evaluating which of several alternative algorithms is best suited to a specific application.

Processes and Techniques

Many learning algorithms have been proposed. In order to understand the relative merits of these alternatives, it is necessary to evaluate them. The primary approaches to evaluation can be characterized as either theoretical or experimental. Theoretical evaluation uses formal methods to infer properties of the algorithm, such as its computational complexity (Papadimitriou, 1994), and also employs the tools of ►computational learning theory to assess learning theoretic properties. Experimental evaluation applies the algorithm to learning tasks to study its performance in practice.

There are many different types of property that may be relevant to assess depending upon the intended application. These include algorithmic properties, such as time and space complexity. These algorithmic properties are often assessed separately with respect to performance when learning a ►model, that is, at ►training time, and performance when applying a learned model, that is, at ►test time.

Other types of property that are often studied are the properties of the models that are learned (see ►model evaluation). Strictly speaking, such properties should be assessed with respect to a specific application or class of applications. However, much machine learning research includes experimental studies in which algorithms are compared using a set of data sets with little or no consideration given to what class of applications those data sets might represent. It is dangerous to draw general conclusions about relative performance on any application from relative performance on this sample of some unknown class of applications. Such experimental evaluation has become known disparagingly as a *bake-off*.

An approach to experimental evaluation that may be less subject to the limitations of bake-offs is the use of experimental evaluation to assess a learning algorithm's ►bias and variance profile. Bias and variance measure properties of an algorithm's propensities in learning models rather than directly being properties of the models that are learned. Hence, they may provide more general insights into the relative characteristics of alternative algorithms than do assessments of the performance of learned models on a finite number of applications. One example of such use of bias–variance analysis is found in Webb (2000).

Techniques for experimental algorithm evaluation include ►bootstrap sampling, ►cross-validation, and ►holdout evaluation.

Cross References

►Computational Learning Theory
►Model Evaluation

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning*. New York: Springer.
Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.
Papadimitriou, C. H. (1994). *Computational complexity*. Reading, MA: Addison-Wesley.
Webb, G. I. (2000). MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, *40*(2), 159–196.
Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Analogical Reasoning

►Instance-Based Learning

Analysis of Text

►Text Mining

Analytical Learning

►Deductive Learning
►Explanation-Based Learning

Ant Colony Optimization

Marco Dorigo, Mauro Birattari
Université Libre de Bruxelles, Brussels, Belgium

Synonyms

ACO

Definition

Ant colony optimization (ACO) is a population-based metaheuristic for the solution of difficult combinatorial

optimization problems. In ACO, each individual of the population is an artificial agent that builds incrementally and stochastically a solution to the considered problem. Agents build solutions by moving on a graph-based representation of the problem. At each step their moves define which solution components are added to the solution under construction. A probabilistic model is associated with the graph and is used to bias the agents' choices. The probabilistic model is updated online by the agents so as to increase the probability that future agents will build good solutions.

Motivation and Background

Ant colony optimization is so called because of its original inspiration: the foraging behavior of some ant species. In particular, in Beckers, Deneubourg, and Goss (1992) it was demonstrated experimentally that ants are able to find the shortest path between their nest and a food source by collectively exploiting the pheromone they deposit on the ground while walking. Similar to real ants, ACO's artificial agents, also called artificial ants, deposit artificial pheromone on the graph of the problem they are solving. The amount of pheromone each artificial ant deposits is proportional to the quality of the solution the artificial ant has built. These artificial pheromones are used to implement a probabilistic model that is exploited by the artificial ants to make decisions during their solution construction activity.

Structure of the Optimization System

Let us consider a minimization problem $(\mathcal{S}, f)$, where $\mathcal{S}$ is the *set of feasible solutions*, and f is the *objective function*, which assigns to each solution $s \in \mathcal{S}$ a cost value $f(s)$. The goal is to find an optimal solution s^*, that is, a feasible solution of minimum cost. The set of all optimal solutions is denoted by $\mathcal{S}^*$.

Ant colony optimization attempts to solve this minimization problem by repeating the following two steps:

- Candidate solutions are constructed using a parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
- The candidate solutions are used to modify the model in a way that is intended to bias future sampling toward low cost solutions.

The Ant Colony Optimization Probabilistic Model

We assume that the combinatorial optimization problem $(\mathcal{S}, f)$ is mapped on a problem that can be characterized by the following list of items:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_C}\}$ of *components*, where N_C is the number of components.
- A finite set $\mathcal{X}$ of *states* of the problem, where a state is a sequence $x = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$ over the elements of $\mathcal{C}$. The length of a sequence x, that is, the number of components in the sequence, is expressed by $|x|$. The maximum length of a sequence is bounded by a positive constant $n < +\infty$.
- A set of (candidate) solutions $\mathcal{S}$, which is a subset of $\mathcal{X}$ (i.e., $\mathcal{S} \subseteq \mathcal{X}$).
- A set of feasible states $\tilde{\mathcal{X}}$, with $\tilde{\mathcal{X}} \subseteq \mathcal{X}$, defined via a set of *constraints* Ω.
- A nonempty set $\mathcal{S}^*$ of optimal solutions, with $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ and $\mathcal{S}^* \subseteq \mathcal{S}$.

Given the above formulation (Note that, because this formulation is always possible, ACO can in principle be applied to any combinatorial optimization problem.) artificial ants build candidate solutions by performing randomized walks on the completely connected, weighted graph $\mathcal{G} = (\mathcal{C}, \mathcal{L}, \mathcal{T})$, where the vertices are the components $\mathcal{C}$, the set $\mathcal{L}$ fully connects the components $\mathcal{C}$, and $\mathcal{T}$ is a vector of so-called *pheromone trails* τ. Pheromone trails can be associated with components, connections, or both. Here we assume that the pheromone trails are associated with connections, so that $\tau(i,j)$ is the pheromone associated with the connection between components i and j. It is straightforward to extend the algorithm to the other cases. The graph $\mathcal{G}$ is called the *construction graph*.

To construct candidate solutions, each artificial ant is first put on a randomly chosen vertex of the graph. It then performs a randomized walk by moving at each step from vertex to vertex on the graph in such a way that the next vertex is chosen stochastically according to the strength of the pheromone currently on the arcs.

While moving from one node to another of the graph $\mathcal{G}$, constraints Ω may be used to prevent ants from building infeasible solutions. Formally, the solution construction behavior of a generic ant can be described as follows:

ANT_SOLUTION_CONSTRUCTION

- For each ant:
  - Select a start node c_1 according to some problem dependent criterion.
  - Set $k = 1$ and $x_k = \langle c_1 \rangle$.
- While $x_k = \langle c_1, c_2, \ldots, c_k \rangle \in \tilde{\mathcal{X}}, x_k \notin \mathcal{S}$, and the set J_{x_k} of components that can be appended to x_k is not empty, select the next node (component) c_{k+1} randomly according to:

$$P_{\mathcal{T}}(c_{k+1} = c | x_k) = \begin{cases} \dfrac{F_{(c_k,c)}(\tau(c_k,c))}{\sum_{(c_k,y)\in J_{x_k}} F_{(c_k,y)}(\tau(c_k,y))} & \text{if } (c_k,c) \in J_{x_k}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where a connection (c_k, y) belongs to J_{x_k} if and only if the sequence $x_{k+1} = \langle c_1, c_2, \ldots, c_k, y \rangle$ satisfies the constraints Ω (that is, $x_{k+1} \in \tilde{\mathcal{X}}$) and $F_{(i,j)}(z)$ is some monotonic function – a common choice being $z^{\alpha}\eta(i,j)^{\beta}$, where $\alpha, \beta > 0$, and $\eta(i,j)$'s are heuristic values measuring the desirability of adding component j after i. If at some stage $x_k \notin \mathcal{S}$ and $J_{x_k} = \emptyset$, that is, the construction process has reached a dead-end, the current state x_k is discarded. However, this situation may be prevented by allowing artificial ants to build infeasible solutions as well. In such a case, an infeasibility penalty term is usually added to the cost function. Nevertheless, in most of the settings in which ACO has been applied, the dead-end situation does not occur.

For certain problems, one may find it useful to use a more general scheme, where F depends on the pheromone values of several "related" connections rather than just a single one. Moreover, instead of the *random-proportional rule* above, different selection schemes, such as the *pseudo-random-proportional rule* (Dorigo & Gambardella, 1997), may be used.

The Ant Colony Optimization Pheromone Update

Many different schemes for pheromone update have been proposed within the ACO framework. For an extensive overview, see Dorigo and Stützle (2004). Most pheromone updates can be described using the following generic scheme:

GENERIC_ACO_UPDATE

- $\forall s \in \hat{S}_t, \forall (i,j) \in s: \ \tau(i,j) \leftarrow \tau(i,j) + Q_f(s|S_1, \ldots, S_t)$,
- $\forall (i,j): \ \tau(i,j) \leftarrow (1-\rho) \cdot \tau(i,j)$,

where S_i is the sample in the ith iteration, ρ, $0 \le \rho < 1$, is the evaporation rate, and $Q_f(s|S_1, \ldots, S_t)$ is some "quality function," which is typically required to be non-increasing with respect to f and is defined over the "reference set" $\hat{S}_t$.

Different ACO algorithms may use different quality functions and reference sets. For example, in the very first ACO algorithm – Ant System (Dorigo, Maniezzo, & Colorni, 1991, 1996) – the quality function is simply $1/f(s)$ and the reference set $\hat{S}_t = S_t$. In a subsequently proposed scheme, called *iteration best update* (Dorigo & Gambardella, 1997), the reference set is a singleton containing the best solution within S_t (if there are several iteration-best solutions, one of them is chosen randomly). For the *global-best update* (Dorigo et al., 1996; Stützle & Hoos, 1997), the reference set contains the best among all the iteration-best solutions (and if there are more than one global-best solution, the earliest one is chosen). In Dorigo et al. (1996) an *elitist* strategy was introduced, in which the update is a combination of the previous two.

In case a good lower bound on the optimal solution cost is available, one may use the following quality function (Maniezzo, 1999):

$$Q_f(s|S_1, \ldots, S_t) = \tau_0 \left(1 - \frac{f(s) - \mathrm{LB}}{\bar{f} - \mathrm{LB}}\right) = \tau_0 \frac{\bar{f} - f(s)}{\bar{f} - \mathrm{LB}}, \tag{2}$$

where $\bar{f}$ is the average of the costs of the last k solutions and LB is the lower bound on the optimal solution cost. With this quality function, the solutions are evaluated by comparing their cost to the average cost of the other recent solutions, rather than by using the absolute cost values. In addition, the quality function is automatically scaled based on the proximity of the average cost to the lower bound.

A pheromone update that slightly differs from the generic update described above was used in *ant colony system* (ACS) (Dorigo & Gambardella, 1997). There the pheromone is evaporated by the ants online during the solution construction, hence only the pheromone involved in the construction evaporates.

Another modification of the generic update was introduced in $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (Stützle & Hoos, 1997, 2000), which uses maximum and minimum pheromone trail limits. With this modification, the probability of generating any particular solution is kept above some positive threshold. This helps to prevent search stagnation and premature convergence to suboptimal solutions.

Cross References

►Swarm Intelligence

Recommended Reading

Beckers, R., Deneubourg, J. L., & Goss, S. (1992). Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology, 159*, 397–415.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

Dorigo M., Maniezzo V., & Colorni A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26*(1), 29–41.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge, MA: MIT Press.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing, 11*(4), 358–369.

Stützle, T., & Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In *Proceedings of the 1997 Congress on Evolutionary Computation – CEC'97* (pp. 309–314). Piscataway, NJ: IEEE Press.

Stützle, T., & Hoos, H. H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generation Computer Systems, 16*(8), 889–914, 2000.

Anytime Algorithm

An *anytime algorithm* is an algorithm whose output increases in quality gradually with increased running time. This is in contrast to algorithms that produce no output at all until they produce full-quality output after a sufficiently long execution time. An example of an algorithm with good anytime performance is ►Adaptive Real-Time Dynamic Programming (ARTDP).

AODE

►Averaged One-Dependence Estimators

Apprenticeship Learning

►Behavioral Cloning

Approximate Dynamic Programming

►Value Function Approximation

Apriori Algorithm

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996) is a ►data mining method which outputs all ►frequent itemsets and ►association rules from given data.

Input: set $\mathcal{I}$ of items, multiset $\mathcal{D}$ of subsets of $\mathcal{I}$, frequency threshold *min_fr*, and confidence threshold *min_conf*.

Output: all frequent itemsets and all valid association rules in $\mathcal{D}$.

Method:

1: level := 1; frequent_sets := $\emptyset$;
2: candidate_sets := $\{\{i\} \mid i \in \mathcal{I}\}$;
3: while candidate_sets $\neq \emptyset$
3.1: scan data $\mathcal{D}$ to compute frequencies of all sets in candidate_sets;
3.2: frequent_sets := frequent_sets $\cup$ $\{C \in$ candidate_sets $\mid$ frequency$(C) \geq$ *min_fr*$\}$;
3.3 level := level + 1;
3.4: candidate_sets := $\{A \subset \mathcal{I} \mid |A| =$ level and $B \in$ frequent_sets for all $B \subset A, |B| =$ level $- 1\}$;

Meta-Combiner

A *meta-combiner* is a form of ►ensemble learning technique used with ►missing attribute values. Its common topology involves base learners and classifiers at the first level, and meta-learner and meta-classifier at the second level. The meta-classifier combines the decisions of all the base classifiers.

Metaheuristic

Marco Dorigo, Mauro Birattari,
Thomas Stützle

A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework that can be applied to different optimization problems with relatively few modifications. Examples of metaheuristics include simulated annealing, tabu search, iterated local search, evolutionary algorithms, and ant colony optimization.

Metalearning

Pavel Brazdil[1], Ricardo Vilalta[2],
Christophe Giraud-Carrier[3], Carlos Soares[1]
[1]University of Porto, Porto, Portugal
[2]University of Houston,
Houston TX, USA
[3]Brigham Young University, UT, USA

Synonyms

Adaptive learning; Dynamic selection of bias; Learning to learn; Ranking learning methods; self-adaptive systems

Definition

Metalearning allows machine learning systems to benefit from their repetitive application. If a learning system fails to perform efficiently, one would expect the learning mechanism itself to adapt in case the same task is presented again. Metalearning differs from base-learning in the scope of the level of adaptation; whereas learning at the base-level is focused on accumulating experience on a specific task (e.g., credit rating, medical diagnosis, mine-rock discrimination, fraud detection, etc.), learning at the metalevel is concerned with accumulating experience on the performance of multiple applications of a learning system.

Briefly stated, the field of metalearning is focused on the relation between tasks or domains, and learning algorithms. Rather than starting afresh on each new task, metalearning facilitates evaluation and comparison of learning algorithms on many different previous tasks, establishes benefits and disadvantages, and then recommends the learning algorithm, or combination of algorithms that maximizes some utility function on the new task. This problem can be seen as an algorithm selection task (Rice, 1976).

The utility or usefulness of a given learning algorithm is often determined through a mapping between characterization of the task and the algorithm's estimated performance (Brazdil & Henery, 1994). In general, metalearning can recommend more than one algorithm. Typically, the number of recommended algorithms is significantly smaller than the number of all possible (available) algorithms (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009).

Motivation and Background

The application of machine learning systems to ►classification and ►regression tasks has become a standard, not only in research but also in commerce and industry (e.g., finance, medicine, and engineering). However, most successful applications are custom-designed, the result of skillful use of human expertise. This is due, in part, to the large, ever increasing number of available machine learning systems, their relative complexity, and the lack of systematic methods for discriminating among them. The problem is further compounded by the fact that, in ►Knowledge Discovery from Databases, each operational phase (e.g., preprocessing, model generation) may involve a choice among various possible alternatives (e.g., progressive vs. random sampling, neural network vs. decision tree learning), as observed by Bernstein, Provost, and Hill (2005).

see Lampert (2009) for computer vision and Schölkopf et al. (2004) for bioinformatics. Finally, Vapnik (1998) provides the details on statistical learning theory.

Recommended Reading

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., & Vishwanathan, S. V. N. (2007). *Predicting structured data*. Cambridge: MIT Press.

Borgwardt, K. M. (2007). *Graph Kernels*. Ph.D. thesis, Ludwig-Maximilians-University, Munich, Germany.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *Proceedings of annual conference computational learning theory* (pp. 144–152). Pittsburgh: ACM Press.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*(3), 273–297.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Tech. Rep. UCS-CRL-99-10). University of California, Santa Cruz.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European conference on machine learning* (pp. 137–142). Berlin: Springer.

Jordan, M. I., Bartlett, P. L., & McAuliffe, J. D. (2003). *Convexity, classification, and risk bounds* (Tech. Rep. 638). University of California, Berkeley.

Lampert, C. H. (2009). Kernel methods in computer vision. *Foundations and Trends in Computer Graphics and Vision*, *4*(3), 193–285.

Platt, J. C. (1999a). Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods—support vector learning* (pp. 185–208). Cambridge, MA: MIT Press.

Platt, J. C. (1999b). Probabilities for sv machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans, (Eds.), *Advances in large margin classifiers* (pp. 61–74). Cambridge: MIT Press.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge: MIT Press.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (2004). *Kernel methods in computational biology*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2000). Margin distribution and soft margin. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans, (Eds.), *Advances in large margin classifiers* (pp. 349–358). Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Shawe-Taylor, J., Bartlett, P. L., Williamson, R. C., & Anthony, M. (1998). Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, *44*(5), 1926–1940.

Smola, A., Vishwanathan, S. V. N., & Le, Q. (2007). Bundle methods for machine learning. In D. Koller, & Y. Singer, (Eds.), *Advances in neural information processing systems* (Vol. 20). Cambridge: MIT Press.

Taskar, B. (2004). *Learning structured prediction models: A large margin approach*. Ph.D. thesis, Stanford University.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Wahba, G. (1990). *Spline models for observational data*. *CBMS-NSF regional conference series in applied mathematics* (Vol. 59). Philadelphia: SIAM.

Swarm Intelligence

Swarm intelligence is the discipline that studies the collective behavior of systems composed of many individuals that interact locally with each other and with their environment and that rely on forms of decentralized control and self-organization. Examples of such systems are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals, and also some artifacts, including swarm robotic systems and some computer programs for tackling optimization problems such as ►ant colony optimization and ►particle swarm optimization.

Symbolic Dynamic Programming

Scott Sanner[1], Kristian Kersting[2]
[1]Statistical Machine Learning Group, NICTA, Canberra, ACT, Australia
[2]Fraunhofer IAIS, Sankt Augustin, Germany

Synonyms

Dynamic programming for relational domains; Relational dynamic programming; Relational value iteration; SDP

Definition

Symbolic dynamic programming (SDP) is a generalization of the ►dynamic programming technique for solving ►Markov decision processes (MDPs) that exploits the symbolic structure in the solution of relational and