, optimization

1 Introduction

Searching and machine learning have been closely related since the very beginning of artificial intelligence. When the machine learning (ML) field was growing up, there was a strong awareness of the role of optimization as a tool to perform learning (Mitchell, 1982). In several fundamental ML algorithms, the optimization aspect has been strongly emphasized by their inventors. Good examples are support vector machines (Vapnik, 1998) and neural networks (Hertz *et al.*, 1991), where the learner can be characterized by a vector of n parameters which are real numbers, and the learning consists in minimization of an error measure in R^n . The real vector based representation makes it possible to use numerical optimization methods which have been developed since the beginning of the computer era. Polak (1997) gives a good overview of that early work.

When learners employ more complex knowledge representation, the optimization aspect of the learning process seems less important in the heuristic definition of the learning algorithm — good examples are ID3 (Quinlan, 1990) or a family of AQ methods (Michalski *et al.*, 1998). Indeed, optimization in the space of decision trees or rule sets is much less intuitive than in R^n , since such symbolic structures cannot be organized into a linear space and the neighborhood relationship must be somehow arbitrarily defined.

In parallel to ML methods, new optimization methods have been developed which are known nowadays under the name *metaheuristics* (Glover and Kochenberger, 2003). An impulse for that development was given by the fact that in “canonical” numerical optimization methods, many rigorous assumptions are made about the search space and the objective function which makes them quite limited for practical use. When available computational power was significantly less than nowadays, these limitations were indispensable to make the method operate in reasonable time, but with the increase of computer power, it is possible to solve much harder tasks.

An example of such nonstandard approach to optimization is a family of methods called evolutionary computation (EC), which roots in the 1960s. Evolutionary techniques were presented by their inventors as methods for learning some behavior or to adapt through the simulation of nature-like evolution (Fogel *et al.*, 1966; Holland, 1975). In more recent texts on EC, the learning aspect of evolution is still present, but it has been gradually vanishing over time. Instead, growing attention has been paid to the relation of EC to global optimization (Bäck and Schwefel, 1993), which was originally a flavor of Evolution Strategies only (Rechenberg, 1994). In the same time, EC researchers have made successful attempts to solve optimization tasks where solutions have various nonstandard representations (Koza, 1992; Michalewicz, 1996). This is perhaps a reason why EC methods are quite frequently used to perform some elements of the machine learning process.

In this article we attempt to characterize relationships between optimization and learning. In Sections 2 and 3 we suggest taxonomies to characterize optimization and learning tasks and establish context for the discussion occupying the remainder of the text, without going into details that are irrelevant for this purpose. Then we show that, in addition to one well studied perspective where learning can be formulated as an optimization process, there exist several other possible relationships between these two domains. We briefly outline these relationships in Section 4. The article is concluded by Section 5.

2 Search and Optimization Methods

It is a well known fact in computer science that universal Turing machines (UTMs) are general models of algorithms that can be programmed on serial computers. The operation of the UTM can be completely described by a set of its states and the execution of a program results in generating a certain sequence of states. UTMs are too abstract to be applied in practical considerations, but the flavor of the UTM, namely searching the space of states, is very inspiring for many kinds of problems, including those related to search and optimization.

2.1 Search Space and the Objective Function

Consider a *search space* X and an *objective function* $f : X \rightarrow R$. Assume that for each point $x \in X$, a set of neighbors $N(x) \subseteq X$ is defined. If a metric $\delta : X \times X \rightarrow R$ exists in X , then the set of neighbors can be defined with the use of a radius and we get $N_r(x) = \{y \in X, \delta(x, y) < r\}$. We will call $N_r(x)$ or $N(x)$ the “neighborhood of point x ”.

Function f is used to rate the quality of points from X and the process of searching for best quality points is called *optimization*. Various versions of the optimization task can be distinguished which differ in the definition of conditions the desired solutions must satisfy. For example, in global minimization we search for a point $\hat{x} = \arg \min_{x \in X} f(x)$ or for a value $\hat{f} = \min_{x \in X} f(x)$, and in local minimization we search for a point $\hat{x} = \arg \min_{y \in N(\hat{x})} f(x)$ or for the value $\hat{f} = \min_{y \in N(\hat{x})} f(x)$. Both local and global minimization have versions in which the minimum is to be located with a certain accuracy.

2.2 Navigation in the Search Space

A search algorithm results in generating a sequence of points $x^* \in X^*$, where X^* denotes the space of all sequences of points from search space X , including the empty sequence. Search problem is a tuple $\langle X, f, S, t \rangle$ where x is the search space, f is the objective function, $S \subseteq X$ is a set of starting points, and $t : X^* \rightarrow \{0, 1\}$ is a termination condition which returns 1 if the sequence of points that have been generated by the algorithm satisfies criteria defined by the problem being solved. With Π we denote the space of all search problems.

A search algorithm can be characterized by its *state space* V . In general, $V = X^* \times U^*$, where U^* is the space of sequences some of auxiliary values; without losing generality, we can assume that $U = R$. We assume that the sequence from U^* which is contained by the state depends neither on the searching problem nor on the search algorithm and is used to model randomness in the search process. The sequence of points yielded by the search algorithm results from iterating a transformation called the *aggregated operator* $O : \Pi \times V \rightarrow X^*$. More precisely, in iteration number i , the state v_i is used which contains a subset of the sequence of points $(x_{01}, \dots, x_{0n_0}, \dots, x_{(i-1),1}, \dots, x_{(i-1),n_{i-1}})$. After applying the aggregated operator, a new sequence $(x_{i1}, \dots, x_{i,n_i})$ is generated. The initial sequence $(x_{01}, \dots, x_{0n_0})$ contains points that have been generated by an *initialization operator* $I : S \times U^* \rightarrow X^*$, where $S \subseteq X$ is the set of starting points.

In many optimization methods the aggregated operator can be split into two elements: *selection* $o_s : V \rightarrow V$ and *variation* $o_v : V \rightarrow X^*$. In the selection step, points from the sequence generated so far are selected to build the current state. Then the variation takes place which consists in generating new points from the current state, so the sequence generated by the aggregated operator equals $O(v) = o_v(o_s(v))$. These points are appended to the sequence generated by the method, and the selection-variation process is iterated again.

2.3 Taxonomy of Optimization Methods

A variety of optimization methods have been introduced so far — see overviews, e.g., by Polak (1997); Du and Pardalos (1998); Horst and Pardalos (1995); Glover and Kochenberger (2003). Here we attempt to provide several taxonomies of optimization methods, taking into account different elements of the method or the method's results. In Table 1 we characterize several optimization methods which seem to be most widely known. If a method is known in different versions, we characterize the basic method, and the characteristics of possible versions is given

in brackets. For example, it is a rule that evolutionary methods are asymptotically complete (label **c** in the column “completeness”), but under some modifications they may become partial (labeled with **p**). To make the table better readable, we give abbreviations of feature names in the forthcoming text.

Determinism. The state of the method is defined as $v = (x^*, u^*)$, $x^* \in X^*$, $u^* \in U^*$. If the sequence u^* is empty then $v = x^*$ and the method is *deterministic* (**d**), i.e., the sequence which is yielded by the method is a function of the initial sequence. Otherwise, the method is *stochastic* (**s**). A specific case of a stochastic method is a *memory free* method (**mf**) when $v = u^*$. If the aggregated operator is a combination of selection and variation, we can speak of deterministic/stochastic selection and/or variation.

State size. The number $l(x^*)$ of points in the sequence $x^* \in v$ may be variable (**v**) or fixed; in the latter case we distinguish *population-based* methods (**pb**), where $l(x^*) > 1$, and *single-point* methods (**sp**), where $l(x^*) = 1$.

Recency: In each iteration of the search method, the sequence of points $x^* \in v$ may be defined as $x^* = x_{m-k}, \dots, x_m$. If there exists such $K < \infty$ that $k \leq K$ for all iterations then the method is a *forgetting* method (**f**), otherwise it is *non-forgetting* (**nf**).

Locality of search. The definition of the aggregated operator may introduce operations that are localized in neighborhoods of points from the method’s state. Consider a sequence y^* which is generated from the state $v = (x^*, u^*)$. If there exists $r < \infty$ such that $(\forall y_i \in y^*)(\exists x_j \in x^*) y_i \in N_r(x_j)$ then the method is *deterministically local* (**dl**). If there exists a number $p \in [0, 1]$ such that $(\exists r < \infty)(\forall y_i \in y^*)(\exists x_j \in x^*) \text{Prob}\{y_i \in N_r(x_j)\} > p$, then the method is *stochastically local* (**sl**). Otherwise, the method is *global* (**g**).

Completeness. First consider a finite search space X . If there is a termination condition t and a finite number n for which the method generates a sequence no longer than n , and the sequence contains all points from X regardless of the initialization operator I and the initial state S , then the method is called *complete* or *exhaustive* (**c**). If X is infinite, completeness is impossible, but we can still speak about an asymptotic equivalent of completeness. Assume that X is measurable, and consider a nonzero measure subset $A \subseteq X$. Assume also that the measure of the search space is finite. The method is called *asymptotically complete* (**ac**) if for any such A , the probability that at least one point from A is contained by the sequence yielded by the method, increases to one with the sequence length increasing. Note that all complete methods are asymptotically complete. If a method is not asymptotically complete, then it is called *incomplete* or *partial* (**p**).

2.4 Feasible and Infeasible Solutions

In some problems, the search space contains points which can be regarded true solutions to the problem — *feasible* solutions, and some points which cannot, since they satisfy only a subset of conditions implied for the true solution — they will be called *infeasible* solutions. For example, if the problem is to find the shortest path

TABLE 1: Taxonomy of several optimization methods.

| method | selection | variation | state size | recency | locality | completeness |
|--------------------------------|-----------|-----------|------------|---------|----------|--------------|
| uniform cost and A* | d | d | v | n | dl | c |
| Monte Carlo | mf | s | – | – | g | ac |
| k -opt (Lin Kernighan) | d | d | sp | y | dl | p |
| 2-phase optimization in R^n | d | d | sp | y | g | p |
| simulated annealing | s | s | sp | n | sl | ac |
| evolutionary computation | s(d) | s | pb | y(n) | sl | ac(p) |
| particle swarm, immune systems | d(s) | s | pb | n | sl | p(ac) |

between a particular pair of nodes in a graph, the search space may contain *all* paths that can be defined in that graph, and feasible solutions will be paths which start and terminate in nodes under consideration. The optimization method may visit both feasible and infeasible points, so an objective function has to be defined for both categories of points.

If the search method is based on a selection-variation process, then multiple application of the variation operator for a certain starting point $x \in X$ may eventually lead to points from a subset $F(x) \subseteq X$ which contains only a fraction of all feasible solutions. Thus, objective function values of points from $F(x)$ can be used to bound the objective function value for the point x .

In the tradition of heuristic search methods like A* method (e.g., Russel and Norvig, 1990) and in numerical optimization (e.g., Polak, 1997) where a *penalty function* approach is applied, the objective function can be defined for infeasible points as a sum $f(x) = g(x) + h(x)$, and for feasible points, as $f(x) = g(x)$. Function $g(x)$ is defined by the same formula as the objective function for feasible points, (e.g., in the task of finding shortest path between points, it equals the total cost of the current path).

In the A* method it is assumed that $(\forall y \in F(x)) g(y) \geq g(x)$. Function h — called a *heuristic function* — assigns to each infeasible point x a correction term $h(x)$ for which $(\exists z \in F(x)) f(z) \geq g(x) + h(x)$. Thus, $g(x)$ is a lower bound for any feasible solution accessible by subsequent variation of x , whereas $g(x) + h(x)$ is a lower bound for the best feasible solution accessible from x . In the penalty function approach, the penalty function h measures the “degree of infeasibility” and equals zero for feasible points. It is desired that $(\forall y \in F(x)) f(y) < g(x) + h(x)$, to encourage the search method to stay within the feasible area.

In A* it is required that for each point $x \in X$ there exists at least one path generated by the subsequent application of the variation operator which leads to the best point from $F(x)$, and along that path value of $g(x) + h(x)$ does not increase. In the penalty method, this property is also highly recommended.

3 Learning

Learning is commonly considered one of crucial capabilities of intelligent systems, both natural and artificial. It is a process of acquiring or improving knowledge needed to perform some task (e.g., to classify objects, to predict numerical function values, to recognize strings in some language, to control an agent) based on some kind of information related to this task (e.g., examples of correct classifications, examples of actual function values, examples of correct and incorrect strings, outcomes of experimental control actions). There is a variety of forms in which learning can be observed for humans and animals that can be hardly unified within a single theoretical or experimental framework. Similarly, the variety of approaches to machine learning is hard to cover within a unified concise discussion, not speaking about a unified theory. We will get around this challenge by sticking to a rather abstract view of learning most of the time and adopting a simple taxonomy of learning tasks. This is by no means intended to be a survey of machine learning which can be found elsewhere (e.g., Carbonell *et al.*, 1983; Kubat *et al.*, 1998; Dietterich, 1997), but just to provide sufficient context for discussing links between learning and optimization in the next section.

3.1 Taxonomy of Learning Tasks

To appropriately highlight the key dimensions along which learning tasks can be classified, let us assume a simple generic learning setup in which the learner is required to generate a *hypothesis* or *model* that represents a mapping from a set of *instances* selected from some *domain* to a set of *labels* that can be assigned to them. The hypothesis is derived by the learner by some *inference* process based on *training information* and (possibly) *background knowledge*. This view applies to most commonly studied learning tasks that can be distinguished by a number of factors reviewed below.

Instance representation. For most common learning tasks instances are single objects from some domain described by a fixed set of discrete or real-valued attributes. This is sometimes referred to as a (fixed-length) *propositional* representation and contrasted to a *relational* representation, in which instances are tuples of objects described by relations to which they belong.

Label range. The range of possible labels that can be assigned to instances is an important aspect of many learning tasks. Most often labels are scalar discrete or numeric values, but they can be also multidimensional or structured.

Training information. This is any kind of input that is passed to the learner that guides the learning process and affects the final hypothesis. In the most common and basic case the training information consists of a set of labeled instances, but there are several other less obvious possibilities, e.g., unlabeled instances, labels assigned to instances selected by the learner (queries), or rewards.

Background knowledge. Apart from being provided training information, in some tasks the learner may possess some (“innate”) background knowledge about the underlying domain. Whereas in most cases background knowledge

TABLE 2: Selected families of learning tasks.

| learning task | instance representation | label range | training information | inference type | training mode | background knowledge |
|-----------------------------|-----------------------------|-------------|---------------------------------|----------------|---------------|----------------------|
| classification | propositional | discrete | labeled instances | inductive | any | optional |
| regression | propositional | numeric | labeled instances | inductive | any | optional |
| inductive logic programming | relational | discrete | labeled instances | inductive | any | optional |
| clustering | propositional | discrete | unlabeled instances | inductive | any | optional |
| grammar learning | string | discrete | labeled instances or queries | inductive | any | optional |
| reinforcement learning | propositional | any | unlabeled instances and rewards | inductive | online | optional |
| analytical learning | propositional or relational | discrete | labeled instances | deductive | any | mandatory |

is optional — for some learning tasks, which are entirely focused on refining what is already known, it may be mandatory.

Inference type. Inference is the process of deriving the hypothesis based on the training information and (optionally) background knowledge (Michalski, 1994). Inference type refers to the principle on which the learner’s inference is based. Most common learning tasks employ some forms of inductive inference.

Training mode. This describes the way of coupling the inference process with the form and delivery of training information. If knowledge derivation is a step-by-step process in which the learner applies its inference mechanism to update the hypothesis based on single pieces of training information, its training mode is *online*. If the complete set of training information is used to generate the hypothesis, we speak about *offline* training mode. Some mix of these two is also not uncommon in practice and may be called *semi-online*.

Performance measure. The hypothesis that is generated through learning has to be evaluated with respect to the task to which it is supposed to be applied. This involves calculating one or more performance measures, based on a set of instances, not necessarily seen during learning. Special care is needed to obtain reliable estimates of the real-life performance of the hypothesis, i.e., the expected performance on new instances. A considerable part of research on machine learning algorithms is devoted to fighting (more realistically, reducing the risk of) *overfitting*, i.e., learning a hypothesis that performs poorly on new instances despite excellent performance on training instances.

Table 2 illustrates different properties of learning tasks summarized above by using them to describe a few most common families of learning tasks. The column corresponding to performance measure is omitted as too much task-specific. Classification, regression, and clustering are three most popular types of inductive learning corresponding to a vast portion of the literature and practical applications.

In particular, the extensive classic work on decision tree induction (e.g., Quinlan, 1993; Breiman *et al.*, 1984) and rule induction (e.g., Michalski, 1973; Michalski *et al.*, 1986; Clark and Niblett, 1989; Cohen, 1995) falls into this category. Inductive logic programming (Muggleton, 1992; Quinlan, 1990) is an extension of classification learning using a relational representation. Grammar or finite-automata learning (e.g., Angluin, 1987; Dean *et al.*, 1995) can be also considered a variation of classification in which instances are represented as strings over a finite alphabet. Reinforcement learning (e.g., Sutton and Barto, 1998; Kaelbling *et al.*, 1996) assumes evaluative training information in the form of real-valued rewards and online learning from a series of interactions with an environment. Whereas for all these families of learning tasks background knowledge can be optionally used to improve the efficiency of learning or hypothesis quality (Pazzani and Kibler, 1992, e.g.), it plays an essential role for analytical learning, also called explanation-based learning (Mitchell *et al.*, 1986), which applies deductive inference to refine the background knowledge.

It does not take much “data mining” effort to notice that training information is the most discriminating feature for the tasks shown in the table. The subsections below outline three major types of learning tasks distinguished by the form of training information: supervised learning, unsupervised learning, and reinforcement learning.

3.2 Supervised Learning

In supervised learning tasks the training information delivered to the learner specifies the labels that should be assigned to a given set of training instances. For such tasks the source of training information is sometimes explicitly referred to as the teacher. The labels of training instances are usually given directly, but — in online or semi-online learning modes — they can be also provided indirectly by indicating the discrepancies between the labels assigned to training instances by the learner’s current hypothesis and the desired correct (target) labels. Supervised learning also includes tasks in which the teacher does not provide training instances, but only answers to the learner’s queries: in such cases the learner specifies a set of instances of the correct labels for which are delivered by the teacher. The key issue in supervised learning is the appropriate generalization of the training information, so that the hypothesis produced by the learner can assign labels to previously unseen instances with high accuracy.

3.3 Unsupervised Learning

The distinguishing feature of unsupervised learning is that no direct or indirect indication of correct labels is given to the learner. It only receives a set of unlabeled training instances and has to come up with a hypothesis based solely on the representation of these instances and possibly some background knowledge. This is the case for clustering tasks, where the learner has to assign distinct discrete labels to groups of training instances determined based on their similarity, and then to generalize this labeling to arbitrary new instances.

3.4 Reinforcement Learning

Unlike in supervised learning, training information does not specify correct labels for training instances (which may be unknown), but unlike in unsupervised learning, they are accompanied with some additional information that guides the learning process. Apart from a set of instances, the learner receives real-valued *rewards*. Instances come from a series of interactions with an *environment* and form a temporal sequence. The source of training information, typically used in online training mode, has the nature of a critic rather than a teacher, as it only evaluates the performance of the current hypothesis without suggesting any way of improving it. In the context of reinforcement learning, instances are usually referred to as states or *situations*, labels — as *actions*, and the mapping from situations to actions is called a *policy*. The learner is required to identify a policy that leads to the maximization of some long-term performance measure based on the sequence of received rewards, which means, in particular, that rewards can represent evaluations of arbitrarily long sequences of preceding actions produced by the learner. The most common discounted reward performance measure is the expected sum of rewards, weighted by subsequent powers of the discount factor $\gamma \in (0, 1)$. The primary issue that must be solved by the learner is *credit assignment*, which consists in assigning credit or blame for received reward values to individual actions.

4 Optimization and Learning: Links and Intersections

As we mentioned in Section 1, machine learning methods have been introduced with a strong emphasis on optimization aspects, and some optimization techniques were treated initially as a kind of learning methods. In this section we discuss these issues in more detail, and we also point to finer relationships between optimization and learning, where these two techniques cooperate together in one algorithm.

4.1 Optimization of Model Predictions

In the computer aided decision support (Sprague and Carlson, 1982) or predictive control systems (Maciejowski, 2002), quite a common approach is to use a certain model of the considered process. The model is used to predict consequences of actions that control the process, and an optimization method is used to find sequence of actions which leads to best results (according to the model's predictions). In this approach, learning the model is made prior to optimization, and we do not consider this combination of optimization and learning as true synergy between these two processes.

4.2 Optimizing Representation for Learning

For many learning tasks it is not the choice of the learning algorithm that has the biggest impact on hypothesis quality. The choice of instance representation is likely to bear a much more substantial effect on the performance of the learned hypothesis. The issue of altering the original propositional representation to another

propositional representation that would be better suited to a particular learning task has received considerable interest in machine learning research and has been studied in at least a few different setups. In particular, for real-valued attributes it can be seen as the issue of finding a transformation from a multidimensional linear space of original attributes to another linear space (usually of different, most often higher) dimensionality. An elegant and powerful approach to efficiently handling such transformations for some types of learning algorithms is provided by *kernel machines* (Schölkopf and Smola, 2002).

Arbitrary propositional representations, regardless of attribute type, can be modified by a *constructive induction* process, also called *feature selection and extraction*. This is essentially a set of transformations which can create new derived attributes based on single or multiple original attributes and select a subset of attributes (original and derived) that appears most promising. Subset selection is a well defined optimization problem if only the objective function for evaluating subsets is specified. Typically, the evaluation is based on the quality of the hypothesis learned when using the evaluated subset. Various techniques have been applied to this problem, including simple local greedy search with attributes being added or deleted from the candidate subset one by one (e.g., Caruana and Freitag, 1994) as well as general global optimization methods (e.g., Yang and Honavar, 1998; Shafti and Pérez, 2005).

4.3 Learning as Optimization

The optimization aspect of learning is quite well recognized, especially when the learner uses a real vector for hypothesis representation. Great effort of researchers has gone into numerical optimization applied for backpropagation neural networks (Hertz *et al.*, 1991), support vector machines (Vapnik, 1998), centroid-based clustering (Kohonen, 2001), and many others. When the representation is based on binary vectors then combinatorial optimization can be applied to perform learning — see e.g., the application of EC to learn decision rules (Goldberg, 1989) or decision forests (Rokach, 2008).

Main problems in formulating learning as an optimization task are the definition of the objective function and of the search space. In our opinion a real intellectual effort is to properly state the learning problem as an optimization task, and the definition of the search method to find the solution is a technical question. In fact, a similar situation can be observed in some areas of operations research or in the domain of logic programming, where a specific solution method (e.g., linear programming or backward-chaining inference) is implemented as a solver, and the problem has to be expressed in a way which is tractable for the solver.

Research efforts in that stream usually go into the direction of formulating learning tasks as convex optimization tasks which allows one to take advantage of good efficiency of optimization methods. It happened however that partial optimization method, which were dedicated for tasks with a single local minimum, were used for problems with multiple local minima (e.g., backpropagation, k-means). With the increase of the available computational power, it has become tempting to improve the quality of solutions by applying optimization methods

which may yield better solutions at the price of computing time. A tendency to substitute local (partial) by global (asymptotically complete) optimization methods is quite well visible in approaches with the real vector representation, e.g., multilayer perceptrons (Žurada *et al.*, 1994) or support vector machines (Bennett and Parrado-Hernandez, 2006).

A possible concern against applying global optimization to learning is related to the fear of overfitting. This concern is misplaced, however, since the risk of overfitting is affected by what is optimized (the objective function used to evaluate candidate hypotheses and the space of possible hypotheses) and not by how it is optimized (the optimization method). This problem needs to be addressed regardless of the optimization method used. Discussing possible approaches, which include structural risk minimization (Vapnik, 1991), Bayesian information criterion (Schwarz, 1978), and minimum message encoding (Oliver and Hand, 1994), along with other more or less formal interpretations of Ockham’s razor (Domingos, 1999), is beyond the scope of this article.

Whereas any learning process can be viewed as search in the space of possible knowledge representations, reinforcement learning is related to search also in an additional and more direct way. The learning scenario, which is a series of interactions between the learner and its environment, closely resembles the search process. Environment situations can be considered analogs of search space points, actions are a way of performing variation, and the action selection policy (being learnt) represents the selection operation. Reinforcement learning can be seen as most directly related to a particular form of search referred to as real-time heuristic search (Korf, 1990). Real-time refers to some assumptions limiting the possible ways of navigating in the search space. In particular, the search state is limited to a single search space point, a limited number of new points can be generated by applying the variation operation, and one of them has to be selected as the new state by the selection operation. The relationships between real-time heuristic search and reinforcement learning have been emphasized by Barto *et al.* (1995).

4.4 Optimization as Learning

In online supervised learning, the learner adapts the labels assigned to training instances to the requirements of the teacher specified via training information. Assuming this perspective we can say that each optimization method is an instantiation of a learning task. Recall that an optimization method is described by its aggregated operator $O : \Pi \times V \rightarrow X^*$. When we focus on a particular problem, the operator is a mapping $X^* \times U^* \rightarrow X^*$, so it defines a sequence of points to generate basing on points that have been generated so far. If the objective function is to be minimized, it can be interpreted as an error measure, and the task to be learnt is the point where the objective function (the error function) has its minimum. A good example for this philosophy is the artificial immune system inspired optimization (Wierzchoń, 2001).

Learning can also be used as a substitute for optimization. If the objective function f is computationally expensive or the optimization problem is hard, then, instead of optimizing f directly to find the solution, a somehow related learning problem is defined where the error function has its minimum in the minimum

of the objective function f . An example for such technique can be found for the Traveling Salesman Problem when a graph is Euclidean. Kohonen (2001) presents an approximate solution to the TSP with the use of a Self-Organizing Map (SOM). Geographical positions of cities are presented to the SOM which has a linear topology, and the best mapping is located. Quite usually, the best mapping means the cheapest (or nearly cheapest) salesman's tour.

4.5 Learning What to Optimize

Not infrequently the objective function is computationally expensive in solving practical optimization problems. Therefore some practitioners define a function $\varphi : X \rightarrow R$ which approximates the objective function f using the sequence of points generated so far. The function φ is then minimized.

This methodology can be found in various approaches which differ in algorithmic details. In the *Response Surface Method* (RSM), the function φ is learnt from the historical data. Then a minimum of function φ is located, and this is the solution yielded by the method. This methodology assumes no interaction between the optimization method and function φ .

A *sequential RSM* procedure (e.g., Booker *et al.*, 1999) is a version of the RSM which iterates the process of learning the function φ and finding its minimum. In each iteration, when the function φ has been learnt and a point x which minimizes φ has been found, the original objective function value $f(x)$ is computed. Then the function φ is learnt again with the updated set of data, and the process is repeated.

Function φ can be a global or only a local approximation of f . In the latter case, the approximation is made in a neighborhood of a point to be varied, and the local approximation is made every time a variation is performed. A potential advantage of this local approximation strategy, in contrast to the global approximation version of RSM, is that local minima of functions φ and f may be close to each other, which makes it possible to use precise local optimization methods.

Kriging (Sacks *et al.*, 1989) is yet another technique that extends the idea of the sequential RSM. Similarly to the sequential RSM, a global approximation to the objective function is learnt whenever a new point is generated. The approximation function has a structure $\varphi(x) = \sum_{i=1}^n \beta_i h_i(x) + Z(x) + \varepsilon$, where h_i are basis functions, and β_i are their coefficients, $Z(x)$ is the systematic error, and ε represents the nonsystematic error. The basis functions can be, e.g., polynomials or Gaussian functions. ε is assumed to be a zero mean, uncorrelated random variable, and $Z(x)$ is the realization of a stationary Gaussian stochastic process.

Note that in the case of both kriging and sequential RSM, the key issue is to learn to approximate well the objective function, and therefore it is needed to repeat the learning many times during the single run of the optimization method. If the ultimate goal is still to reduce the computational effort, learning procedures must be simple enough to require only small amount of extra computations.

4.6 Learning How to Perform Variation

In the RSM and kriging, function φ is used to predict values of the objective function. When the variation of the search method is stochastic, then, instead of the approximation to the objective function, a distribution of a random variable can be learnt. This distribution, called the *sampling distribution* (SD), expresses the possibility of finding the best solution in a certain area of the search space. SD is updated each time when the objective function is evaluated in a certain point from the search space. Thus the method defines a sequence of SDs which are used to define the stochastic variation operator.

SD is used to draw some number of points, and after that, it is updated so that the objective function values in these points are taken into account. A simple update rule is to increase probability of selecting subsets of A where points with better quality are more frequently met. Application of such strategy can be found, e.g., in Bayesian global optimization (BGO, Törn and Žilinskas, 1989) or in estimation of distribution algorithms (EDA, Mühlenbein and Paass, 1996), where SD is represented as a linear combination of base distributions (e.g., a Gaussian mixture).

Yet another possibility to perform adaptation can be met in evolutionary computation. Since in EC the method's state contains a population of points from the search space, the variation operator is usually implemented as a set of variations of single points (mutation) or pairs of points (crossover). In the standard version of EC, mutation consists in adding a random value to the mutated point, and the distribution of this random variable is independent of the mutated point. Non-standard approaches include an adaptation scheme for parameters. For example, instead of a single distribution to change the mutated point, a variety of distributions can be defined. Each version of the mutation operator can be assigned a probability to be selected and applied, and the probabilities are updated according to some update rule (e.g., a bucket brigade — Julstrom, 1995).

4.7 Substituting Variation by Learning

Effectiveness in precise finding local minima of the objective function differs significantly among optimization methods. Especially methods which are asymptotically complete are known to be ineffective in that task. Therefore there is a tendency to combine methods which are asymptotically complete with partial methods, which in turn perform well in local optimization. Since optimization and learning can be interpreted dually, it is possible to make a hybrid of an asymptotically complete method with a certain learning algorithm. A good example is a *multistart method* (Törn and Žilinskas, 1989) where many solutions are generated starting from different initial states of the optimization method. The initial states are generated randomly, usually with a uniform distribution in the search space, or deterministically in a certain systematical way which uniformly covers the search space (Tezuka, 1995).

The multistart method can be improved by nonuniform generation of initial states, and the generator can be an asymptotically complete method. Coming back to the example of evolutionary methods, such kind of hybridization can be

made by applying the learning or local search procedure each time when the objective function value has to be computed in a certain point (Bäck *et al.*, 1996). The evaluated point can serve as the initial state of the local procedure, and the point which is returned by the local procedure either substitutes the initial one (Lamarckian evolution), or is used to modify the selection operator (Darwinian version).

When the variation is adapted (Section 4.6), a learning procedure changes the variation in an indirect way. A direct application of the learning procedure as the variation operator has also been tested. The algorithmic framework of this methodology when the selection is applied according to the evolutionary method, was called the Learnable Evolution Model (LEM, Michalski, 2000). There are reports on applications of this methodology to data mining (the AQ method is used for the learning task). Another approach that fits into the LEM framework is the Evolutionary Gradient Search (Arnold and Salomon, 2007) where the role of a learning procedure is played by a gradient based local minimization technique.

4.8 Learning How to Perform Selection

The way of performing selection in search methods usually affects considerably their computational efficiency, and may also impact the quality of achievable solutions. It is also a natural place to “inject” domain-specific knowledge and make the search process more informed.

4.8.1 Learning to Reduce the State Size

If the state of an optimization method includes more than a single point then it is possible to apply certain data analysis techniques to describe relations between positions of these points in the search space. This relation can be a source of some additional knowledge and can be used to control the search process. A family of algorithms called the *Clustering Method* (CM — Törn and Žilinskas, 1989) has been suggested to perform numerical optimization in R^n . The method is inspired by the observation that if a local optimization method is applied with any starting point, then it will eventually converge to a local minimum of the objective function.

All points of the search space for which a local optimization method, when started from them, will converge to the same point, are called the *attraction basin* of that point. Observe that if we start the local optimization method twice for two different starting points, and it generates sequences x^* and y^* that converge to the same point, then the distance between two points $x_j \in x^*$ and $y_j \in y^*$ will decrease with j increasing. So, if a local optimization method has been started for a population of different starting points, then, waiting sufficiently long, we will observe that points from the generated sequences tend to form clusters in the search space. Therefore we can reject some number of sequences which will converge to the same local minimum and thus avoid unnecessary computations. In the CM method, prior to defining a state, clustering is performed to points generated in the previous iteration, and the state contains only points which differ in the cluster label. Thus the number of generated points decreases over time.

4.8.2 Learning Heuristic Functions

For optimization tasks where the search space contains (in several cases, mostly) infeasible solutions it may be unclear how to define the objective functions for infeasible solutions so that the applied search method is well guided towards good quality feasible solutions. An improperly defined objective function will lead to inefficient search at best, and may even prevent the search method from finding a good solution at all. In particular, if the objective function is specified using a heuristic function (or a penalty function), as discussed in Section 2.4, defining such a function may require in-depth understanding of the task and several trial-and-error attempts. The idea of having heuristic functions derived or refined by a learning process is therefore interesting and potentially very useful. It is based on the assumption that the same heuristic function can be useful for a family of related optimization tasks sharing the same objective function, but with different starting points or even different (although similarly structured) search spaces. The experience gathered during solving some specific task instances could be therefore used to improve or replace the initial heuristic function, leading to better performance for subsequent task instances.

Assuming the setup introduced in Section 2.4, we need to derive a heuristic function $h(x)$ that represents the difference between the lower bound of the objective function for any feasible solution accessible by subsequent variation of x and the lower bound of the objective function for the best feasible solution that can be reached from x . The task of learning h can be formulated as a supervised or reinforcement learning task. The former formulation is fairly straightforward. Each solved task instance provides a training instance for every point visited during the search process that led to the solution. We can calculate the target function value for a visited point x that was encountered during the search that finally reached solution y as the difference $f(y) - g(x)$. One problem with this approach is that, when y is not the best feasible solution that can be reached from x , the resulting learned heuristic function may be overestimating and therefore inadmissible for A*-like search methods. A possible remedy is to scale down either target values for training examples or the output of the model for h to preserve non-overestimation constraints.

The reinforcement learning approach is slightly less obvious, but it may offer some advantages. Consider a reinforcement learner interacting with an environment which represents an optimization task, with situations corresponding to search space points and actions performing the variation operation. The reinforcement function can be defined as the objective function for situations corresponding to feasible solutions and a constant $-\epsilon$ for all other situations, where $\epsilon \geq 0$. The value of a situation x should therefore converge to the expected reinforcement that will be received along the path from x to a situation y corresponding to a feasible solution. Assuming the most typical discounted reinforcement learning setup, this can be expressed as $V(x) \rightarrow \mathbf{E} \left[-\epsilon \sum_{k=0}^{n-1} \gamma^k + \gamma^n f(y) \right]$, where n is the path length (i.e., the number of situations visited starting from x before arriving at y) and $\gamma \in (0, 1]$ is the (optional) discount factor. If $\epsilon = 0$, which corresponds to a zero cost of generating points, and $\gamma = 1$, which applies no discounting, the above reduces to $f(y)$. The value $V(x)$ learned that way corresponds to the sum

$g(x) + h(x)$ used for guiding the search process of A*-like methods.

Note that, unless $\gamma = 1$ and $\epsilon = 0$, the value of situation x will depend not only on the quality of the achieved solution, but also on the length of the path in the search space leading to this solution. This is usually desired for reinforcement learning to trade off between the solution quality and the time needed to find it. Both the quality of the solution and the corresponding path length depend on the selection method used, which decides which of the possible actions generating new points to apply in a given situation. If reinforcement learning is used for an optimization task, the variation operation will be usually applied to generate a number of points (corresponding to different actions of the reinforcement learner) and one of them will be selected to retain. The selection method is based on the current value function and is responsible for handling the exploration vs. exploitation tradeoff, preferring maximum-value points most of the time, but making it possible to occasionally select low-value points as well. The processes of learning the value function and applying it for selection during search are therefore interleaving, which is a typical feature of reinforcement learning. This may be a substantial advantage over the supervised learning approach to heuristic function learning for tasks where no reasonable initial heuristic function can be defined and therefore solving a sufficient number of task instances to generate training examples would be computationally prohibitive. The reinforcement learning approach can start with an arbitrary initial heuristic function and start to improve it as soon as the first feasible solution is reached.

The idea of acquiring and refining search heuristics through reinforcement learning was first explored by Samuel (1959), many years before the term reinforcement learning was coined. It has been successfully applied to learn search heuristics for two-player games (Tesauro, 1992; Schraudolph *et al.*, 1994) and combinatorial optimization problems (Zhang and Dietterich, 1995).

4.8.3 Learning Heuristic Selection Rules

The selection operation can be performed by applying a set of rules to candidate points rather than by numeric evaluations. When variation makes it possible to generate a number of new points based on the current search state, such heuristic selection rules would select a subset of them. Sometimes this can be actually preselection, i.e., the decision can be made before applying the variation operation, based only on its properties and the current state. A typical example would be the variation operation composed of several elementary operators, each capable of generating a single new point. The current state may allow to select a subset of “most promising” elementary operators.

The task of learning heuristic selection or preselection rules can be naturally formulated as a multi-concept classification learning task in which a separate classifier is assigned to each elementary operator, using some state descriptions as input instances. A well-known example of this approach is the work of Mitchell *et al.* (1983) on symbolic integration. A closely related approach was adopted by Langley (1983). An important design decision associated with this learning task is the proper selection of training instances and their labeling as positive and negative examples of applying a given elementary operator in a given state. This

can be thought of as a “binary version” of the credit assignment problem in reinforcement learning. Typically, a number of task instances have to be solved for this purpose and some or all operator applications labeled as positive or negative depending on whether they belong to a shortest path to a good solution or not.

5 Conclusions

The ability to make reasonable decisions based on existing knowledge and to acquire or improve knowledge to make better decisions in the future is at heart of intelligent behavior that we experience with natural systems and seek to achieve with artificial systems. The essential role of learning and optimization among artificial intelligence tasks has been widely recognized since the very beginning of this field (Minsky, 1961). While it remains unquestionable that both optimization and learning are essential and needed — sometimes simultaneously — to develop intelligent systems, the relationships between these two that have emerged over the several past decades of their development deserve more serious interest than they have received so far.

Through an incomplete but systemized overview of several links and intersections between optimization and learning we have shown two apparently contradictory facts: that they have more in common and that they can benefit from each other more than is usually realized. The contradiction is indeed only apparent since it is the common core of optimization and learning methods that makes them both closely related and mutually useful. This common core is best characterized in terms of search, which can be called — with a bit of bombast, but without exaggeration — the mother of all artificial intelligence tasks.

We believe that understanding how learning and optimization can be viewed as two related ways of searching for intelligent behavior has something more to offer than a marginally elegant key to the AI literature. It can facilitate discovering new opportunities of mutual boosting between optimization and learning, i.e., developing computational techniques that combine one with the other, apply one to the other, or even replace one by the other in novel ways. These techniques may better exploit the hidden potential that this article has been trying to partially unhide.

References

- D. ANGLUIN (1987), Learning Regular Sets from Queries and Counterexamples, *Information and Computation*, 75:87–106.
- D. V. ARNOLD and R. SALOMON (2007), Evolutionary Gradient Search Revisited, *IEEE Trans. on Evolutionary Computation*, 11.
- T. BÄCK, D. FOGEL, and Z. MICHALEWICZ, editors (1996), *Handbook of Evolutionary Computation*, Oxford University Press, New York.
- T. BÄCK and H.-P. SCHWEFEL (1993), An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation*, 1:1–23.
- A. G. BARTO, S. J. BRADTKE, and S. P. SINGH (1995), Learning to Act Using Real-Time Dynamic Programming, *Artificial Intelligence*, 72:81–138.

- K. P. BENNETT and E. PARRADO-HERNANDEZ (2006), The Interplay of Optimization and Machine Learning Research, *J. of Machine Learning Research*, 7:1265–1281.
- A. J. BOOKER, J. E. DENNIS, Paul D. FRANK, David B. SERAFINI, V. TORCZON, and M. W. TROSSET (1999), A Rigorous Framework for Optimization of Expensive Functions by Surrogates, *Structural and Multidisciplinary Optimization*, 17:1–13.
- L. BREIMAN, J. H. FRIEDMAN, R. A. OLSHEN, and P. J. STONE (1984), *Classification and Regression Trees*, Wadsworth International Group.
- J. G. CARBONELL, R. S. MICHALSKI, and T. M. MITCHELL (1983), An Overview of Machine Learning, in R. S. MICHALSKI, J. G. CARBONELL, and T. M. MITCHELL, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, Tioga (now Morgan Kaufmann).
- R. CARUANA and D. FREITAG (1994), Greedy Attribute Selection, in *Proc. 11th Int. Conf. on Machine Learning (ML-94)*, Morgan Kaufmann.
- P. CLARK and T. NIBLETT (1989), The CN2 Induction Algorithm, *Machine Learning*, 3:261–283.
- W. W. COHEN (1995), Fast Effective Rule Induction, in *Proc. 12th Int. Conf. on Machine Learning (ML-95)*, Morgan Kaufmann.
- T. DEAN, D. ANGLUIN, K. BASYE, S. ENGELSON, L. P. KAEHLING, E. KOKKEVIS, and O. MARON (1995), Inferring Finite Automata with Stochastic Output Functions and an Application to Map Learning, *Machine Learning*, 18:81–108.
- T. G. DIETTERICH (1997), Machine Learning: Four Current Directions, *AI Magazine*, 18:97–136.
- P. DOMINGOS (1999), The Role of Occam’s Razor in Knowledge Discovery, *Data Mining and Knowledge Discovery*, 3:409–425.
- D.-Z. DU and P. PARDALOS (1998), *Handbook of Combinatorial Optimization*, Springer.
- L. FOGEL, A. OWENS, and M. WALSH (1966), *Artificial Intelligence through Simulated Evolution*, Wiley.
- F. GLOVER and G. KOCHENBERGER, editors (2003), *Handbook of Metaheuristics*, Springer.
- D. GOLDBERG (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Wiley.
- J. HERTZ, A. KROGH, and R. PALMER (1991), *Introduction to the Theory of Neural Computation*, Addison-Wesley.
- J. H. HOLLAND (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.
- R. HORST and P. PARDALOS (1995), *Handbook of Global Optimization*, Kluwer.
- Bryant A. JULSTROM (1995), What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm, in *ICGA95*, pp. 81–87.
- L. P. KAEHLING, M. L. LITTMAN, and A. W. MOORE (1996), Reinforcement Learning: A Survey, *J. of Artificial Intelligence Research*, 4:237–285.
- T. KOHONEN (2001), *Self-Organizing Maps*, Springer.
- R. E. KORF (1990), Real-Time Heuristic Search, *Artificial Intelligence*, 42:189–211.
- J. KOZA (1992), *Genetic Programming*, MIT Press, Cambridge.
- M. KUBAT, I. BRATKO, and R. S. MICHALSKI (1998), A Review of Machine Learning Methods, in *Machine Learning and Data Mining: Methods and Applications*, Wiley.
- P. LANGLEY (1983), Learning Effective Search Heuristics, in *Proc. 8th Int. Joint Conf. on Artificial Intelligence (IJCAI-83)*, Morgan Kaufmann.

- J. M. MACIEJOWSKI (2002), *Predictive Control with Constraints*, Prentice Hall.
- Z. MICHAŁEWICZ (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer.
- R. MICHAŁSKI, I. BRATKO, and M. KUBAT (1998), *Machine Learning and Data Mining*, Wiley.
- R. S. MICHAŁSKI (1973), AQVAL/1 — Computer Implementation of a Variable Valued Logic VL1 and Examples of Its Application to Pattern Recognition, in *Proc. 1st Int. Joint Conf. on Pattern Recognition*.
- R. S. MICHAŁSKI (1994), Inferential Theory of Learning: Developing Foundations For Multistrategy Learning, in R. S. MICHAŁSKI and G. TECUCI, editors, *Machine Learning: A Multistrategy Approach*, volume 4, Morgan Kaufmann.
- R. S. MICHAŁSKI (2000), LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning, *Machine Learning*, 38:9–40.
- R. S. MICHAŁSKI, I. MOZETIC, J. HONG, and N. LAVRAC (1986), The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains, in *Proc. 5th Nat. Conf. on Artificial Intelligence (AAAI-86)*, MIT Press.
- M. L. MINSKY (1961), Steps toward Artificial Intelligence, in *Proc. of the Institute of Radio Engineers*, reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.
- T. M. MITCHELL (1982), Generalization as Search, *Artificial Intelligence*, 18:203–226.
- T. M. MITCHELL, R. M. KELLER, and S. T. KEDAR-CABELLI (1986), Explanation-Based Generalization: A Unifying View, *Machine Learning*, 1:47–80.
- T. M. MITCHELL, P. E. UTGOFF, and R. BANERJI (1983), Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics, in R. S. MICHAŁSKI, J. G. CARBONELL, and T. M. MITCHELL, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, Tioga (now Morgan Kaufmann).
- S. MUGGLETON, editor (1992), *Inductive Logic Programming*, Academic Press.
- H. MÜHLENBEIN and G. PAASS (1996), From Recombination of Genes to the Estimation of Distributions I. Binary Parameters, in *PPSN'96*, pp. 178–187.
- J. J. OLIVER and D. J. HAND (1994), Introduction to Minimum Encoding Inference, Tech. Rep. 205, Department of Computer Science, Monash University, Australia.
- M. J. PAZZANI and D. KIBLER (1992), The Utility of Prior Knowledge in Inductive Learning, *Machine Learning*, 9:54–97.
- E. POLAK (1997), *Optimization: Algorithms and Consistent Approximation*, Springer.
- J. R. QUINLAN (1990), Learning Logical Definitions from Relations, *Machine Learning*, 5:239–266.
- J. R. QUINLAN (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann.
- I. RECHENBERG (1994), *Evolution Strategies 94*, Fromman Holzboog.
- L. ROKACH (2008), An Evolutionary Algorithm for Constructing a Decision Forest: Combining the Classification of Disjoint Decision Trees, *Int. J. of Intelligent Systems*, 23:455–482.
- S. RUSSEL and P. NORVIG (1990), *Artificial Intelligence: A Modern Approach*, Prentice Hall.
- J. SACKS, W. J. WELCH, T. J. MITCHELL, and H.P. WYNN (1989), Design and Analysis of Computer Experiments (with discussion), *Statistical Science*, 4:409–435.

- A. L. SAMUEL (1959), Some Studies in Machine Learning Using the Game of Checkers, *IBM J. on Research and Development*, 3:210–229, reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.
- B. SCHÖLKOPF and A. SMOLA (2002), *Learning with Kernels: Support Vector Machines, Regularization, and Beyond*, MIT Press.
- N. N. SCHRAUDOLPH, P. DAYAN, and T. J. SEJNOWSKI (1994), Temporal Difference Learning of Position Evaluation in the Game of Go, in *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann.
- G. SCHWARZ (1978), Estimating the Dimension of a Model, *Annals of Statistics*, 6:461–464.
- L. S. SHAFTI and E. PÉREZ (2005), Constructive Induction and Genetic Algorithms for Learning Concepts with Complex Interaction, in *GECCO-2005*, ACM.
- R. H. SPRAGUE and E. D. CARLSON (1982), *Building Effective Decision Support Systems*, Prentice Hall.
- R. S. SUTTON and A. G. BARTO (1998), *Reinforcement Learning: An Introduction*, MIT Press.
- G. TESAURO (1992), Practical Issues in Temporal Difference Learning, *Machine Learning*, 8:257–277.
- S. TEZUKA (1995), *Uniform Random Numbers: Theory and Practice*, Kluwer.
- A. TÖRN and A. ŽILINSKAS (1989), *Global Optimization*, Springer.
- V. VAPNIK (1991), Principles of Risk Minimization for Learning Theory, in *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann.
- V. VAPNIK (1998), *Statistical Learning Theory*, Wiley.
- S.T. WIERZCHOŃ (2001), *Artificial Immune Systems. Theory and Applications (in Polish)*, EXIT, Warszawa.
- J. YANG and V. HONAVAR (1998), Feature Subset Selection Using a Genetic Algorithm, *IEEE Intelligent Systems*, 13:44–49.
- W. ZHANG and T. G. DIETTERICH (1995), A Reinforcement Learning Approach to Job-Shop Scheduling, in *Proc. 14th Int. J. Conf. on Artificial Intelligence (IJCAI-95)*, Morgan Kaufmann.
- J. ŻURADA, R. MARKS, and C. ROBINSON, editors (1994), *Computational Intelligence: Imitating Life*, IEEE Press.