E

EBL

►Explanation-Based Learning

Echo State Network

►Reservoir Computing

ECOC

►Error Correcting Output Codes

Edge Prediction

►Link Prediction

Efficient Exploration in Reinforcement Learning

John Langford

Synonyms

PAC-MDP learning

Definition

An agent acting in a world makes observations, takes actions, and receives rewards for the actions taken. Given a history of such interactions, the agent must make the next choice of action so as to maximize the long-term sum of rewards. To do this well, an agent may take suboptimal actions which allow it to gather the information necessary to later take optimal or near-optimal actions with respect to maximizing the long-term sum of rewards. These information gathering actions are generally considered exploration actions.

Motivation

Since gathering information about the world generally involves taking suboptimal actions compared with a later learned policy, minimizing the number of information gathering actions helps optimize the standard goal in reinforcement learning. In addition, understanding exploration well is key to understanding reinforcement learning well, since exploration is a key aspect of reinforcement learning which is missing from standard supervised learning settings (Fig. 1).

Efficient Exploration in Markov Decision Processes

One simplification of reinforcement learning is the ►Markov decision process setting. In this setting, an agent repeatedly takes an action a, resulting in a transition to a state according to a conditional probability transition matrix $P(s'|s,a)$, and a (possibly probabilistic) reward $R(s',a,s) \in [0,1]$. The goal is to efficiently output a policy π which is ϵ-optimal over T timesteps. The value of policy π in a start state s is defined as

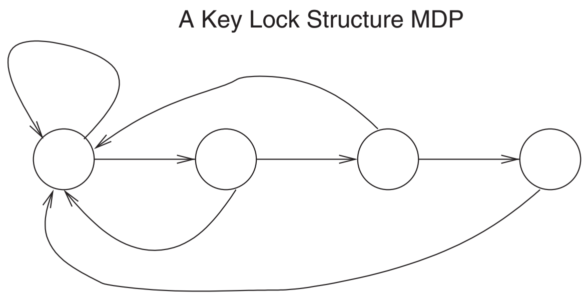
$$\eta(\pi,s) = E_{(a,s,r)^T \sim (\pi,P,R)^T} \sum_{t=1}^{T} r_t,$$

which should be read as the expectation over T-length sequences drawn from the interaction of the policy π with the world as represented by P and R. An ϵ-optimal policy π therefore satisfies:

$$\max_{\pi'} \eta(\pi',s) - \eta(\pi,s) \le \epsilon.$$

There are several notable results in this setting, typically expressed in terms of the dependence on the number of actions A, and the number of states S. The first is for the β-greedy strategy commonly applied when using ►Q-learning (Watkins & Dayan, 1992) which explores randomly with probability β.

© Springer Science+Business Media LLC 2011



Efficient Exploration in Reinforcement Learning. Figure 1. An example of a keylock MDP. The state are arranged in a chain. In each state, one of the two actions leads to the next state while the other leads back to the beginning. The only reward is in the transition to the last state in the chain. Keylock MDPs defeat simple greedy strategies, because the probability of randomly reaching the last transition is exponentially small in the length of the chain

Theorem 1 *There exists MDPs such that with probability at least* $1/2$, β*-greedy requires* $\Theta(A^S)$ *explorations to find an* ϵ*-optimal policy.*

This is essentially a negative result, saying that a greedy exploration strategy cannot quickly discover a good policy in some settings. The proof uses an MDP with a key-lock like structure where for each state all actions but one take the agent back to the beginning state, and the reward is at the end of a chain of states.

It turns out that there exists algorithms capable of finding a near-optimal policy in an MDP with only a polynomial number of exploratory transitions.

Theorem 2 *For all MDPs, for any* $\delta > 0$, *with probability* $1-\delta$, *the algorithm Explicit-Explore-or-Exploit finds an* ϵ*-optimal policy after* $\tilde{O}(S^2A)$ *explorations.*

In other words, E^3 (Kearns & Singh, 1998) requires exploration steps at most proportional to the size of the probability table driving the dynamics of the agent's world. The algorithm works in precisely the manner which might be expected: it builds a model of the world based on its observations and solves the model to determine whether to explore or exploit. The basic approach was generalized to stochastic games and reformulated as an "optimistic initialization" style algorithm named R-MAX (Brafman & Tennenholtz, 2002).

It turns out that an even better dependence is possible using the delayed Q-learning (Strehl, Li, Wiewiora, Langford, & Littman, 2006) algorithm.

Theorem 3 *For all MDPs, for any* $\delta > 0$, *with probability* $1-\delta$, *the algorithm delayed Q-learning finds an* ϵ*-optimal policy after* $\tilde{O}(SA)$ *explorations.*

The delayed Q-learning algorithm requires explorations proportional to the size of the solution policy rather than proportional to the size of world dynamics. At a high level, delayed Q-learning operates by keeping values for exploration and exploitation of observed state-actions, uses these values to decide between exploration and exploitation, and carefully updates these values. Delayed Q-learning does not obsolete E^3, because the (nonvisible) dependence on ϵ and T are worse (Strehl, 2007).

This is a best possible result in terms of the dependence on S and A (up to log factors), as the following theorem (Kakade, 2003) states:

Theorem 4 *For all algorithms, there exists an MDP such that with* $\Omega(SA)$ *explorations are required to find an* ϵ *optimal policy with probability at least* $\frac{1}{2}$.

Since even representing a policy requires a lookup table of size SA, this algorithm-independent lower bound is relatively unsurprising.

Variations on MDP Learning

There are several minor variations in the setting and goal definitions which do not qualitatively impact the set of provable results. For example, if rewards are in a bounded range, they can be offset and rescaled to the interval $[0,1]$.

It's also common to use a soft horizon (or discounting) where the policy evaluation is changed to:

$$\eta_\gamma(\pi,s) = E_{(a,s,r)^\infty \sim (\pi,P,R)^\infty} \sum_{t=1}^{\infty} \gamma^t r_t$$

for some value $\gamma < 1$. This setting is not precisely equivalent to the hard horizon, but since

$$sum_{t=(\ln(1/\epsilon)+\ln(1/1-\gamma))/1-\gamma}^{\infty} \gamma^t r_t \leq \epsilon$$

similar results are provable with $1/(1-\gamma)$ taking the role of T and slightly altered algorithms.

One last variation changes the goal. Instead of outputting an ϵ-optimal policy for the next T timesteps, we could have an algorithm to handle both the exploration and exploitation, then retrospectively go back over a trace of experience and mark a subset of the actions as "exploration actions," with a guarantee that the remainder of the actions are according to an ϵ-optimal policy (Kakade, 2003). Again, minor alterations to known algorithms in the above setting appear to work here.

Alternative Settings

There are several known analyzed variants of the basic setting formed by making additional assumptions about the world. This includes Factored MDPs (Kearns & Koller, 1999), Metric MDPs (Kakade, Kearns, & Langford, 2003), Continuous MDPs (Brunskill, Leffler, Li, Littman, & Roy, 2008), MDPs with a Bayesian prior (Poupart, Vlassis, Hoey, & Regan, 2006), and apprenticeship learning where there is access to a teacher for an MDP (Abbeel & Ng, 2005). The structure of these results are all similar at a high level: with some additional information, it is possible to greatly ease the difficulty of exploration allowing tractable application to much larger problems.

Cross References

►*k* Armed Bandit
►Reinforcement Learning

Recommended Reading

Abbeel, P., & Ng, A. (2005). Exploration and apprenticeship learning in reinforcement learning. In *ICML 2005*, Bonn, Germany.
Brafman, R. I., & Tennenholtz, M. (2002). R-MAX – A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 3,* 213–231.
Brunskill, E., Leffler, B. R., Li, L., Littman, M. L., & Roy, N. (2008). CORL: A continuous-state offset-dynamics reinforcement learner. In *UAI-08*, Helsinki, Finland, July 2008.
Kakade, S. (2003). Thesis at Gatsby Computational Neuroscience Unit.
Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric state spaces. In *ICML 2003*.
Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th international joint conference on artificial intelligence* (pp. 740–747). San Francisco: Morgan Kaufmann.
Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *ICML 1998* (pp. 260–268). San Francisco: Morgan Kaufmann.
Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *ICML 2006* (pp. 697–704). New York: ACM Press.
Strehl, A. (2007). Thesis at Rutgers University.
Strehl, A. L., Li, L., Wiewiora, E., Langford, J., & Littman, M. L. (2006). PAC model-free reinforcement learning. In *Proceedings of the 23rd international conference on machine learning (ICML 2006)* (pp. 881–888).
Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning Journal, 8*, 279–292.

EFSC

►Evolutionary Feature Selection and Construction

Elman Network

►Simple Recurrent Network

EM Algorithm

►Expectation–Maximization Algorithm

EM Clustering

►Expectation Maximization Clustering

Embodied Evolutionary Learning

►Evolutionary Robotics

Emerging Patterns

Definition

Emerging pattern mining is an area of ▶supervised descriptive rule induction. Emerging patterns are defined as itemsets whose support increases significantly from one data set to another (Dong & Li, 1999). Emerging patterns are said to capture emerging trends in time-stamped databases, or to capture differentiating characteristics between classes of data.

Recommended Reading

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99)* (pp. 43–52).

Empirical Risk Minimization

Xinhua Zhang
Australian National University
NICTA London Circuit
Canberra, Australia

Definition

The goal of learning is usually to find a model which delivers good generalization performance over an underlying distribution of the data. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. Assume the pairs $(X \times Y) \in \mathcal{X} \times \mathcal{Y}$ are random variables whose (unknown) joint distribution is P_{XY}. It is our goal to find a predictor $f : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes the expected risk:

$$P(f(X) \neq Y) = \mathbb{E}_{(X,Y)\sim P_{XY}}[\delta(f(X) \neq Y)],$$

where $\delta(z) = 1$ if z is true, and 0 otherwise.

However, in practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Since P_{XY} is unknown, we often use the risk on the training set (called empirical risk) as a surrogate of the expected risk on the underlying distribution:

$$\frac{1}{n}\sum_{i=1}^{n} \delta(f(X_i) \neq Y_i).$$

Empirical risk minimization (ERM) refers to the idea of choosing a function f by minimizing the empirical risk. Although it is often effective and efficient, ERM is subject to ▶overfitting, i.e., finding a model which fits the training data well but predicts poorly on unseen data. Therefore, ▶regularization is often required.

More details about ERM can be found in Vapnik (1998).

Recommended Reading

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

Ensemble Learning

Gavin Brown
The University of Manchester
Manchester, UK

Synonyms

Committee machines; Multiple classifier systems

Definition

Ensemble learning refers to the procedures employed to train multiple learning machines and combine their outputs, treating them as a "committee" of decision makers. The principle is that the decision of the committee, with individual predictions combined appropriately, should have better overall ▶accuracy, on average, than any individual committee member. Numerous empirical and theoretical studies have demonstrated that ensemble ▶models very often attain higher accuracy than single models.

The members of the ensemble might be predicting real-valued numbers, class labels, posterior probabilities, rankings, clusterings, or any other quantity. Therefore, their decisions can be combined by many methods, including averaging, voting, and probabilistic methods. The majority of ensemble learning methods are generic, applicable across broad classes of model types and learning tasks.

Motivation and Background

If we could build the "perfect" machine learning device, one which would give us the best possible answer every time, there would be no need for *ensemble learning* methods – indeed, there would be no need for this encyclopedia either. The underlying principle of ensemble learning is a recognition that in real-world situations, every model has limitations and will make errors. Given that each model has these "limitations," the aim of ensemble learning is to manage their strengths and weaknesses, leading to the best possible decision being taken overall. Several theoretical and empirical results have shown that the accuracy of an ensemble can significantly exceed that of a single model.

The principle of combining predictions has been of interest to several fields over many years. Over 200 years ago, a controversial question had arisen, on how best to estimate the mean of a probability distribution given a small number of sample observations. Laplace (1818) demonstrated that the sample mean was not always optimal: under a simple condition, the sample median was a better combined predictor of the population mean. The financial forecasting community has analyzed model combination for several decades, in the context of stock portfolios. The contribution of the machine learning (ML) community emerged in the 1990s – automatic construction (from data) of both the models and the method to combine them. While the majority of the ML literature on this topic is from 1990 onward, the principle has been explored briefly by several independent authors since the 1960s. See Kuncheva (2004b) for historical accounts.

The study of ensemble methods, with model outputs considered for their abstract properties rather than the specifics of the algorithm which produced them, allows for a wide impact across many fields of study. If we can understand precisely why, when, and how particular ensemble methods can be applied successfully, we would have made progress toward a powerful new tool for Machine Learning: *the ability to automatically exploit the strengths and weaknesses of different learning systems*.

Methods and Algorithms

An ensemble consists of a set of models and a method to combine them. We begin this section by assuming that we have a set of models, generated by any of the learning algorithms in this encyclopedia; we explore popular methods of combining their outputs, for classification and regression problems. Following this, we review some of the most popular ensemble algorithms, for *learning* a set of models given the knowledge that they will be combined, including extensive pointers for further reading. Finally, we take a theoretical perspective, and review the concept of ensemble *diversity*, the fundamental property which governs how well an ensemble can perform.

Methods for Combining a Set of Models

There exist numerous methods for model combination, far too many to fully detail here. The *linear* combiner, the *product* combiner, and the *voting* combiner are by far the most commonly used in practice. Though a combiner could be specifically chosen to optimize performance in a particular application, these three rules have shown consistently good behavior across many problems, and are simple enough that they are amenable to theoretical analysis.

The linear combiner is used for models that output real-valued numbers, so is applicable for ►regression ensembles, or for ►classification ensembles producing class probability estimates. Here, notation for the latter case is only shown. We have a model $f_t(y|\mathbf{x})$, an estimate of the probability of class y given input $\mathbf{x}$. For a set of these, $t = \{1, \ldots, T\}$, the ensemble probability estimate is,

$$\bar{f}(y|\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(y|\mathbf{x}). \quad (1)$$

If the weights $w_t = 1/T, \forall t$, this is a simple uniform averaging of the probability estimates. The notation clearly allows for the possibility of a nonuniformly weighted average. If the classifiers have different accuracies on the data, a nonuniform combination could *in theory* give a lower error than a uniform combination. However, in practice, the difficulty of estimating the $\mathbf{w}$ parameters without overfitting, and the relatively small gain that is available (see Kuncheva, 2004b, p. 282), have meant that in practice the uniformly weighted average is by far the most commonly used. A notable exception, to be discussed later in this article, is the *mixture of experts* paradigm – in MoE, weights are nonuniform, but are learnt and dependent on the input value $\mathbf{x}$. An alternative combiner is the *product rule*:

$$\bar{f}(y|\mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{T} f_t(y|\mathbf{x})^{w_t}, \qquad (2)$$

where Z is a normalization factor to ensure $\bar{f}$ is a valid distribution. Note that Z is not *required* to make a valid decision, as the order of posterior estimates remain unchanged before/after normalization. Under the assumption that the class-conditional probability estimates are independent, this is the theoretically optimal combination strategy. However, this assumption is highly unlikely to hold in practice, and again the weights $\mathbf{w}$ are difficult to reliably determine. Interestingly, the linear and product combiners are in fact special cases of the *generalized mean* (Kuncheva, 2004b) allowing for a continuum of possible combining strategies.

The linear and product combiners are applicable when our models output real-valued numbers. When the models instead output class labels, a majority (or plurality) vote can be used. Here, each classifier votes for a particular class, and the class with the most votes is chosen as the ensemble output. For a two-class problem the models produce labels, $h_t(\mathbf{x}) \in \{-1, +1\}$. In this case, the ensemble output for the *voting* combiner can be written as

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right). \qquad (3)$$

The weights $\mathbf{w}$ can be uniform for a simple majority vote, or nonuniform for a weighted vote.

We have discussed only a small fraction of the possible combiner rules. Numerous other rules exist, including methods for combining rankings of classes, and unsupervised methods to combine clustering results. For details of the wider literature, see Kuncheva (2004b) or Polikar (2006).

Algorithms for Learning a Set of Models

If we had a committee of *people* taking decisions, it is self-evident that we would not want them all to make the same bad judgments *at the same time*. With a committee of learning models, the same intuition applies: we will have no gain from combining a set of identical models. We wish the models to exhibit a certain element of "diversity" in their group behavior, though still retaining good performance individually.

We therefore make a distinction between two types of ensemble learning algorithms, those which encourage diversity *implicitly*, and those which encourage it *explicitly*. The vast majority of ensemble methods are *implicit*, in that they provide different *random subsets* of the training data to each learner. Diversity is encouraged "implicitly" by *random* sampling of the data space: at no point is a *measurement* taken to ensure diversity will emerge. The random differences between the datasets might be in the selection of examples (the ▶Bagging algorithm), the selection of features (▶Random Subspace Method, Ho, 1998 or ▶Rotation Forests, Rodriguez, Kuncheva, & Alonso, 2006), or combinations of the two (the Random Forests algorithm, Breiman, 2001). Many other "randomization" schemes are of course possible.

An alternative is to *explicitly* encourage diversity, constructing each ensemble member with some measurement ensuring that it is substantially different from the other members. ▶Boosting algorithms achieve this by altering the distribution of training examples for each learner such that it is encouraged to make more accurate predictions where previous predictors have made errors. The DECORATE algorithm (Melville & Mooney, 2005) explicitly alters the distribution of class labels, such that successive models are forced to learn different answers to the same problem. ▶Negative correlation learning (see Brown, 2004; Brown, Wyatt, Harris, & Yao, 2005), includes a penalty term when learning each ensemble member, explicitly *managing* the accuracy-diversity trade-off.

In general, ensemble methods constitute a large class of algorithms – some based on heuristics, and some on sound learning-theoretic principles. The three algorithms that have received the most attention in the literature are reviewed here. It should be noted that we present only the most basic form of each; numerous modifications have been proposed for a variety of learning scenarios. As further study the reader is referred to the many comprehensive surveys of the field (Brown et al., 2005; Kuncheva, 2004b; Polikar, 2006).

Bagging

In the Bagging algorithm (Breiman, 1996), each member of the ensemble is constructed from a different training dataset, and the predictions combined either by uniform averaging or voting over class labels. Each dataset is generated by sampling from the total N data examples, choosing N items uniformly at random *with replacement*. Each sample is known as a *bootstrap*; the name Bagging is an acronym derived from Bootstrap *AGG*regat*ING*. Since a bootstrap samples N items uniformly at random with replacement, the probability of any individual data item *not* being selected is $p = (1 - 1/N)^N$. Therefore with large N, a single bootstrap is expected to contain approximately 63.2% of the original set, while 36.8% of the originals are not selected.

Like many ensemble methods, Bagging works best with *unstable* models, that is those that produce differing generalization behavior with small changes to the training data. These are also known as *high variance* models, examples of which are ▶decision trees and ▶neural networks. Bagging therefore tends not to work well with very simple models. In effect, Bagging samples randomly from the space of possible models to make up the ensemble – with very simple models the sampling produces almost identical (low diversity) predictions.

Despite its apparent capability for variance reduction, situations have been demonstrated where Bagging can converge *without* affecting variance (see Brown et al., 2005). Several other explanations have been proposed for Bagging's success, including links to Bayesian model averaging. In summary, it seems that several years from its introduction, despite its apparent simplicity, Bagging is still not fully understood.

Algorithm 1 Bagging

```
Input: Required ensemble size $T$
Input: Training set $S=\{(x_1,y_1),(x_2,y_2),\dots,(x_N,y_N)\}$
for $t = 1$ to $T$ do
    Build a dataset $S_t$, by sampling $N$ items, randomly with replacement from $S$.
    Train a model $h_t$ using $S_t$, and add it to the ensemble.
end for
For a new testing point $(x',y')$,
If model outputs are continuous, combine them by averaging.
If model outputs are class labels, combine them by voting.
```

Adaboost

Adaboost (Freund & Schapire, 1996) is the most well known of the *Boosting* family of algorithms (Schapire, 2003). The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models should be able to compensate for errors made by earlier models.

Adaboost occupies somewhat of a special place in the history of ensemble methods. Though the procedure seems heuristic, the algorithm is in fact grounded in a rich learning-theoretic body of literature. Schapire (1990) addressed a question posed by Kearns and Valiant (1988) on the nature of two complexity classes of learning problems. The two classes are *strongly learnable* and *weakly learnable* problems. Schapire showed that these classes were equivalent; this had the corollary that a weak model, performing only slightly better than random guessing, could be "boosted" into an arbitrarily accurate *strong* model. The original Boosting algorithm was a proof by construction of this equivalence, though had a number of impractical assumptions built-in. The Adaboost algorithm (Freund & Schapire, 1996) was the first practical Boosting method. The authoritative historical account of the development can be found in Schapire (1999), including discussion of numerous variants and interpretations of the algorithm. The procedure is shown in Algorithm 2. Some similarities with Bagging are evident; a key differences is that at each round t, Bagging has a uniform distribution D_t, while Adaboost adapts a nonuniform distribution.

The ensemble is constructed by iteratively adding models. Each time a model is learnt, it is checked to ensure it has at least $\epsilon_t < 0.5$, that is, it has performance *better than random guessing* on the data it was supplied with. If it does not, either an alternative model is constructed, or the loop is terminated.

After each round, the distribution D_t is updated to emphasize incorrectly classified examples. The update causes half the distribution mass of D_{t+1} to be over the examples incorrectly classified by the previous model. More precisely, $\sum_{h_t(\mathbf{x}_i)\neq y_i} D_{t+1}(i) = 0.5$. Thus, if h_t has an error rate of 10%, then examples from that small 10% will be allocated 50% of the next model's training "effort,"

Algorithm 2 Adaboost

```
Input: Required ensemble size $T$
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$
Define a uniform distribution $D_1(i)$ over elements of $S$.
for $t = 1$ to $T$ do
    Train a model $h_t$ using distribution $D_t$.
    Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$
    If $\epsilon_t \geq 0.5$ break
    Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
    where $Z_t$ is a normalization factor so that $D_{t+1}$ is a valid distribution.
end for
For a new testing point $(x', y')$,
$H(x') = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x'))$
```

while the remaining examples (those correctly classified) are underemphasized. An equivalent (and simpler) writing of the distribution update scheme is to multiply $D_t(i)$ by $1/2(1-\epsilon_t)$ if $h_t(x_i)$ is correct, and by $1/2\epsilon_t$ otherwise.

The updates cause the models to sequentially minimize an exponential bound on the error rate. The training error rate on a data sample $\mathcal{S}$ drawn from the true distribution $\mathcal{D}$ obeys the bound,

$$P_{\mathbf{x},y\sim\mathcal{S}}(yH(\mathbf{x}) < 0) \leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}. \quad (4)$$

This *upper bound* on the training error (though not the *actual* training error) is guaranteed to decrease monotonically with T, given $\epsilon_t < 0.5$.

In an attempt to further explain the performance of Boosting algorithms, Schapire also developed bounds on the *generalization* error of voting systems, in terms of the voting margin, the definition of which was given in (10). Note that, this is not the same as the *geometric margin*, optimized by ▶support vector machines. The difference is that the voting margin is defined using the one-norm $||\mathbf{w}||_1$ in the denominator, while the geometric margin uses the *two*-norm $||\mathbf{w}||_2$. While this is a subtle difference, it is an important one, forming links between SVMs and Boosting algorithms – see Rätsch, Mika, Schölkopf, and Müller (2002) for details. The following bound holds with probability $1-\delta$,

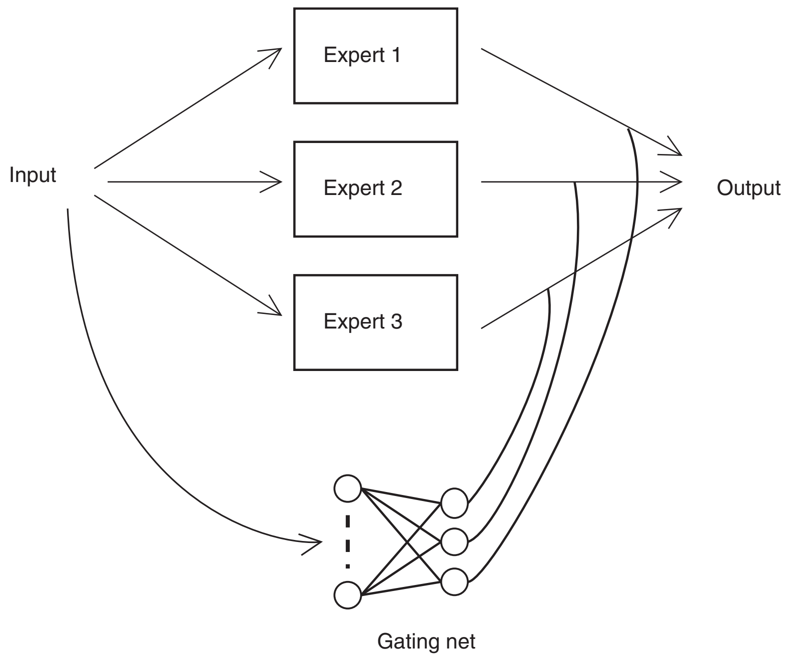
$$P_{\mathbf{x},y\sim\mathcal{D}}(H(\mathbf{x}) \neq y) \leq P_{\mathbf{x},y\sim\mathcal{S}}(yH(\mathbf{x}) < \theta) + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2} - \ln\delta}\right), \quad (5)$$

where the $\tilde{O}$ notation hides constants and logarithmic terms, and d is the ▶VC-dimension of the model used. Roughly, this states that the generalization error is less than or equal to the training error plus a term dependent on the voting margin. The larger the minimum margin in the training data, the lower the testing error. The original bounds have since been significantly improved, see Koltchinskii and Panchenko (2005) as a comprehensive recent work. We note that this bound holds generally for *any* voting system, and is not specific to the Boosting framework.

The margin-based theory is only one explanation of the success of Boosting algorithms. Mease and Wyner (2008) present a discussion of several questions on why and how Adaboost succeeds. The subsequent 70 pages of discussion demonstrate that the story is by no means simple. The conclusion is, while no single theory can fully explain Boosting, each provides a different part of the still unfolding story.

Mixtures of Experts

The mixtures of experts architecture is a widely investigated paradigm for creating a combination of models (Jacobs, Jordan, Nowlan, & Hinton, 1991). The principle underlying the architecture is that certain models will be able to "specialize" to particular parts of the input space. It is commonly implemented with a neural network as the base model, or some other model capable of estimating probabilities. A *Gating network* receives the same inputs as the component models, but its outputs are used as the weights for a linear combiner. The Gating network is responsible for learning the appropriate weighted combination of the specialized models ("experts") for any given input. Thus, the input space is "carved-up" between the experts, increasing and decreasing their weights for particular examples. In effect, a mixture of experts explicitly learns how to create expert ensemble members in different portions of the input space, and select the most appropriate subset for a new testing example (Fig. 1).



Ensemble Learning. Figure 1. **The mixtures of experts architecture**

The architecture has received wide attention, and has a strong following in the probabilistic modeling community, where it may go under the pseudonym of a "mixture model." A common training method is the ►expectation-maximization algorithm.

Theoretical Perspectives: Ensemble Diversity

We have seen that all ensemble algorithms in some way attempt to encourage "diversity." In this section, we take a more formalized perspective, to understand what is meant by this term.

What is Diversity?

The optimal "diversity" is fundamentally a *credit assignment* problem. If the committee as a whole makes an erroneous prediction, how much of this error should be attributed to each member? More precisely, how much of the committee prediction is due to the accuracies of the individual models, and how much is due to their interactions when they were combined? We would ideally like to reexpress the ensemble error as two distinct components: a term for the accuracies of the individual models, plus a term for their interactions, i.e., their *diversity*.

It turns out that this so-called *accuracy-diversity* breakdown of the ensemble error is not always possible, depending on the type of error function, and choice of combiner rule. It should be noted that when "diversity" is referred to in the literature, it is most often meant to indicate classification with a majority vote combiner, but for completeness we address the general case here. In the following sections, the existing work to understand diversity in three distinct cases is described: for regression tasks (a linear combiner), and classification tasks, with either a linear combiner or a voting combiner.

Regression Error with a Linear Combination Rule

In a regression problem, it is common to use the squared error criterion. The accuracy-diversity breakdown for this case (using a linear combiner) is called the *ambiguity decomposition* (Krogh & Vedelsby, 1995). The result states that the squared error of the linearly combined ensemble, $\bar{f}(\mathbf{x})$, can be broken into a sum of two components:

$$(\bar{f}(\mathbf{x}) - d)^2 = \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - d)^2 - \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - \bar{f}(\mathbf{x}))^2. \tag{6}$$

The first term on the right hand side is the average squared error of the individual models, while the second term quantifies the interactions *between* the predictions. Note that this second term, the "ambiguity," is always positive. This guarantees that, for an arbitrary data point, the ensemble squared error is always less than or equal to the average of the individual squared errors.

The intuition here can be understood as follows. Imagine five friends, playing "guess the weight of the cake" (an old English fairground game): if a player's guess is close enough to the true weight, they win the cake. Just as they are about to play, the fairground manager states that they can only submit *one* guess. The dilemma seems to be in whose guess they should submit – however, the ambiguity decomposition shows us that taking the average of their guesses, and submitting that, will *always* be closer (on average) than choosing a person at random and submitting their guess. Note that this is qualified with "on average" – it may well be that one of the predictions will in fact be closer than the average prediction, but we presume that we have no way of identifying *which* prediction to choose, other than random. It can be seen that greater diversity in the predictions (i.e., a larger ambiguity term) results in a larger gain over the average individual performance. However, it is also clear that there is a trade-off to be had: too much diversity and the average error is extremely large.

The idea of a trade-off between these two terms is reminiscent of the ▶bias-variance decomposition (Geman, Bienenstock, & Doursat, 1992); in fact, there is a deep connection between these results. Taking the expected value of (6) over all possible training sets gives us the ensemble analogy to the bias-variance decomposition, called the ▶bias-variance-covariance decomposition (Ueda & Nakano, 1996). This shows that the expected squared error of an ensemble $\bar{f}(\mathbf{x})$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{(\bar{f}(\mathbf{x})-d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1-\frac{1}{T}\right)\overline{\text{covar}}, \quad (7)$$

where the expectation is with respect to all possible training datasets $\mathcal{D}$. While the bias and variance terms are constrained to be positive, the covariance between models can become negative – thus the definition of diversity emerges as an extra degree of freedom in the bias-variance dilemma. This extra degree of freedom allows an ensemble to approximate functions that are difficult (if not impossible) to find with a single model. See Brown et al. (2005) for extensive further discussion of this concept.

Classification Error with a Linear Combination Rule

In a classification problem, our error criterion is the misclassification rate, also known as the *zero-one* loss function. For this type of loss, it is well known there is no unique definition of bias-variance; instead there exist multiple decompositions each with advantages and disadvantages (see Kuncheva, 2004b, p. 224). This gives us a clue as to the situation with an ensemble – there is also no simple accuracy-diversity separation of the ensemble classification error. Classification problems can of course be addressed either by a model producing class probabilities (where we linearly combine), or directly producing class labels (where we use majority vote). Partial theory has been developed for each case.

For linear combiners, there exist theoretical results that relate the correlation of the probability estimates to the ensemble classification error. Tumer and Ghosh (1996) showed that the reducible classification error (i.e., above the Bayes rate) of a simple averaging ensemble, e_{ave}, can be written as

$$e_{\text{ave}} = e_{\text{add}}\left(\frac{1+\delta(T-1)}{T}\right), \quad (8)$$

where e_{add} is the classification error of an individual model. The δ is a correlation coefficient between the model outputs. When the individual models are identical, the correlation is $\delta = 1$. In this case, the ensemble error is equal to the individual error, $e_{\text{ave}} = e_{\text{add}}$. When the models are statistically independent, $\delta = 0$, and the ensemble error is a fraction $1/T$ of the individual error, $e_{\text{ave}} = 1/T \times e_{\text{add}}$. When δ is negative, the models are negatively correlated, and the ensemble error is lower than the average individual error. However, (8) is derived under quite strict assumptions, holding only for a local area around the decision boundary, and ultimately resting on the bias-variance-covariance theory from regression problems. Further details, including recent work to lift some of the assumptions (Kuncheva, 2004b).

Classification Error with a Voting Combination Rule

The case of a classification problem with a majority vote combiner is the most challenging of all. In general, there is no known breakdown of the ensemble classification error into neat accuracy and diversity components. The simplest intuition to show that correlation between models does affect performance is given by the Binomial theorem. If we have T models each with identical error probability $p = P(h_t(\mathbf{x}) \neq y)$, assuming they make statistically *independent* errors, the following error probability of the majority voting committee holds,

$$P(H(\mathbf{x}) \neq y) = \sum_{k>(T/2)}^{T} \binom{T}{k} p^k (1-p)^{(T-k)}. \qquad (9)$$

For example, in the case of $T = 21$ ensemble members, each with error $p = 0.3$, the majority voting error will be 0.026, an order of magnitude improvement over the individual error. However, this *only* holds for statistically independent errors. The correlated case is an open problem. Instead, various authors have proposed their own heuristic definitions of diversity in majority voting ensembles. Kuncheva (2004b) conducted extensive studies of several suggested diversity measures; the conclusion was that "*no measure consistently correlates well with the majority vote accuracy.*" In spite of this, some were found useful as an approximate guide to characterize performance of ensemble methods, though should not be relied upon as the "final word" on diversity. Kuncheva's recommendation in this case is the *Q-statistic* (Kuncheva, 2004b, p. 299), due to its simplicity and ease of computation.

Breiman (2001) took an alternative approach, deriving not a *separation* of error components, but a *bound* on the generalization error of a voting ensemble, expressed in terms of the correlations of the models. To understand this, we must introduce concept of *voting margin*. The voting margin for a two-class problem, with $y \in \{-1, +1\}$, is defined,

$$m = \frac{y_t \sum_{t=1}^{T} w_t h_t(\mathbf{x})}{\sum_{t=1}^{T} |w_t|} = yH(\mathbf{x}). \qquad (10)$$

If the margin is positive, the example is correctly classified, if it is negative, the example is incorrectly classified. The expected margin $s = \mathcal{E}_{\mathcal{D}}\{m\}$ measures the extent to which the average number of votes for the correct class exceeds the average vote for any other class, with respect to the data distribution $\mathcal{D}$. The larger the voting margin, the more confidence in the classification. Breiman's bound shows,

$$P_{\mathcal{D}}(H(\mathbf{x}) \neq y) = P_{\mathcal{D}}(yH(\mathbf{x}) < 0) \leq \frac{\bar{\rho}(1-s^2)}{s^2}. \qquad (11)$$

Here $\bar{\rho}$ is the average pairwise correlation between the errors of the individual models. Thus, the generalization error is minimized by a small $\bar{\rho}$, and an s as close to 1 as possible. The balance between a high accuracy (large s) and a high diversity (low $\bar{\rho}$) constitutes the tradeoff in this case, although the bound is quite loose.

Summary

In summary, the definition of diversity depends on the problem. In a regression problem, the optimal diversity is the trade-off between the bias, variance and covariance components of the squared error. In a classification problem, with a linear combiner, there exists partial theory to relate the classifier correlations to the ensemble error rate. In a classification problem with a voting combiner, there is no single theoretical framework or definition of diversity. However, the lack of an agreed definition of diversity has not discouraged researchers from trying to achieve it, nor has it stalled the progress of effective algorithms in the field.

Conclusions & Current Directions in the Field

Ensemble methods constitute some of the most robust and accurate learning algorithms of the past decade (Caruana & Niculescu-Mizil, 2006). A multitude of heuristics have been developed for randomizing the ensemble parameters, to generate diverse models. It is arguable that this line of investigation is nowadays rather oversubscribed, and the more interesting research is now in methods for nonstandard data. ►Cluster ensembles (Strehl & Ghosh, 2003) are ensemble techniques applied to unsupervised learning problems. Problems with *nonstationary* data, also known as *concept drift*, are receiving much recent attention (Kuncheva, 2004a). The most up to date innovations are to be found in the biennial *International Workshop on Multiple Classifier Systems* (Roli et al., 2000).

Recommended Reading

Kuncheva (2004b) is the standard reference in the field, which includes references to many further recommended readings. In addition, Brown et al. (2005) and Polikar (2006) provide extensive literature surveys. Roli et al. (2000) is an international workshop series dedicated to ensemble learning.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning 45*(1), 5–32.

Brown, G. (2004). *Diversity in neural network ensembles*. PhD thesis, University of Birmingham.

Brown, G., Wyatt, J. L., Harris, R., & Yao, X. (2005). Diversity creation methods: A survey and categorisation. *Journal of Information Fusion 6*(1), 5–20.

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on machine learning* (pp. 161–168). New York: ACM.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on machine learning (ICML'96)* (pp. 148–156). San Francisco: Morgan Kauffman Publishers.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation 4*(1), 1–58.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*(8), 832–844.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation 3*(1), 79–87.

Kearns, M., & Valiant, L. G. (1988). *Learning Boolean formulae or finite automata is as hard as factoring*. Technical report TR-14-88, Harvard University Aiken Computation Laboratory.

Koltchinskii, V., & Panchenko, D. (2005). Complexities of convex combinations and bounding the generalization error in classification. *Annals of Statistics 33*(4), 1455.

Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross-validation and active learning. In *Advances in neural information processing systems* (pp. 231–238). Cambridge, MA: MIT Press.

Kuncheva, L. I. (2004a). Classifier ensembles for changing environments. In *International workshop on multiple classifier systems*. Lecture Notes in Computer Science 3007. Berlin: Springer.

Kuncheva, L. I. (2004b). *Combining pattern classifiers: Methods and algorithms*. New York: Wiley.

Laplace, P. S. (1818). *Deuxieme supplement a la theorie analytique des probabilites*. Paris: Gauthier-Villars.

Mease, D., & Wyner, A. (2008). Evidence contrary to the statistical view of Boosting. *Journal of Machine Learning Research 9*, 131–156.

Melville, P., & Mooney, R. J. (2005). Creating diversity in ensembles using artificial data. *Information Fusion 6*(1), 99–111.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine, 6*(3), 21–45.

Rätsch, G., Mika, S., Schölkopf, B., & Müller, K. R. (2002). Constructing Boosting algorithms from SVMs: An application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*(9), 1184–1199.

Rodriguez, J., Kuncheva, L., & Alonso, C. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*(10), 1619–1630.

Roli, F., Kittler, J., Windridge, D., Oza, N., Polikar, R., Haindl, M., et al. (Eds.). *Proceedings of the international workshop on multiple classifier systems 2000–2009. Lecture notes in computer science*. Berlin: Springer. Available at: http://www.informatik.uni-trier.de/ley/db/conf/mcs/index.html

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*, 197–227.

Schapire, R. E. (1999). A brief introduction to Boosting. In *Proceedings of the 16th international joint conference on artificial intelligence* (pp. 1401–1406). San Francisco, CA: Morgan Kaufmann.

Schapire, R. E. (2003). *The boosting approach to machine learning: An overview*. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, & B. Yu (Eds.), *Nonlinear estimation & classification Lecture notes in statistics* (pp. 149–172). Berlin: Springer.

Strehl, A., & Ghosh, J. (2003). Cluster ensembles – A knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research 3*, 583–617.

Tumer, K., & Ghosh, J. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science 8*(3–4), 385–403.

Ueda, N., & Nakano, R. (1996). Generalization error of ensemble estimators. In *Proceedings of IEEE international conference on neural networks* (Vol. 1, pp. 90–95). ISBN: 0-7803-3210-5

Entailment

Synonyms

Implication; Logical consequence

Definition

The term entailment is used in the context of logical reasoning. Formally, a logical formula T entails a formula c if and only if all models of T are also a model of c. This is usually denoted as $T \vDash c$ and means that c is a logical consequence of T or that c is implied by T.

Let us elaborate this definition for propositional clausal logic, where the formulae T could be the following expression:

```
flies :- bird, normal.
bird :- blackbird.
bird :- ostrich.
```

Here, the first clause or rule can be read as flies *if* normal *and* bird, that is, normal birds fly, the second and third one as stating that blackbirds, resp. ostriches, are birds. An interpretation is then an assignment of truth-values to the propositional variables. For instance, for the above domain

```
{ostrich, bird}
{blackbird, bird, normal}
```

are interpretations, specified through the set of propositional variables that are true. This means that in the first interpretation, the only true propositions are `ostrich` and `bird`. An interpretation specifies a kind of possible world. An interpretation I is then a model for a clause $h : -b_1, \ldots, b_n$ if and only if $\{b_1, \ldots, b_n\} \subseteq I \rightarrow h \in I$ and it is model for a clausal theory if and only if it is a model for all clauses in the theory. Therefore, the first interpretation above is a model for the theory, but the second one is not because the interpretation is not a model for the first clause (as `{bird, normal}` $\subseteq I$ but `flies` $\notin I$). Using these notions, it can now be verified that the clausal theory T above logically entails the clause

```
flies :- ostrich, normal.
```

because all models of the theory are also a model for this clause.

In machine learning, the notion of entailment is used as a covers relation in ►inductive logic programming, where hypotheses are clausal theories, instances are clauses, and an example is covered by the hypothesis when it is entailed by the hypothesis.

Cross References

►Inverse Entailment
►Learning from Entailment
►Logic of Generality

Recommended Reading

Russell, S., & Norvig, P. *Artificial intelligence: A modern approach* (2nd ed.). Prentice Hall.

Entity Resolution

Indrajit Bhattacharya[1], Lise Getoor[2]
[1]IBM India Research Laboratory, New Delhi, India
[2]University of Maryland, College Park, MD, USA

Synonyms

Co-reference resolution; Deduplication; Duplicate detection; Identity uncertainty; Merge-purge; Object consolidation; Record linkage; Reference reconciliation

Definition

A fundamental problem in data cleaning and integration (see ►Data Preparation) is dealing with uncertain and imprecise references to real-world entities. The goal of entity resolution is a take a collection of uncertain entity references (or references, in short) from a single data source or multiple data sources, discover the unique set of underlying entities, and map each reference to its corresponding entity. This typically involves two subproblems – identification of references with different attributes to the same entity, and disambiguation of references with identical attributes by assigning them to different entities.

Motivation and Background

Entity resolution is a common problem that comes up in different guises (and is given different names) in many computer science domains. Examples include computer vision, where we need to figure out when regions in two different images refer to the same underlying object (the correspondence problem); natural language processing when we would like to determine which noun phrases refer to the same underlying entity (co-reference resolution); and databases, where, when merging two databases or cleaning a database, we would like to determine when two tuple records are referring to the same real-world object (deduplication and data integration). Deduplication is important for removing redundancy and for accurate analysis. In information integration, determining approximate joins is important for consolidating information from multiple sources; most often there will not be a unique key that can be used to join tables across databases.

Such ambiguities in entity references can occur due to multiple reasons. Often times, data may have data entry errors, such as typographical errors. Multiple representations, such as abbreviations, are also possible. Different databases typically have different keys – one person database may use social security numbers while another uses name and address.

Traditional entity resolution approaches focus on matching attributes of different references for resolving entities. However, many data sources have explicit or implicit relationships present among the entity references. These relations are indicative of relationships between the underlying entities themselves. For example, person records in census data are linked by family relationships such as sibling, parent, and spouse. Researchers collaborate mostly within their organization, or their research community, as a result of which references to related researchers tend to occur closely together. Recent entity resolution approaches in statistical relational learning make use of relationships between references to improve entity resolution accuracy, and additionally to discover relationships between the underlying entities.

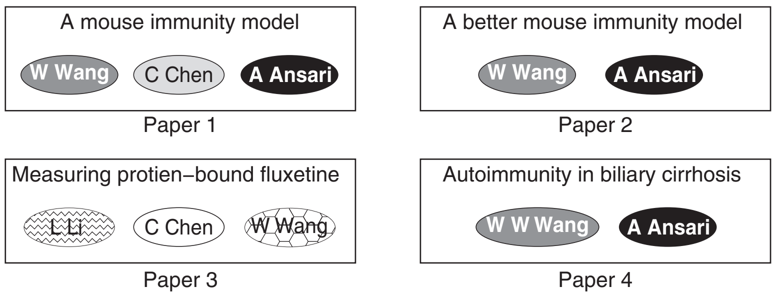
Theory/Solution

As an illustration of the entity resolution problem, consider the task of resolving the author references in a database of academic publications similar to DBLP, CiteSeer or PubMed. Let us take as an example the following set of four papers:

1. W. Wang, C. Chen, A. Ansari, "A mouse immunity model"
2. W. Wang, A. Ansari, "A better mouse immunity model"
3. L. Li, C. Chen, W. Wang, "Measuring protein-bound fluxetine"
4. W. W. Wang, A. Ansari, "Autoimmunity in biliary cirrhosis"

Now imagine that we would like to find out, given these four papers, which of these author names refer to the same author entities. This process involves determining whether paper 1 and paper 2 are written by the same author named Wang, or whether they are different authors. We need to answer similar questions about all such similar author names in the database.

In this example, it turns out there are six underlying author entities, which we will call *Wang*1 and *Wang*2, *Chen*1 and *Chen*2, *Ansari* and *Li*. The three references with the name "A. Ansari" correspond to author *Ansari* and the reference with name "L. Li" to author *Li*. However, the two references with name "C. Chen" map to two different authors *Chen*1 and *Chen*2. Similarly, the four references with name "W. Wang" or "W. W. Wang" map to two different authors. The "Wang" references from the first, second, and fourth papers correspond to author *Wang*1, while that from the third paper maps to a different author *Wang*2. This inference illustrates the twin problems of *identifying* "W. Wang" and "W. W. Wang" as the same author, and *disambiguating* two references with name "W. Wang" as different authors. This is shown pictorially in Fig. 1,



Entity Resolution. Figure 1. The references in different papers in the bibliographic example. References to the same entity are identically shaded

where references that correspond to the same authors are shaded identically. In the entity resolution process, all those and only those author references that are shaded identically should be resolved as corresponding to the same underlying entity.

Formally, in the entity resolution problem, we are given a set of references $\mathcal{R} = \{r_i\}$, where each reference r has attributes $r.A_1, r.A_2, \ldots, r.A_k$, such as observed names and affiliations for author references, as in our example above. The references correspond to some set of unknown entities $\mathcal{E} = \{e_i\}$. We introduce the notation $r.E$ to refer to the entity to which reference r corresponds. The goal is to recover the hidden set of entities $\mathcal{E} = \{e_i\}$ and the entity labels $r.E$ for individual references given the observed attributes of the references. In addition to the attributes, in some data sources we have information in the form of relationships between the references, such as coauthor relationships between author references in publication databases. We can capture the relationships with a set of hyper-edges $\mathcal{H} = \{h_i\}$. Each hyper-edge h may have attributes as well to capture the attributes of relationships, which we denote $h.A_1, h.A_2, \ldots, h.A_l$, and we use $h.R$ to denote the set of references that it connects. In our example, each rectangle denotes one hyper-edge corresponding to one paper in the database. The first hyper-edge corresponding to *Paper*1 has as its attribute the title "A mouse immunity model" and connects the three references having name attributes "W. Wang," "C. Chen," and "A. Ansari." A reference r can belong to zero or more hyper-edges and we use $r.H$ to denote the set of hyper-edges in which r participates. For example, if we have paper, author, and venue references, then a paper reference may be connected to multiple author references and also to a venue reference. In general, the underlying references can refer to entities of different types, as in a publication database, or in newspaper articles, which contain references to people, places, organizations, etc. When the type information is known for each reference, resolution decisions are restricted within references of the same type. Otherwise, the types may need to be discovered as well as part of the entity resolution process.

Traditional entity resolution approaches pose entity resolution as a pair-wise decision problem over references based on their attribute similarity. It can also be posed as a ►graph clustering problem, where references are clustered together based on their attribute similarities and each cluster is taken to represent one underlying entity. Entity resolution approaches differ in how the similarities between references are defined and computed and how the resolution decisions are made based on these similarities. Traditionally, each pair-wise decision is made independently of the others. For example, the decision to resolve the two *Wang* references from papers 1 and 3 would be made independently of the decision to resolve the two *Chen* references from the same papers.

The first improvement is to account for the similarity of the coauthor names when such relationships are available. However, this still does not consider the "entities" of the related references. For the two "Wang" references in the earlier example, the two "C. Chen" coauthors match regardless of whether they refer to *Chen1* or *Chen2*. The correct evidence to use here is that the "Chen's" are not co-referent. In such a setting, in order to resolve the "W. Wang" references, it is necessary to *resolve* the "C Chen" references as well, and not just consider their name similarity. In the collective relational entity resolution approach, resolutions are not made independently, but instead one resolution decision affects other resolutions via hyper-edges.

Below, we discuss the different entity resolution approaches in greater detail.

Attribute-Based Entity Resolution

As discussed earlier, exact matching of attributes does not suffice for entity resolution. Several sophisticated similarity measures have been developed for textual strings (Cohen, Ravikumar, & Fienberg, 2003; Chaudhuri, Ganjam, Ganti, & Motwani, 2003) that may be used for unsupervised entity resolution. Finally, a weighted combination of the similarities over the different attributes for each reference is used to compute the attribute similarity between two references. An alternative is to use adaptive supervised algorithms that learn string ►similarity metrics from labeled data (Bilenko & Mooney, 2003). In the traditional entity resolution approach (Fellegi & Sunter, 1969; Cohen et al., 2003), similarity is computed for each pair of references r_i, r_j based on their attributes and only those pairs that have similarity above some threshold are considered co-referent.

Efficiency

Even the attribute-only approach to entity resolution is known to be a hard problem computationally, since it is infeasible to compare all pairs of references using expensive similarity measures. Therefore, efficiency issues have long been a focus for data cleaning, the goal being the development of inexpensive algorithms for finding approximate solutions. The key mechanisms for doing this involve computing the matches efficiently and employing techniques commonly called "blocking" to quickly find potential duplicates (Hernández & Stolfo, 1995; Monge & Elkan, 1997), using cheap and index-based similarity computations to rule out non-duplicate pairs. Sampling approaches can quickly compute cosine similarity between tuples for fast text-joins within an SQL framework (Gravano, Ipeirotis, Koudas, & Srivastava, 2003). Error-tolerant indexes can also be used in data warehousing applications to efficiently look up a small but "probabilistically safe" set of reference tuples as candidates for matching for an incoming tuple (Chaudhuri et al., 2003). Generic entity resolution frameworks also exist for resolving and merging duplicates as a database operator and minimize the number of record-level and feature-level operations (Menestrina, Benjelloun, & Garcia-Molina, 2006).

Probabilistic Models for Pairwise Resolution

The groundwork for posing entity resolution as a probabilistic ▸classification problem was done by Fellegi and Sunter (1969), who studied the problem of labeling pairs of records from two different files to be merged as "match" (M) or "non-match" (U) on the basis of agreement γ among their different fields or attributes. Given an agreement pattern γ, the conditional probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of γ given matches and non-matches are computed and compared to decide whether the two references are duplicates or not. Fellegi and Sunter showed that the probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of field agreements can be estimated without requiring labeled training data if the different fields agreements are assumed to be independent. Winkler (2002) used the EM algorithm to estimate the probabilities without making the independence assumption.

Probabilistic Models for Relational Entity Resolution

Probabilistic models that take into account interaction between different entity resolution decisions through hyper-edges have been proposed for named-entity recognition in natural language processing and for citation matching (McCallum & Wellner, 2004; Singla & Domingos, 2004). Such ▸relational learning approaches introduce a decision variable y_{ij} for every pair of references r_i and r_j, but instead of inferring the y_{ij}'s independently, use conditional random fields for joint reasoning. For example, the decision variables for the "Wang" references and the "Chen" references in papers 1 and 3 would be connected to each other features functions would be defined to ensure that they are more likely to take up identical values.

Such relational models are supervised and require labeled data to train the parameters. One of the difficulties in using a supervised method for resolution is constructing a good training set that includes a representative collection of positive and negative examples. Accordingly, unsupervised relational models have also been developed (Bhattacharya & Getoor, 2006; Li, Morie, & Roth, 2005; Pasula, Marthi, Milch, Russell, & Shpitser, 2003). Instead of introducing pairwise decision variables, this category of approaches use generative models for references using latent entity labels. Note that, here, the number of entities is unknown and needs to be discovered automatically from the available references. Relationships between the references, such as co-mentions or co-occurrences, are captured using joint distributions over the entity labels.

All of these probabilistic models have been shown to perform well in practice and have the advantage that the match/non-match decisions do not depend on any user-specified similarity measures and thresholds but are learned directly from data. However, this benefit comes at a price. Inference in relational probabilistic models is an expensive process. Exact inference is mostly intractable and approximate strategies such as loopy belief propagation and Monte Carlo sampling strategies are employed. Even these approximate strategies take several iterations to converge and extending such approaches to large datasets is still an open problem.

Other Approaches for Relational Entity Resolution

Alternative approaches (Dong, Halevy, & Madhavan, 2005; Bhattacharya & Getoor, 2007; Kalashnikov, Mehrotra, & Chen, 2005) consider relational structure of the entities for data integration but avoid the complexity of probabilistic inference. By avoiding a formal probabilistic model, these approaches can handle complex and longer-range relationships between different entity references and the resolution process is significantly faster as well. Such approaches also create pairwise decision nodes between references and create a dependency graph over them to capture the relationships in the data. But instead of performing probabilistic inference, they keep updating the value associated with each decision node by propagating relational evidence from one decision node to another over the dependency graph.

When the relationships between the references and the entities can be captured in a single graph, the matching entity for a specific reference may be identified using path-based similarities between their corresponding nodes in the graph. The connection strength associated with each edge in the graph can be determined in the unsupervised fashion given all the references, their candidate entity choices, and the relationships between them, by solving a set of nonlinear equations (Kalashnikov et al., 2005). This approach is useful for incremental data cleaning when the set of entities currently in the database is known and an incoming reference needs to be matched with one of these entities.

An alternative approach to performing collective entity resolution using relational evidence is to perform collective relational clustering (Bhattacharya & Getoor, 2007). The goal here is to cluster the references into entities by taking into account the relationships between the references. This is achieved by defining a similarity measure between two clusters of references that take into account not only the attribute similarity of the references in the two clusters, but also the neighboring clusters of each cluster. The neighboring clusters of any reference cluster c are defined by considering the references r' connected to references r belonging to c via hyper-edges, and the clusters to which these related references belong. If the $r.C$ represents the current cluster for reference c, then $N(c) = \bigcup r'.C$, where $r.H = r'.H$ and $r.C = c$. For instance, the neighboring clusters for a *Wang* cluster in our example containing the *Wang* references from papers 1,2 and 4 are the *Ansari* cluster and the *Chen* clusters containing the other references from the same papers. The relational similarity between two clusters is then computed by comparing their neighborhoods. This relational similarity complements attribute similarity in the combined similarity between two clusters. Intuitively, two entities are likely to be the same if they are similar in attributes and are additionally connected to the same other entities. Collective relational clustering can be efficiently implemented by maintaining a priority queue for merge-able cluster pairs and updating the "neighboring" queue elements with every merge operation.

Applications

Data cleaning and reference disambiguation approaches have been applied and evaluated in a number of domains. The earliest applications were on medical data. Census data is an area where detection of duplicates poses a significant challenge and Winkler (Winkler, 2002) has successfully applied his research and other baselines to this domain. A great deal of work has been done making use of bibliographic data (Pasula et al., 2003; Singla & Domingos, 2004; Bhattacharya & Getoor, 2007). Almost without exception, the focus has been on the matching of citations. Work in coreference resolution and disambiguating entity mentions in natural language processing (McCallum & Wellner, 2004) has been applied to text corpora and newswire articles like the TREC corpus. There have also been significant applications in information integration in data-warehouses (Chaudhuri et al., 2003).

Cross References

- ►Classification
- ►Data Preparation
- ►Graph Clustering
- ►Similarity Metrics
- ►Statistical Relational Learning

Recommended Reading

Bhattacharya, I., & Getoor, L. (2006). A latent dirichlet model for unsupervised entity resolution. In *The SIAM international conference on data mining* (SIAM-SDM), Bethesda, MD, USA.

Bhattacharya, I., & Getoor, L. (2007). Collective entity resolution in relational data. *ACM transactions on knowledge discovery from data, 1*(1), 5.

Bilenko, M., & Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (KDD-2003), Washington, DC.

Chaudhuri, S., Ganjam, K., Ganti, V., & Motwani, R. (2003). Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on management of data* (pp. 313–324). San Diego, CA.

Cohen, W. W., Ravikumar, P., & Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 workshop on information integration on the web* (pp. 73–78). Acapulco, Mexico.

Dong,X.,Halevy,A.,&Madhavan,J.(2005).Referencereconciliationin complexinformationspaces.In *TheACMinternationalconference on management of data (SIGMOD)*, Baltimore, MD, USA.

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association, 64*, 1183–1210.

Gravano, L., Ipeirotis, P., Koudas, N., & Srivastava, D. (2003). Text joins for data cleansing and integration in an rdbms. In *19th IEEE international conference on data engineering.*

Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD international conference on management of data (SIGMOD-95)* (pp. 127–138). San Jose, CA.

Kalashnikov, D. V., Mehrotra, S., & Chen, Z. (2005). Exploiting relationships for domain-independent data cleaning. In *SIAM international conference on data mining (SIAM SDM)*, April 21–23 2005, Newport Beach, CA, USA.

Li, X., Morie, P., & Roth, D. (2005). Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine. Special issue on semantic integration, 26*(1).

McCallum, A., & Wellner, B. (2004). Conditional models of identity uncertainty with application to noun coreference. In *NIPS*, Vancouver, BC.

Menestrina, D., Benjelloun, O., & Garcia-Molina, H. (2006). Generic entity resolution with data confidences. In *First Int'l VLDB workshop on clean databases*, Seoul, Korea.

Monge, A. E., & Elkan, C. P. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 workshop on research issues on data mining and knowledge discovery* (pp. 23–29). Tuscon, AZ.

Pasula, H., Marthi, B., Milch, B., Russell, S., & Shpitser, I. (2003). Identity uncertainty and citation matching. In *Advances in neural information processing systems* 15. Cambridge, MA: MIT Press.

Singla, P., & Domingos, P. (2004). Multi-relational record linkage. In *Proceedings of 3rd workshop on multi-relational data mining at ACM SI GKDD*, Seattle, WA.

Winkler, W. E. (2002). Methods for record linkage and Bayesian networks. *Technical Report, Statistical Research Division*, U.S. Census Bureau, Washington, DC.

EP

►Expectation Propagation

Epsilon Covers

Thomas Zeugmann
Hokkaido University
Sapparo, Japan

Definition

Let (M,ρ) be a metric space, let $S \subseteq M$, and let $\varepsilon > 0$. A set $E \subseteq M$ is an *ε-cover* for S, if for every $s \in S$ there is an $e \in E$ such that $\rho(s,e) \leq \varepsilon$.

An *ε-cover* E is said to be *proper*, if $E \subseteq S$.

Application

The notion of an ε-cover is frequently used in kernel-based learning methods.

For further information, we refer the reader to Herbrich (2002).

Cross References

►Statistical Machine Learning
►Support Vector Machines

Recommended Reading

Herbrich, R. (2002). *Learning kernel classifiers: Theory and algorithms*. Cambridge, MA: MIT Press.

Epsilon Nets

Thomas Zeugmann
Hokkaido University
Sapparo, Japan

Definition

Epsilon nets were introduced by Haussler and Welz (1987) and their usefulness for computational learning theory has been discovered by Blumer, Ehrenfeucht, Haussler, & Warmuth (1989).

Let $X \neq \emptyset$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class. For the sake of simplicity, we also use $\mathcal{C}$ here as hypothesis space. In order to guarantee that all probabilities considered below do exist, we restrict ourselves to *well-behaved* concept classes (►PAC Learning).

Furthermore, let D be any arbitrarily fixed probability distribution over the learning domain X and let $c \in \mathcal{C}$ be any fixed concept.
A hypothesis $h \in \mathcal{C}$ is said to be *bad* for c iff

$$d(c,h) = \sum_{x \in c \triangle h} D(x) > \varepsilon.$$

Furthermore, we use

$$\Delta(c) =_{df} \{h \triangle c \mid h \in \mathcal{C}\}$$

to denote the set of all possible *error regions* of c with respect to $\mathcal{C}$ and D. Moreover, let

$$\Delta_\varepsilon(c) =_{df} \{h \triangle c \mid h \in \mathcal{C},\ d(c,h) > \varepsilon\}$$

denote the set of all *bad error regions* of c with respect to $\mathcal{C}$ and D.

Now we are ready to formally define the notion of an ε-net.

Definition

Let $\varepsilon \in (0,1)$ and let $S \subseteq X$. The set S is said to be an *ε-net for* $\Delta(c)$ iff $S \cap r \neq \emptyset$ for all $r \in \Delta_\varepsilon(c)$.

Remarks

Conceptually, a set S constitutes an ε-net for $\Delta(c)$ iff every bad error region is hit by at least one point in S.

Example

Consider the one-dimensional Euclidean space $\mathbb{E}$ and let $X = [0,1] \subseteq \mathbb{E}$. Furthermore, let $\mathcal{C}$ be the set of all closed intervals $[a,b] \subseteq [0,1]$. Consider any fixed $c \in \mathcal{C}$ and let D be the uniform distribution, i.e., $D([a,b]) = 1/(b-a)$ for all $[a,b] \in \mathcal{C}$. Furthermore, let $h \in \mathcal{C}$; then we may write $c \triangle h = I_1 \cup I_2$, where $I_1, I_2 \in \mathcal{C}$. Let $\varepsilon \in (0,1)$ be arbitrarily fixed and let

$$S = \{k\varepsilon/2 \mid 0 \le k \le \lceil 2/\varepsilon \rceil,\ k \in \mathbb{N}\}.$$

Then, S forms an ε-net for $\Delta(c)$. This can be seen as follows. Assume $r \in \Delta_\varepsilon(c)$. Then, $D(I_1) > \varepsilon/2$ or $D(I_2) > \varepsilon/2$. Now, by the definition of S it is obvious that $D(I_i) > \varepsilon/2$ implies $I_i \cap S \neq \emptyset$, $i = 1, 2$.

Application

Recall that in ►PAC Learning, the general strategy to design a learner has been to draw a sufficiently large finite sample and then to find a hypothesis that is consistent with it. For showing that this strategy is always successful, the notion of an ε-net plays an important role. This can be expressed by the following observation.

Observation. Let $S = \{x_1, \dots, x_m\}$ be an ε-net for $\Delta(c)$, and let $h \in \mathcal{C}$ be any hypothesis such that $h(x_i) = c(x_i)$ for all $1 \le i \le m$, i.e., h is consistent. Then we have $d(c,h) \le \varepsilon$.

It then remains to show that the ►VC Dimension of $\mathcal{C}$ and of $\Delta(c)$ are the same and to apply Sauer's Lemma to complete the proof.

For further information, we refer the reader to Blumer, Ehrenfeucht, Haussler, & Warmuth (1989) as well as to Kearns and Vazirani (1994).

Cross References

►PAC Learning
►VC Dimension

Recommended Reading

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, *36*(4), 929–965.
Haussler, D., & Welz, E. (1987). Epsilon nets and simplex range queries. *Discrete & Computational Geometry*, *2*, 127–151.
Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge, MA: MIT Press.

Equation Discovery

Ljupčo Todorovski
University of Ljubljana
Ljubljana, Slovenia

Synonyms

Computational discovery of quantitative laws; Symbolic regression

Definition

Equation discovery is a machine learning task that deals with the problem of learning quantitative laws and models, expressed in the form of equations, in collections of measured numeric data. Equation discovery methods take at input a ►data set consisting of measured values of a set of numeric variables of an observed system or phenomenon. At output, equation discovery methods provide a set of equations, such that, when used to calculate the values of system variables, the calculated values closely match the measured ones.

Motivation and Background

Equation discovery methods can be used to solve complex modeling tasks, i.e., establishing a mathematical model of an observed system. Modeling tasks are omnipresent in many scientific and engineering domains.

Equation discovery is strongly related to *system identification*, another approach to mathematical modeling. System identification methods work under the assumption that the structure of the model (the form of the model equations) is known or comes from a well-defined class of model structures, such as polynomials or neural networks. Therefore, they are mainly concerned with the parameter estimation task, that is, the task of determining the values of the model parameters that minimize the discrepancy between measured data and data obtained by simulating the model. Equation discovery methods, on the other hand, aim at identifying both, an adequate structure of the model equations and appropriate values of the model parameters.

►Regression also deals with building predictive models from numeric data. The focus of regression methods is on building descriptive black-box models that can reconstruct the training data with high accuracy. In contrast, equation discovery methods focus on establishing explanatory models that, beside accurate predictions, provide explanations of the mechanisms that govern the behavior of the modeled system.

Early equation discovery methods dealt with rediscovering empirical laws from the history of science (this is where the synonym "computational discovery of quantitative laws" comes from). Through the years, the focus of the equation discovery methods has shifted from discovering quantitative laws to modeling real-world systems.

Structure of the Learning System

The task of equation discovery can be decomposed into two closely coupled subtasks of structural identification and parameter estimation. The first task of structural identification deals with the problem of finding the optimal structure of an equation. The second task of parameter estimation deals with the problem of finding the optimal values of the constant parameters in the equation. General approaches to and specific methods for equation discovery use different techniques to solve these two subtasks.

Approaches and Methods

There are two general and fundamentally different approaches to equation discovery. The first approach relies on a definition of a space of candidate equation structures. Following this definition, a generate-and-test (or ►learning as search) approach is used to generate different equation structures, solve the parameter estimation task for each of them, and report those equations that most closely approximate the data. The second approach relies on heuristics, used by scientists and engineers in the discovery or modeling processes, to establish an appropriate equation structure.

The first equation discovery system, Bacon (Langley, 1981), follows the second approach described above. It incorporates a set of data-driven heuristics for detecting regularities (constancies and trends) in measured data and for formulating hypotheses based on them. An example heuristic would, when faced with a situation where the values of two observed variables increase/decrease simultaneously, introduce a new equation term by multiplying them. Furthermore, Bacon builds equation structure at different levels of description. At each level of description, all but two variables are held constant and hypotheses connecting the two changing variables are considered. Using a relatively small set of data-driven heuristics, Bacon is able to rediscover a number of physical laws including the ideal gas law, the law of gravitation, the law of refraction, and Black's specific heat law.

An alternative set of heuristics for equation discovery can be derived from dimensional analysis that is routinely used to check the plausibility of equations by using rules that specify the proper ways to combine variables and terms with different *measurements units*, different measurement scales, or types thereof. Following these rules, equation discovery method Coper (Kokar, 1986) considers only equation structures that properly combine variables and constants, given the knowledge about their exact measurement units. Equation discovery method SDS (Takashi & Hiroshi, 1998) extends Coper to cases, where the exact measurement units of the variables and constants involved in the equation are not known, but only knowledge about the types of the ▶measurement scales is available.

Finally, the heuristics and design of the equation discovery method E* (Schaffer, 1993) is based on a systematic survey of more than a hundred laws and models published in the Physical Review journal. The review shows that many of the published laws and models follow one of five different equation structures. By including only these five structures as its main heuristic for solving the structure identification task (implementing it as a ▶language bias), E* was able to reconstruct the correct laws and models in about a third of the test cases collected from the same journal.

Abacus (Falkenhainer & Michalski, 1990) was the first equation discovery method that followed the generate-and-test (or ▶learning as search) approach, mentioned above. Abacus experimented with different search strategies within a fixed space of candidate equation structures. Other methods that follow the generate-and-test approach differ in the ways they define the space of candidate equation structures and solve the parameter estimation task.

Equation discovery methods EF (Zembowitz & Zytkow, 1992) and Lagrange (Džeroski & Todorovski, 1995) explore the space of polynomial equation structures that are linear in the constant parameters, so they apply ▶linear regression to estimate parameters. The user can shape the space of candidate structures by specifying parameters, such as, the maximal polynomial degree, the maximal number of multiplicative terms included in a polynomial, and a set of functions that can be used to transform the original variables before combining them into multiplicative terms.

While all of the above methods assume a fixed predefined ▶language bias (via specification of the class of candidate equation structures or via heuristics for establishing appropriate structure), equation discovery method Lagramge (Todorovski & Džeroski, 1997) employs dynamic declarative ▶language bias, that is, let the user of the equation discovery method choose or specify the space of candidate equation structures. In its first version, Lagramge uses the formalism of context-free grammars for specifying the space of equation structures. The formalism has been shown to be general enough to allow users to build their specification upon many different types of modeling knowledge, from measurement units to very specific knowledge about building models in a particular domain of interest (Todorovski & Džeroski, 2007). For solving the structure identification task, Lagramge defines a refinement operator that orders the search space of candidate equation structures, defined by the user-specified grammar, from the simplest ones to more complex. Exhaustive and ▶beam search strategies are then being employed to the search space and for each structure considered during the search, Lagramge uses gradient-descent methods for nonlinear optimization to solve the parameter estimation task. The heuristic function that guides the search is based on the ▶mean squared error that measures the discrepancy between the measured and simulated values of the observed system variables. Alternatively, Lagramge can use heuristic function that takes into account the complexity of the equation and is based on the ▶minimum description length principle.

Successors of Lagramge, equation discovery methods, Lagramge 2 (Todorovski & Džeroski, 2007), IPM (Bridewell, Langley, Todorovski, & Džeroski, 2008), and HIPM (Todorovski, Bridewell, Shiran, & Langley, 2005), primarily focus on the improvement of the knowledge representation formalism used to formalize the modeling knowledge and transform it to ▶language bias for equation discovery. All of them follow the paradigm of ▶inductive process modeling.

Types of Equations

At first, equation discovery methods dealt with the problem of learning algebraic equations from data. Equation discovery method Lagrange (Džeroski & Todorovski, 1995) extended the scope of equation discovery to modeling dynamics from ▶time series data with ordinary differential equations. It took a naïve approach based on transforming the task of discovering ordinary differential equations to the simpler task of discovering algebraic equations, by extending the set of observed system variables with numerically calculated time derivatives thereof. By doing so, any of the existing equation discovery methods could be, in principle, used to discover differential equations. However, the naïve approach has a major drawback of introducing large numerical errors, due to instability of methods for numerical differentiation. Equation discovery method GoldHorn (Križman, Džeroski, & Kompare, 1995) replaced the instable numerical differentiation with the stable numerical methods for the inverse problem of integration. Goldhorn also upgrades Lagrange with filtering methods to cope with measurement errors and noisy data.

While ordinary differential equations can model systems that change their state along a single dimension, time, partial differential equations can be used to model systems that change along many (temporal and spatial) dimensions. The naïve approach of introducing numerically calculated partial derivatives has been used in the Paddles (Todorovski, Džeroski, Srinivasan, Whiteley, & Gavaghan, 2000) method for discovery of partial differential equations. The method first slices the measurement data into narrow spatial subsets, induces ordinary differential equations in each of them, and uses most frequently obtained equation structures to extend them with partial derivatives and to obtain a relatively small class of partial differential equation structures to explore. All the equation discovery tasks in Paddles are solved using Lagramge (Todorovski & Džeroski, 1997).

Applications

Equation discovery methods have been applied to various tasks of discovering equation-based laws and models from measured and/or simulation data. Application domains range from physics (mechanical and electrical engineering, fluid dynamics) (Takashi & Hiroshi, 1998; Todorovski & Džeroski, 1997, 2007), through ecology (population dynamics) (Todorovski & Džeroski, 2007; Todorovski et al., 2005) to biochemistry (chemical kinetics) (Džeroski & Todorovski, 2008; Langley, Shiran, Shrager, Todorovski, & Pohorille, 2006).

Cross References

►Inductive Process Modeling
►Language Bias
►Learning as Search
►Linear Regression
►Measurement Scales
►Regression
►System Identification

Recommended Reading

Bridewell, W., Langley, P., Todorovski, L., & Džeroski, S. (2008). Inductive process modeling. *Machine Learning, 71*(1), 1–32.

Džeroski, S., & Todorovski, L. (1995). Discovering dynamics: From inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, *4*(1), 89–108.

Džeroski, S., & Todorovski, L. (2008). Equation discovery for systems biology: Finding the structure and dynamics of biological networks from time course data. *Current Opinion in Biotechnology*, *19*, 1–9.

Falkenhainer, B., & Michalski, R. (1990). Integrating quantitative and qualitative discovery in the ABACUS system. In Y. Kodratoff & R. Michalski (Eds.), *Machine learning: An artificial intelligence approach*. San Mateo: Morgan Kaufmann.

Kokar, M. M. (1986). Determining arguments of invariant functional descriptions. *Machine Learning*, *1*(4), 403–422.

Križman, V., Džeroski, S., & Kompare, B. (1995). Discovering dynamics from measured data. *Electrotechnical Review*, *62*(3–4), 191–198.

Langley, P. (1981). Data-driven discovery of physical laws. *Cognitive Science*, *5*(1), 31–54.

Langley, P., Shiran, O., Shrager, J., Todorovski, L., & Pohorille, A. (2006). Constructing explanatory process models from biological data and knowledge. *Artificial Intelligence in Medicine*, *37*(3), 191–201.

Schaffer, C. (1993). Bivariate scientific function finding in a sampled, real-data testbed. *Machine Learning*, *12*(1–3), 167–183.

Takashi, W., & Hiroshi, M. (1998). Discovery of first-principle equations based on scale-type-based and data-driven reasoning. *Knowledge-Based Systems*, *10*(7), 403–411.

Todorovski, L., Bridewell, W., Shiran, O., & Langley, P. (2005). Inducing hierarchical process models in dynamic domains. In M.M. Veloso & S. Kambhampati (Eds.), *Proceedings of the twentieth national conference on artificial intelligence*, Pittsburgh, PA, USA.

Todorovski, L., & Džeroski, S. (1997). Declarative bias in equation discovery. In D.H. Fisher (Ed.), *Proceedings of the fourteenth international conference on machine learning*, Nashville, TN, USA.

Todorovski, L., & Džeroski, S. (2007). Integrating domain knowledge in equation discovery. In S. Džeroski & L. Todorovski (Eds.), *Computational discovery of scientific knowledge. LNCS* (Vol. 4660). Berlin: Springer.

Todorovski, L., Džeroski, S., Srinivasan, A., Whiteley, J., & Gavaghan, D. (2000). Discovering the structure of partial differential equations from example behaviour. In P. Langley (Ed.), *Proceedings of the seventeenth international conference on machine learning*, Stanford, CA, USA.

Zembowitz, R., & Zytkow, J. (1992). Discovery of equations: Experimental evaluation of convergence. In W. R. Swartout (Ed.), *Proceedings of the tenth national conference on artificial intelligence*, San Jose, CA, USA.

Error

►Error Rate

Error Correcting Output Codes

Synonyms
ECOC

Definition
Error correcting output codes are an ►ensemble learning technique. It is applied to a problem with multiple classes, decomposing it into several binary problems. Each class is first encoded as a binary string of length T, assuming we have T models in the ensemble. Each model then tries to separate a subset of the original classes from all the others. For example, one model might learn to distinguish "class A" from "not class A." After the predictions, with T models we have a binary string of length T. The class encoding that is closest to this binary string (using Hamming distance) is the final decision of the ensemble.

Recommended Reading
Kong, E. B., & Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. In *International conference on machine learning*.

Error Curve

►Learning Curves in Machine Learning

Error Rate

Kai Ming Ting

Synonyms
Error

Definition
Error rate refers to a measure of the degree of prediction error of a ►model made with respect to the true model.

The term *error rate* is often applied in the context of ►classification models. In this context, *error rate* = $P(\lambda(X) \neq Y)$, where XY is a joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

Two common measures of *error rate* for ►regression models are ►mean squared error and ►mean absolute error.

The error rate of a model is often assessed or estimated by applying it to test data for which the ►class labels (Y values) are known. The error rate of a classifier on test data may be calculated as *number of incorrectly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ►Laplace estimate or an ►*m*-estimate.

Error rate is directly related to ►accuracy, such that *error rate* = 1.0 − *accuracy* (or when expressed as a percentage, *error rate* = 100 − *accuracy*).

Cross References
►Accuracy
►Confusion matrix
►Mean absolute error
►Mean squared error

Error Squared

Synonyms
Squared error

Definition
Error squared is a common ►loss function used with ►regression. This is the square of the difference between the predicted and true values.

Estimation of Density Level Sets

►Density-Based Clustering

Evaluation

Evaluation is a process that assesses some property of an artifact. In machine learning, two types of artifacts are most commonly evaluated, ►models and algorithms. ►Model evaluation often focuses on the predictive efficacy of the model, but may also assess factors such as its complexity, the ease with which it can be understood, or the computational requirements for its application. ►Algorithm evaluation often focuses on evaluation of the models an algorithm produces, but may also appraise its computational efficiency.

Evaluation Data

►Test Data
►Test Set

Evaluation Set

►Test Set

Evolution of Agent Behaviors

►Evolutionary Robotics

Evolution of Robot Control

►Evolutionary Robotics

Evolutionary Algorithms

Synonyms

Evolutionary computation; Evolutionary computing; Genetic and evolutionary algorithms

Definition

Generic term subsuming all machine learning and optimization methods inspired by neo-Darwinian evolution theory.

Cross References

►Coevolutionary Learning
►Compositional Coevolution
►Evolutionary Clustering
►Evolutionary Computation in Economics
►Evolutionary Computation in Finance
►Evolutionary Computational Techniques in Marketing
►Evolutionary Feature Selection and Construction
►Evolutionary Fuzzy Systems
►Evolutionary Games
►Evolutionary Kernel Learning
►Evolutionary Robotics
►Neuroevolution
►Nonstandard Criteria in Evolutionary Learning
►Test-Based Coevolution

Evolutionary Clustering

David Corne[1], Julia Handl[2], Joshua Knowles[2]
[1]Heriot-Watt University, Edinburgh, UK
[2]University of Manchester

Synonyms

Cluster optimization; Evolutionary grouping; Genetic clustering; Genetic grouping

Definition

Evolutionary clustering refers to the application of ►evolutionary algorithms (also known as genetic algorithms) to data ►clustering (or cluster analysis), a general class of problems in machine learning, with numerous applications throughout science and industry. Different definitions of data clustering exist, but it generally concerns the identification of homogeneous groups of data (clusters) within a given data set. That is, data items that are similar to each other should be grouped together in the same *cluster* or *group*, while (usually) dissimilar items should be placed in separate clusters. The output of any clustering method is therefore a specific collection of clusters. If we have a specific way to evaluate (calculate the quality of) a given grouping into clusters, then we can consider the clustering task as an optimization problem. In general, this optimization problem is NP hard, and it is common to address it with advanced heuristic or metaheuristic methods. Evolutionary algorithms are prominent among such methods, and have led to a variety of promising and successful techniques for cluster optimization.

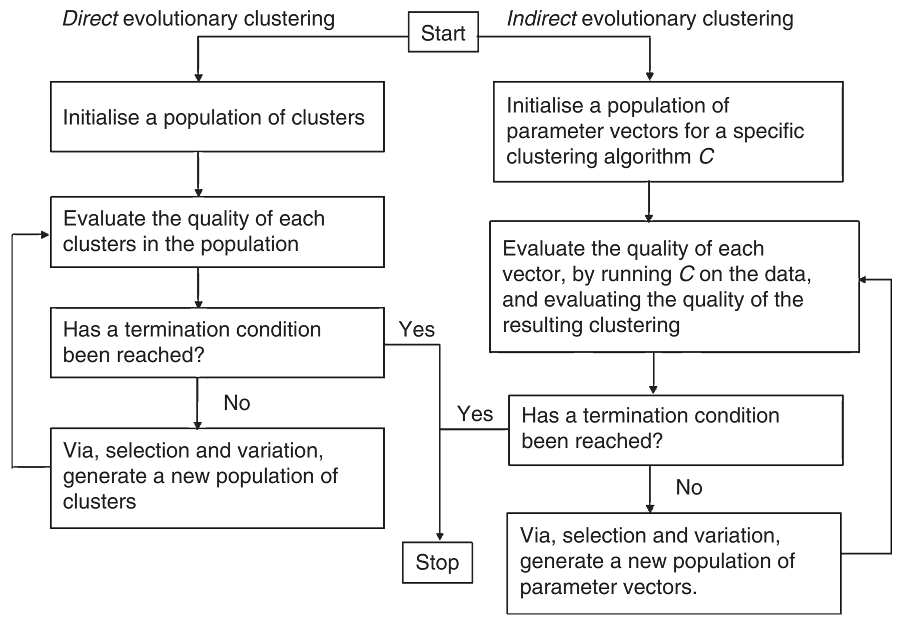
Motivation and Background

In many problem-solving scenarios, we have large amounts of data. We need to cluster those data sensibly into groups in order to help us understand the problem and decide how to proceed further (see ►clustering). It is common, in fact, for this initial "cluster analysis" stage to be the most important (or only) stage in the investigation. In bioinformatics, for example, a frequent activity is the clustering of gene expression data (data that indicate, for a specific cell, how active each of several thousands of genes are at different points in time, or under different experimental conditions). A very important current challenge is to understand the role of each gene; by clustering such data, which means arranging genes into groups such that genes in the same group have similar patterns of activity, we find important clues about genes whose role is currently unknown, simply by assigning their putative role as being related to that of genes (whose role is known) that are in the same cluster. Meanwhile, a ubiquitous situation in industry and commerce is the clustering of data about customers or clients. Here, the role of clustering is all about identifying what types of clients (for example, based on age, income, postcode, and many other attributes that may make up a customer's profile) buy or use certain kinds of products and services. Effective ways to identify groups enable companies to better target their products and their direct marketing campaigns, and/or make more effective decisions about loans, credit and overdrafts. Many machine learning techniques can be used to predict things about customers, or predict things about genes, and so forth. However, the value of clustering (in a similar way to visualization of the data) is that it can lead to a much deeper understanding of the data, which in turn informs the continuing process of applying machine learning methods to it. In this general context, there are many well-known and well-used clustering methods, such as k-means, hierarchical agglomerative clustering, neighbor-joining, and so forth. However, there are also well-known difficulties with these methods; specifically, there is often a need to choose in advance the number of clusters to find in the data, and: they tend to be strongly biased towards finding certain types of groupings. For these reasons, methods that are more flexible have been recently investigated, and evolutionary clustering techniques are prominent among these. They are flexible in that (unlike k-means, for example), the choice of the number of clusters does not have to be made a priori, and the method is not tied to any particular way of identifying the distance between two items of data, nor is there any a priori ►inductive bias concerning what counts as a good clustering. That is, in broad terms, an evolutionary clustering algorithm allows a user to decide in advance on a definition of cluster quality that is suitable for the problem at hand, and to decide in advance how many clusters are sought, or to leave that decision open; these decisions are then "plugged in to" the algorithm which then proceeds to search for good clusterings.

Structure of Learning System

Evolving Clusters and Evolving Clustering Algorithms

Given a dataset to be clustered, the concept of evolutionary clustering covers two distinct ways in which we can address the problem of finding the best clustering. Each of these approaches is under continuing research, and has proven successful under different conditions. The first approach is to use an evolutionary algorithm to search the space of candidate groupings of the data; this is the most straightforward approach, and perhaps the most flexible in the sense discussed above. The second approach is to "wrap" an evolutionary algorithm around a simpler clustering algorithm (such as k-means), and either use the evolutionary algorithm to search the space of features for input to the clustering algorithm (i.e., the evolutionary algorithm is doing ►feature selection in this case), or to search a space of parameters, such as the number of clusters, feature weights, and/or other parameters of the clustering algorithm in use. Central in all of these approaches is a way to measure the quality of a clustering, which in turn depends on some given metric that provides a distance between any pair of data items. Although some applications often come with pre-identified ways to measure distance and cluster quality, in the following we will assume the most common approach, in which distance is the Euclidean distance between the data items (perhaps Hamming distance, in cases where the data are not numeric), and the measure of quality for a given clustering is the ratio of *within-cluster* and *between-cluster*, where *within-cluster* is the mean distance between pairs of items that are in the same cluster, and *between-cluster*


Direct evolutionary clustering
Start
Indirect evolutionary clustering
Initialise a population of clusters
Evaluate the quality of each clusters in the population
Has a termination condition been reached?
Yes
No
Via, selection and variation, generate a new population of clusters
Stop
Initialise a population of parameter vectors for a specific clustering algorithm *C*
Evaluate the quality of each vector, by running *C* on the data, and evaluating the quality of the resulting clustering
Has a termination condition been reached?
Yes
No
Via, selection and variation, generate a new population of parameter vectors.

Evolutionary Clustering. Figure 1. The two main approaches to evolutionary clustering; direct (*left*) and indirect (*right*)

is the mean distance between pairs of items that are in different clusters.

We illustrate the two main approaches to evolutionary clustering in Fig. 1.

On the left in Fig. 1, we see the direct approach, in which the evolutionary algorithm searches the space of clusterings of the data. The key features in this approach are the encoding and ►genetic operators. After evaluating the quality of each of a population of clusterings, a new population is generated from the old one via selection and variation. Essentially, some individuals from the current population are treated as "parents," and new ones are produced from these by using genetic operators. The encoding specifies precisely how a specific data clustering is represented; while the operators specify how new clusterings are derived from the old ones. To take a simple example, suppose we needed to cluster 10 items (A, B, C,…, J) into an arbitrary number of groups. In a simple encoding, we might represent a clustering as a vector of 10 labels, independently chosen from 1 to 10, in which the ith element gives the group label of the ith item. Hence, the following individual in our population of clusterings:

2 3 5 5 1 5 7 3 2 7

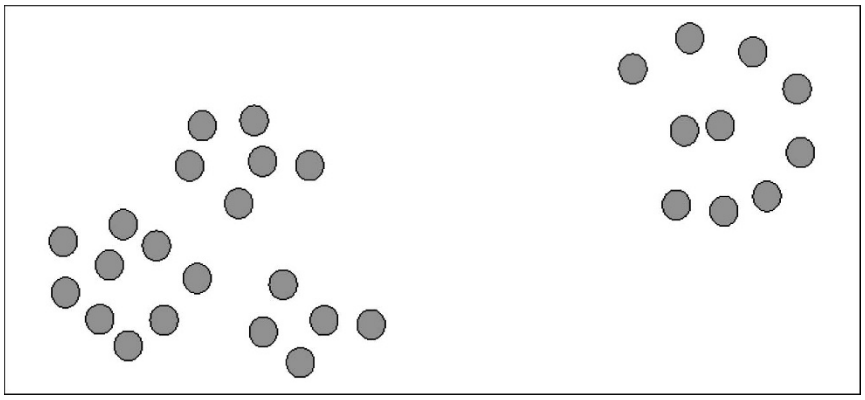
represents the following grouping:

(A, I) (B, H) (C, D, F) (E) (G, I)

Given such a representation, a typical genetic operator might be to randomly change a single label in a single parent. For example, we may choose the fifth element in the above vector and change it randomly to 7, effectively placing item E in the same group as items G and I. Further notes about operators for this and other encodings are given in a special subsection below.

There are several examples of the second type of approach, called "indirect" evolutionary clustering in the Fig. 1 (right). This approach is often used where the "internal" clustering method ("C," in the figure) is very sensitive to initialization conditions and/or parameters of the metric in use to measure distance between items. For example, if C is the k-means algorithm, then, for each application of C, we need choices for the parameter k, and for each of k initial cluster center positions in the data space. The parameter vectors referred to in the figure would be precisely these; the evolutionary algorithm searches this parameter space, finding those that lead to an optimized clustering from k-means.

Figure 2 illustrates why this will often be a more effective approach than k-means alone. In this case, it is entirely unclear whether these data form two, four, or even five clusters. There are two widely separated groups of points, and this two-cluster solution may be


Evolutionary Clustering. Figure 2. **An example with many potential interpretations of the number of clusters**

easily found by a 2-means algorithm. However, to the human eye there is also a clear four-cluster solution, further analysis of which may lead to better understanding of these data. This four-cluster solution is difficult for a 4-means algorithm to find, depending on very fortunate initial settings for the cluster centers. Meanwhile, it is worth noting that there are potentially five clusters, as the group on the right can be perceived as a central group of two items, surrounded by a single backward-C-shaped group. The "backward-C" cluster is an example that simply cannot be reliably detected (*as a distinct cluster from the group of two items contained within it*), with most standard cluster analysis methods. Traditional approaches invariably incorporate the assumption that clusters will be centered around a particular position, with the likelihood of a point belonging to that cluster depending monotonically on distance from that position. However, on of the strengths of evolutionary clustering is that it provides the flexibility to work effectively with arbitrary definitions of what may constitute a valid cluster.

Encodings and Operators for Evolutionary Clustering

The more frequently researched style of evolutionary clustering is the direct approach, and the development of this approach in recent years is essentially characterized by certain key ideas for the encoding method. Encodings range from the straightforward representation noted above (with the *i*th gene coding for the cluster membership of the *i*th data item), to more complex representations, such as matrix-based or permutation-based representations.

Before providing a brief description of other encodings it is worth briefly examining a well-known disadvantage of the simple encoding. Given that they have a population, evolutionary algorithms offer the opportunity to use multi-parent genetic operators – that is, we can design operators that produce a new candidate clustering given two or more "parent" clusterings. Such operators are neither mandatory nor necessarily beneficial in evolutionary algorithms, and there is much literature discussing their merits and how this depends on the problem at hand. However, they are often found helpful, especially in cases where we can see some intuitive merit in combining different aspects of parent solutions, resulting in a new solution that seems to have a chance at being good, but which we would have been immensely unlikely to obtain from single-parent operators given the current population. In this context, we can see, as follows, that the opposite seems to be the case when we use standard multi-parent operators with the simple encoding. Suppose the following are both very good clusterings of ten items:

Clustering 1: 1 1 1 1 1 2 2 2 2 2

Clustering 2: 2 2 2 2 2 1 1 1 1 1

Clearly, a good clustering of these items places items 1–5 together, and items 6–10 together, in separate groups. It is also clear, however, that using a standard crossover operator between these two parents (e.g., producing a child by randomly choosing between clusterings for each item in turn) will lead to a clustering that mixes items from these two groups, perhaps even combining them all into one group. The main point is that a crossover operation destroys the very relationships between the items that underpinned the fitness of the parents.

One of the more prominent and influential representations for clustering, incorporating a design for far more effective multi-parent operators, was that of Falkenauer's "Grouping Genetic Algorithm," which also provides a general template for the implementation of evolutionary algorithms for grouping problems. The essential element of Falkenauer's method is that multi-parent operators recombine entire groups rather than item labels. For example, suppose we encode two clusterings explicitly as follows:

Clustering 3: (A, I, B, H) (C, G) (D, E, F, J)

Clustering 4: (A, I, B, H) (C, D, J) (E, F, G)

A Falkenauer-style crossover operator works as follows. First, we randomly choose some entire groups from the first parent and some entire groups from the second parent; the child in this case might then be:

$\underline{\text{(A, I, B, H) (C, G)}}$ (E, F, G)

in which the groups that come from the first parent are underlined. Typically, we will now have some repeated items; we remove the entire groups that contain these items and came from the first parent, in this case leaving us with:

$\underline{\text{(A, I, B, H)}}$ (E, F, G)

The final step is to add back the missing items, placing them one by one into one of the existing groups, or perhaps forming one or more new groups. The application in hand will often suggest heuristics to use for this step. In clustering, for example, we could make use of the mean Euclidean distance from items in the groups so far. Whatever the end result in this case, note that the fact that A, I, B, and H were grouped together in both parents will be preserved in the child. Similarly, the E, F, G grouping is inherited directly from a parent.

A more recent and effective approach, specifically for clustering, is one first proposed in Park and Song (1998) called a link-based encoding. In this approach, the encoding is simply a list of item indices, and is interpreted as follows. If the *i*th element in the permutation is *j*, then items *i* and *j* are in the same group. So, for example,

B C E E A E G C B G

represents the following grouping:

(A, B, C, D, E, H, I) (F, G, J)

Standard crossover operators may be used with this encoding, causing (intuitively) a reasonable degree of exploration of the space of possible clusterings, yet preserving much of the essential "same-group" relationships between items that were present in the parents. In Handl and Knowles (2007) it is shown why this encoding is effective compared with some alternatives.

We also briefly note other encodings that have been prominent in the history of this subfield. An early approach was that of Jones and Beltramo, who introduced a "permutation with separators" encoding. In this approach, a clustering is encoded by a permutation of the items to be clustered, with a number of *separators* indicating cluster boundaries. For example, if we have ten items to cluster (A–J) and use S as the separator, the following is a candidate clustering:

A I B H S C G S D E F J

representing the same grouping as that of "Clustering 3" above. Jones and Beltramo offered a variant of this encoding that is a cross between the direct and indirect approaches. In their *greedy permutation* encoding, a clustering is represented by a permutation (with no separator characters), with the following interpretation: the first *k* items in the permutation become the centers of the first *k*clusters. The remaining items, in the order they appear, are added to whichever cluster is best for that item according to the objective function (clustering quality metric) in use.

Evolutionary Multiobjective Clustering

It can be strongly argued that the clustering problem is inherently *multiobjective*, yet most methods employ only a single performance criterion to optimize. In fact, there are at least three groups of criteria commonly used (but usually one at a time) in clustering (both evolutionary clustering and other methods). These are: compactness, connectedness, and spatial separation. When an algorithm optimizes for compactness, the idea is that clusters should consist of highly homogeneous data items only – that is, the distance (or other measure of variation) between items in the same cluster should be small. In contrast, if we optimize the degree of connectedness, then we are increasing the extent to which neighboring data items should share the same cluster. This can deal with arbitrarily-shaped clusters, but can lack robustness when there is little spatial separation between clusters. Finally, spatial separation is usually used as a criterion in combination with compactness, or with a measure of the balance of cluster sizes.

In *multiobjective clustering*, the idea is to explicitly explore the solutions that are trade-offs between the conflicting criteria, exploiting the fact that these trade-off solutions are often the ones that most appeal as intuitively "correct" solutions to a clustering problem. Handl and Knowles make use of Park and Song's link-based encoding in their multiobjective evolutionary algorithm, MOCK, which treats a clustering problem as a two-objective problem, using measures of compactness and connectedness for the two objectives. MOCK's multiobjective search process is based on the PESA-II evolutionary multiobjective optimizer (Corne, Jerram, Knowles & Oates, 2001). Following use of MOCK for a clustering problem, an intermediate result (inherent in multiobjective optimization methods) is a (possibly large) collection of different clusterings. These will range from clusterings that score very well on compactness but poorly on connectedness, through to clusterings that achieve excellent connectedness at the expense of poor compactness. It is useful to note that the number of clusters tends to increase as we go from poor connectedness to high-connectedness clusters. Arguably, in many applications such a collection of alternative solutions is useful for the decision-maker. Nevertheless, the MOCK approach incorporates an automated model-selection process that attempts to choose an ideal clustering from the discovered approximate Pareto front. This process is oriented around the notion of determining the "right" number of clusters, and makes use of Tibshirani, Walther, and Hastie (2001) gap statistic (full details are in Handl & Knowles, 2007). Extensive comparison studies, using a wide variety of clustering problems and comparing with many alternative clustering methods, show consistent performance advantages for the MOCK's approach.

Cross References

►Clustering
►Feature Selection
►Semi-Supervised Learning
►Supervised Learning
►Unsupervised Learning

Recommended Reading

Cole, R. M. (1998). *Clustering with genetic algorithms*. Masters dissertation, Department of Computer Science, University of Western Australia.

Corne, D. W., Jerram, N. R., Knowles, J. D., & Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the GECCO* (pp. 283–290).

Delattre, M., & Hansen, P. (1980). Bicriterion cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*(4), 277–291.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. New York: Wiley.

Handl, J., & Knowles, J. (2007). An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, *11*(1), 56–76.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, *31*(3), 264–323.

Jones, D. R., & Beltramo, M. A. (1991). Solving partitioning problems with genetic algorithms. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the fourth international conference on genetic algorithms* (pp. 442–449). Morgan Kaufmann.

Park, Y.-J., & Song, M.-S. (1998). A genetic algorithm for clustering problems. In *Proceedings of the third annual conference on genetic programming* (pp. 568–575). Morgan Kaufman.

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a dataset via the Gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(2), 411–423.

Evolutionary Computation

►Evolutionary Algorithms

Evolutionary Computation in Economics

Serafín Martínez-Jaramillo, Biliana Alexandrova-Kabadjova,
Alma Lilia García-Almanza,
Tonatiuh Peña Centeno
Bank of Mexico,
Mexico, D.F.

Definition

Evolutionary computation (EC) in economics is an area of knowledge which involves the use of any of the EC techniques, also known as evolutionary algorithms (EAs), in order to approach the topics within the economic sciences. This area of knowledge is different from the Evolutionary Economics field which does not necessarily apply EC techniques to study economic problems. The use of EC in economics pursues different purposes mainly to overcome some of the limitations of the classical economic models and to relax some of the strong assumptions made in such models.

Motivation and Background

Evolutionary computation (EC) is a branch of Machine Learning which is inspired in many forms by the principle of evolution. EC techniques, among many other machine learning techniques, have proven to be quite flexible and powerful tools in many different fields and disciplines. Economics-affine fields are by no means the exception for this widespread use of these evolutionary inspired techniques.

In addition to the undeniable necessity of computing in almost every aspect of our modern lives, numerous problems in economics possess algorithmic nature. Therefore, economists must consider computational complexity as an important analysis tool due to the fact that some of such problems belong to the dislikable class of NP-complete (The NP-complete computational complexity class is a subset of "harder" problems from the NP computational class, which is the set of all the decision problems which can be solved using a Nondeterministic Turing Machine in polynomial time). problems. Having said so, EC has been intensively used as an alternative approach to analytical methods in order to tackle numerous NP-complete problems with relative good success.

The first work in economics (Clarifying: such first work approached a classic game known as the Prisoners' Dilemma), which involved the use of EC dates back to the 1980s, in Axelrod and Hamilton (1981) and Axelrod (1987) the authors used Genetic Algorithms (GAs) to derive strategies for the Iterated Prisoner's Dilemma (IPD). From then, EC techniques in economics had been used in areas such as macroeconomics, econometrics, game theory, auctions, learning and agent-based models. There is even a school of thought in economics known as "Evolutionary Economics" (See for example Witt (2008) for an introduction), whose approach to the study of economics involves concepts in evolution but does not necessarily rely on EC techniques.

Rationality and Learning

One of the most relevant concepts in the economics science is the concept of rationality. This concept is at the core of most of the economic models, since it is frequently assumed that economic agents behave in a fully rational way. Unfortunately, it is not clear if such assumption holds after the irrational behavior observed during the recurrent financial crises.

Herbert A. Simon is probably the best known scientist to claim that "decision-making" under uncertainty is not a fully rational process. He developed his theory based on the concept of "bounded rationality" (Simon, 1957), and he was one of the pioneers in the field of artificial intelligence (AI) as well as a highly reputed psychologist and economist. Later, in Arthur (1991), the author made important contributions to the development of agents with bounded rationality, using computational tools. In addition, recent ideas about rationality from a computer scientist's point of view are found in Tsang (2008). In this context to be more precise about the meaning of bounded rationality, let us quote Herbert A. Simon:

- ... boundedly rational agents experience limits in formulating and solving complex problems and in processing (receiving, storing, retrieving, transmitting) information...

Some other common assumptions behind the classical economic theory are that the participants of the model have *homogeneous preferences* and they *interact globally* (Axtell, 2000). In other words, having limited number of participants in the model, the theorists assume that those individuals exhibit the same preferences and all of them interact with each other. These agents are called "representative agents." Moreover, the analysis is focused only at the point of equilibrium, and aspects such as asymmetric information, imperfect competition and network externalities are not considered.

Departing from the assumption of full rationality and homogeneous expectations, the horizon (and the design issues) opens widely. The modeling of the learning behavior of the agents is a central part of the research agenda in computational economics. Regarding the agents' learning process, in Lucas (1986), the author provided an interpretation of adaptive behavior from the economics point of view:

- In general terms, we view or model an individual as a collection of decision rules (rules that dictate the action to be taken in given situations) and a set of preferences used to evaluate the outcomes arising from particular situation-action combinations. These decision rules are continuously under review and revision; new decision rules are tried and tested against experience, and rules that produce desirable outcomes supplant those that do not.

There are many useful techniques to implement what Lucas defined as adaptive learning, like ►genetic algorithms (GAs), as has been done in Bullard and Duffy (1999), and ►genetic programming (GP) as has been done in Martinez-Jaramillo and Tsang (2009b). GP has been previously described as a suitable way to model economic learning in Edmonds (1999). In Brenner (2006), the author provides us a summary of the available options to model agent behavior and learning in economics.

Nevertheless, the more traditional economists are still reluctant to accept an approach in which there is not a rational expectations type of agent, where instead there are inductive, boundedly rational heterogeneous agents (Arthur, 1994).

Economic and Econometric Models

Two of the most relevant areas in economics are macroeconomics and econometrics. Macroeconomics is the branch of economics which analyzes the national economy and its relations with the international economy. Macroeconomic analysis tries to understand the relationships between the broad sectors of the economy by making use of aggregated economic variables such as inflation, unemployment, interest rates, total output, etc. EC has been used in order to analyze some of such macroeconomic variables, a field which is dominated by econometric analysis. Econometrics is a field within the wider area of economics which involves the use of statistics and its tools for the measurement of relationships postulated by economic theory (Greene, 2003).

Many methods in econometrics involve an optimization process, and it is well known that EC is particularly suitable for optimization problems. Probably one of the first applications of GP in econometrics was done by the creator of GP himself in Koza (1992). Additionally, in Agapie and Agapie (2001) the authors use GAs and simulated annealing (SA) for econometric modeling. They found that the performance of the evolutionary algorithms (EAs) is better than the performance of traditional gradient techniques on the specific models in which they performed the comparison. Finally, Östermark (1999) uses a Hybrid GA in several ill-conditioned econometric and mathematical optimization problems with good results.

In addition to the usage of EC in econometrics, some classical economic models such as the Cobweb model and exchange rate models had been also approached with EC techniques. For instance, in Arifovic (1994) and Chen and Yeh (1996) to approach the Cobweb model, in the former work the author uses GAs, whereas in the latter the authors use GP. Furthermore, Arifovic explores the use of GAs in foreign exchange markets in Arifovic (1996). The GA mechanism developed in such works evolved decision rules that were used to determine the composition of the agents' portfolios in a foreign exchange market. Arifovic made two observations rarely seen in the standard overlapping generations (OLG) model with two currencies. First, she evidenced that the returns and exchanges rates were generated endogenously, and second, she observed that the model's equilibrium dynamics is not stable and shows bounded oscillations (the theoretical model implies a constant exchange rate).

The use of GAs in economic modeling is not restricted to the above mentioned works. In Bullard, Arifovic, and Duffy (1995), the authors studied a version of the growth model in which the physical capital is accumulated in a standard form, but the human capital accumulation is subject to increasing returns. In their model, the agents take two decisions when they are young: how much to save by renting physical capital to the companies and how much to invest in training. Returns on training depend on the average level of the human capital of the economy. The authors introduce the agents' learning by means of GAs. In Marimon, McGrattan, and Sargent (1990), Marimon develops an economic model in which the agents adapt by means of a GA.

Game Theory

Game Theory is a branch of applied mathematics that attempts to model the individual's strategic behavior.

The first study considered to establish the fundamentals of the field is the book “Theory of Games and Economic Behavior” (von Neumann & Morgenstern, 1944). The idea behind this theory is that the success of the individual’s decisions depends on the decisions of others. While originally, the aim of the theory was to study the competition in which the agent does better at another’s expense (zero sum games), now it has been extended to study a wider class of interactions among individuals. Furthermore, it is extensively used in economics, biology, and political science among some other disciplines.

A well-defined mathematical object, the game consists of a set of players and a set of strategies (decisions) available to those players. In addition, for each combination of strategies a specification of payoffs is provided. The aim of the traditional applications of the game theory was to find a Nash equilibrium, a solution concept, in which each player of the game adopts a strategy that is unlikely to be changed. This solution concept was named after John Nash, whose work was published in the early 1950s (Nash, 1950). Nevertheless, it took almost 20 years to fully realize what a powerful tool Nash has created. Nowadays, Game Theory is one of the best established theories in economics and it has been extensively used to model the interactions between the economic agents. However, games typically have many Nash equilibria and one of the main assumptions is that the agents behave in a rational way. In more realistic games, the equilibrium selection problem does not have an easy solution though, and the human behavior observed in real life is frequently irrational.

Given the above mentioned constraints, in order to go further, the Evolutionary Game Theory was originated as an application of the mathematical theory of games to biological contexts (see ►Evolutionary Games). In this field, Maynard Smith is considered to be the first one to define the concept of Evolutionary Stable Strategy in Maynard Smith (1972). Furthermore, the possibility of using computer modeling as an extension of the theory of games was first explored in Axelrod and Hamilton (1981). Since then, computer science has been used in traditional game theory problems, like the strategic behavior of agents in auctions, auction mechanism design, etc. By providing approximate solutions in such complex problems this approach can be useful where analytical solutions have not been found. For instance, the iterative prisoners’ dilemma is one of the most studied games by researchers in computer science (Axelrod, 1987). The prisoners’ dilemma is a classic game that consists of the decision-making process by two prisoners who can choose to cooperate or to defect. In the case that the two prisoners choose to cooperate they get a payoff of three each, in the case that both choose to defect they get a payoff of one each, and in the case that any of them decides to defect and the other to cooperate, the former gets a payoff of five and the latter a payoff of zero. In equilibrium, both players decide to defect despite the fact that would be better for them to cooperate.

Axelrod organized a tournament on the iterated prisoners’ dilemma in which he asked people from game theory and amateurs to provide him with strategies. The surprising result was that a very simple strategy (Tit for Tat) won the tournament (Axelrod, 1987). After the reporting of the results from such tournament, Axelrod was able to provide some mathematical results on how cooperation can emerge in a population of egoists. The previous example clearly illustrates how beneficial was the use of computer science to obtain theoretical results in a problem where analytical methods alone have not delivered the desired outcomes.

Game theory is one of the most important areas in economics because it has applications to many fields, such as corporate decision making, microeconomics, market modeling, public policy analysis, environmental systems, etc. We can find more applications of EC to game theory than the IPD. For example, another work related to game theory and EC is the one done by Duffy and Engle-Warnick (2002), which deals with the well-known two-player, repeated ultimatum game. In this work they used GP as a means of inferring the strategies that were played by subjects in economic decision-making experiments. Other works, within the field of EC and game theory, are the duopoly and oligopoly games (Chen & Ni, 2000). References regarding cooperation, coalition, and coordination are also made often and usually driven by EC techniques, Vriend (1995). In Jin and Tsang (2006), the authors applied GP to find strategies for sequential bargaining procedure and confirmed that equilibria can be approximated by GP. This gives opportunity to find approximate solutions to more complex situations for which theoretical solutions are yet to be found.

The interesting research by Riechmann (2002) proposes to study the foundations of the GAs by means of game theory. Riechmann interprets the GA as an N-players repeated game in which an individual of the GA represents a player with a different strategy. Once the author achieves the interpretation of the learning process of a GA as an evolutionary game, he attempts to shed some light on the fundamentals of GAs.

Auction Theory

Auction theory studies the behavior of the participants in auction markets. The study of auctions is relevant because they define the protocol which is followed by the participants in some important markets; for example, some stock markets, such as the New York Stock Exchange, operate under a double auction-like mechanism. There are many different types of auctions: the English auction, the Dutch auction, the Vickrey auctions, etc. In Klemperer (2004), there is a good introduction to the field.

EC techniques, particularly GAs, have been intensively used in auctions to derive bidding strategies in simulated auctions. In Andreoni and Miller (1995), the author uses adaptive learning, modelled with a GA, in order to capture patterns which arise in experimental auctions with humans. Such bidding patterns cannot be explained by the theoretical models, something that allowed the exploration of alternative methods such as adaptive behavior by means of EC. Some other relevant examples of the study of auctions using EC techniques are Anthony and Jennings (2003), Byde (2003), Cliff (2003), Dawid (1999), Mochon, Quintana, Sáez, and Isasi (2005), and Saez, Quintana, Isasi, and Mochon (2007).

Agent-Based Models

Agent-based computational economics (ACE) can be thought of as a branch of a wider area: Agent-based Modeling (ABM) (Wooldridge, 2002). The field of agent-based modeling is not restricted to economics, it has been applied in social sciences in general (Axelrod, 2003), in some classical and not so classical problems in computer science, and in some other disciplines. Axelrod provides an account of his experience using the agent-based methodology for several problems and he suggests that the ABM can be seen as a bridge between disciplines. Axelrod and Tesfatsion provide a good guide to the relevant literature of the ABM in Axelrod and Tesfatsion (2006). In Chen (2007), there is a good introduction to agents in economics and finance; in such work, Chen conceives the agents not just as economic agents but as computational intelligent units.

Most of the economic and finance theory is based on what is known as investor homogeneity or the representative agent. In ACE the researchers can depart from the assumptions of homogeneous expectations and perfect rationality by means of computational-based economic agents. In 2006, Tesfatsion surveys some of the most important works and topics on this area of research.

In ACE one of the main goals is to explain the macrodynamics of the economy by means of the microinteractions of the economic agents. This approach to the study of the economy has been called a "bottom-up" approach in opposition to the more traditional approaches in economics. An additional purpose of ACE is to handle real-world issues, which has become possible due to the technological advances in computational tools. With the use of programming languages, the agent-based approach allows us to represent explicitly agents with bounded rationality and heterogeneous preferences. Given a specific social structure, the simulation of the interaction among agents is the strength and the heart of the ABM. Even in its early stage of development, ABM is a promising area of research, which has opened the opportunity to social scientists to look for new insights in resolving relevant real-world issues. Considered "the third way of doing science" (Axelrod, 2003), modeling the behavior of the autonomous decision-making entities allows researchers to simulate the emergence of certain phenomena in order to gain better understanding of the object of study (Axtell, 2000). In this sense ACE, defined as "the computational study of economic processes modelled as dynamic systems of interacting agents" (Tesfatsion, 2006), is a growing area in the field of ABM. ACE research is developing rapidly, by using machine learning techniques, the researchers model the agents as software programs able to take autonomous decisions. Consequently, the interactions among the individuals at the microlevel give rise to regularities at the macrolevel (globally). The intention is to observe the emerging self-organizing process for a certain period of time, in order to study the presence of patterns or the lack of them.

Currently, the study of this self-organizing capability is one of the most active areas of ACE research.

One of the most crucial tasks in representing explicitly the market participants is the simulation of their autonomous decisions. Nowadays, advances in AI have opened possibilities of tackling this issue. In particular, techniques such as neural networks (NNs), genetic algorithms (GAs), genetic programming (GP), and other population-based algorithms are widely used in the field.

There are some interesting works in which the agent-based methodology is compared with experiments performed with human beings (Chan, LeBaron, Lo, & Poggio, 2001; Duffy, 2006). In both the works, the benefits that each type of research has on each other are identified. For instance, experimental research can be used as an important method to calibrate an agent-based model. On the other hand, agent-based simulations can be used to explain certain phenomena present in human experiments. To summarize, there are many beneficial ways in which both types of research influence each other.

According to Tesfatsion, the economic research being done with the ACE methodology can pursue one of two main objectives: the first one is the constructive explanation of macrophenomena and the second is the design of new economic mechanisms. In Tesfatsion (2006), Tesfatsion updates the classification of the research being made in ACE into four main categories: empirical understanding, normative understanding, methodological advancement, and finally, qualitative insight and theory generation.

EAs have been used for the modeling of the agents' learning in multiagent simulations. In multiagent simulations of economics systems, it is possible to find very different approaches and topics, just to illustrate some few examples of the immense amount of works, let us take a look at the following list:

- Electricity markets (Amin, 2002) (Learning Classifier System).
- Payment card markets (Alexandrova-Kabadjova, 2008) (Population Based Incremental Learning).
- Retail petrol markets (Heppenstall, Evans, & Birkin, 2006) (Genetic Algorithms).
- Stock markets (Arthur et al., 1997) (Learning Classifier Systems) and (Martinez-Jaramillo & Tsang, 2009b) (GP).
- Foreign exchange markets (Arifovic, 1996; Izumi & Ueda, 2001) (Genetic Algorithms).

Related to payment methods and systems, another economic phenomena characterized with complex social interaction suitable for ABM is the market dynamics of some electronic payment instruments, such as payment cards. In this field, the first evolutionary computation model was introduced in Alexandrova-Kabadjova (2008). This paper studies the competition among payment card scheme. The authors apply a Generalized Population Based Incremental Learning Algorithm (GPBIL), an extended version of the PBIL algorithm, in order to find an optimal price strategy for the electronic payment instrument.

Cross References

►Evolutionary Algorithms
►Evolutionary Computation in Finance
►Evolutionary Computational Techniques in Marketing
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Agapie, A., & Agapie, A. (2001). Evolutionary computation for econometric modeling. *Advanced Modeling and Optimization, 3*, 1–5.

Alexandrova-Kabadjova, B. (2008). *Evolutionary learning of the optimal pricing strategy in an artificial payment card market, Studies in computational intelligence* (Vol. 100). Berlin: Springer.

Amin, M. (2002). Restructuring the electric enterprise: Simulating the evolution of the electric power industry with intelligent adaptive agents. In A. Faruqui, & K. Eakin, (Eds.), *Market analysis and resource management* (Chap. 3). Boston: Kluwer Publishers.

Andreoni, J., & Miller, J. H. (1995). Auctions with artificial adaptive agents. *Games and Economic Behavior, 10*, 39–64.

Anthony, P., & Jennings, N. R. (2003). Developing a bidding agent for multiple heterogeneous auctions. *ACM Transactions on Internet Technology, 3*, 185–217.

Arifovic, J. (1994). Genetic algorithm learning and the cobweb model. *Journal of Economic Dynamics and Control, 18*, 3–28.

Arifovic, J. (1996). The behavior of the exchange rate in the genetic algorithm and experimental economics. *Journal of Political Economy, 104*, 510–541.

Arthur, W. B. (1991). Learning and adaptiver economic behavior. Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *American Economic Review, 81*, 353–359.

Arthur, W. B. (1994). Inductive reasoning and bounded rationality: The El Farol problem. *American Economic Review, 84*, 406–411.

Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R. G., & Talyer, P. (1997). Asset pricing under endogenous expectations in an artificial stock market. In W. Brian Arthur, S. Durlauf, & D. Lane, (Eds.), *The economy as an evolving complex system II*. Menlo Park: Addison-Wesley.

Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. In L. Davis (Ed.), *Genetic algorithms and simulated annealing, Research notes in AI* (Chap. 3, pp. 32–41). Los Altos, CA: Morgan Kaufmann.

Axelrod, R. (2003). Advancing the art of simulation in the social sciences. *Japanese Journal for Management Information System, Special Issue on Agent-Based Modeling, 12* (3).

Axelrod, R., & Hamilton, W. D. (1981). The evolution of cooperation. *Science, 211*, 1390–1396.

Axelrod, R., & Tesfatsion, L. (2006). A guide for newcomers to agent-based modeling in the social sciences. In K. L. Judd, & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics (Appendix A, pp. 1647–1656)*. Amsterdam: North-Holland.

Axtell, R. (2000). *Why agents? on the varied motivations for agent computing in the social sciences*. Working Paper 17, Center on Social and Economic Dynamics.

Brenner, T. (2006). Agent learning representation – advice in modelling economic learning. In K. L. Judd, & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics* (Chap. 18, pp. 895–948). Amsterdam: North-Holland.

Bullard, J., Arifovic, J., & Duffy, J. (1995). *Learning in a model of economic growth and development*. Working Paper 1995-017A, Federal Reserve Bank Of St. Louis.

Bullard, J., & Duffy, J. (1999). Using genetic algorithms to model the evolution of heterogeneous beliefs. *Computational Economics, 13*, 41–60.

Byde, A. (2003). Applying evolutionary game theory to auction mechanism design. In *ACM conference on electronic commerce* (pp. 192–193). New York: ACM.

Chan, N. T., LeBaron, B., Lo, A. W., & Poggio, T. (2001). *Agent-based models of financial markets: A comparison with experimental markets*. MIT Sloan Working Paper 4195-01, Massachusetts Institute of Technology.

Chen, S.-H. (2007). Editorial: Computationally intelligent agents in economics and finance. *Information Science, 177*(5), 1153–1168.

Chen, S.-H., & Ni, C. C. (2000). Simulating the ecology of oligopolistic competition with genetic algorithms. *Knowledge Information Systems, 2*(2), 285–309.

Chen, S.-H., & Yeh, C.-H. (1996). Genetic programming learning in the cobweb model with speculators. In *International computer symposium (ICS'96). Proceedings of international conference on artificial intelligence* (pp. 39–46), National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C.

Cliff, D. (2003). Explorations in evolutionary design of online auction market mechanisms. *Electronic Commerce Research and Applications, 2*, 162–175.

Dawid, H. (1999). On the convergence of genetic learning in a double auction market. *Journal of Economic Dynamics and Control, 23*, 1545–1567.

Duffy, J. (2006). Agent-based models and human subject experiments. In K. L. Judd, & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics* (Chap. 19, pp. 949–1012). Amsterdam: North-Holland.

Duffy, J., & Engle-Warnick, J. (2002). Using symbolic regression to infer strategies from experimental data. In S.-H. Chen (Ed.), *Evolutionary computation in economics and finance* (pp. 61–82). New York: Physica-Verlag.

Edmonds, B. (1999). Modelling bounded rationality in agent-based simulations using the evolution of mental models. In T. Brenner (Ed.), *Computational techniques for modelling learning in economics* (pp. 305–332). Dordrecht: Kluwer.

Greene, W. H. (2003). *Econometric analysis* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Heppenstall, A., Evans, A., & Birkin, M. (2006). Using hybrid agent-based systems to model spatially-influenced retail markets. *Journal of Artificial Societies and Social Simulation, 9*, 3.

Izumi, K., & Ueda, K. (2001). Phase transition in a foreign exchange market-analysis based on an artificial market approach. *IEEE Transactions of Evolutionary Computation, 5*(5), 456–470.

Jin, N., & Tsang, E. P. K. (2006). Co-adaptive strategies for sequential bargaining problems with discount factors and outside options. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 7913–7920). Washington, DC: IEEE Press.

Klemperer, P. (2004). *Auctions: Theory and practice. The Toulouse lectures in economics*. Princeton, NJ: Princeton University Press.

Koza, J. (1992). A genetic approach to econometric modelling. In P. Bourgine, & B. Walliser, (Eds.), *Economics and cognitive science* (pp. 57–75). Oxford: Pergamon Press.

Lucas, R. E. (1986). Adaptive behavior and economic theory. In R. M. Hogarth, & M. W. Reder, (Eds.), *Rational choice: The contrast between economics and psychology* (pp. 217–242). Chicago: University of Chicago Press.

Marimon, R., McGrattan, E., & Sargent, T. J. (1990). Money as a medium of exchange in an economy with artificially intelligent agents. *Journal of Economic Dynamics and Control, 14*, 329–373.

Martinez-Jaramillo, S., & Tsang, E. P. K. (2009). An heterogeneous, endogenous and coevolutionary gp-based financial market. *IEEE Transactions on Evolutionary Computation, 13*, 33–55.

Mochon, A., Quintana, D., Sáez, Y., & Isasi, P. (2005). Analysis of ausubel auctions by means of evolutionary computation. In *IEEE congress on evolutionary computation (CEC 2005)* (pp. 2645–2652). Edinburgh, Scotland.

Nash, J. (1950). The barganing problem. *Econometrica, 18*, 155–162.

Östermark, R. (1999). Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm. *Computational Economics, 13*(2), 103–115.

Riechmann, T. (2002). *Genetic algorithm learning and economic evolution. Studies in fuzziness and soft computing* (pp. 45–59). Heidelberg: Physica-Verlag.

Saez, Y., Quintana, D., Isasi, P., & Mochon, A. (2007). Effects of a rationing rule on the ausubel auction: A genetic algorithm implementation. *Computational Intelligence, 23*(2), 221–235.

Simon, H. A. (1957). *Models of man: Social and rational.* New York: John Wiley.

Maynard Smith, J. (1972). *Game theory and the evolution of fighting* (pp. 8–28). Edinburgh: Edinburgh University Press.

Tesfatsion, L. (2006). Agent-based computational economics: A constructive approach to economic theory. In K. L. Judd & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics* (Chap. 16, pp. 831–880). Amsterdam: North-Holland.

Tsang, E. P. K. (2008). Computational intelligence determines effective rationality. *International Journal of Automation and Computing, 5*, 63–66.

von Neumann, J., & Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton, NJ: Princeton University Press.

Vriend, N. J. (1995). Self-organization of markets: An example of a computational approach. *Computational Economics, 8*, 205–231.

Witt, U. (2008). *Evolutionary economics* (2nd ed.). Basingstoke, UK: Palgrave Macmillan.

Wooldridge, M. (2002). *An Introduction to multiagent systems.* Chichester: Wiley.

Evolutionary Computation in Finance

Serafín Martínez-Jaramillo, Alma Lilia García-Almanza,
Biliana Alexandrova-Kabadjova,
Tonatiuh Peña Centeno
Bank of Mexico,
Mexico, D.F

Definition

Evolutionary computation (EC) in finance is an area of research and knowledge which involves the use of techniques, known as evolutionary algorithms (EAs), to approach topics in finance. This area of knowledge is similar to EC in economics, in fact such areas frequently overlap regarding some of the topics approached. The application of EC in finance pursues two main goals: first, to overcome the limitations of some theoretical models (and the strong assumptions being made by such models) and second, to innovate in this extremely competitive area of research.

Motivation and Background

Evolutionary computation is a field in Machine Learning in which the developed techniques apply the principle of Evolution in several different ways. The application of EC in finance includes portfolio optimization, financial forecasting, asset pricing, just to mention some examples.

In finance, competition is at the center of the everyday activities by the individuals and companies that participate in this field. For example, in the stock markets everybody is trying to beat the market in order to make more profits than the other participants.

As a result of this fierce competition, there is a constant need to innovate and machine learning has provided novel and competitive tools in financial research. Therefore, it is natural to find numerous problems in finance being approached by any of the existent EC techniques like ►Genetic Programming (GP), ►Genetic Algorithms (GAs), Evolutionary Strategies (EAs), etc. This field has been called in many different ways like computational finance, computational intelligence in finance, etc. Research in this area is still evolving; therefore, it is difficult to define it clearly or to establish its limits. Moreover, nowadays it is almost impossible to provide a full account of all the relevant work that involves any form of EC in finance. It is also hard to organize this vast amount of human knowledge.

Nowadays, computing in finance is an almost unavoidable tool, from Monte Carlo simulation and optimization to computer intensive methods to valuate complex derivatives; in fact, some of the most critical processes in finance make heavy use of computers. Moreover, this research and professional practices have been known as computational finance and the application of evolutionary techniques in finance fit within such definition. Computational finance is now a frequently mentioned term and is frequently associated with financial engineering. However, in this context we refer to computational finance as the use of nonconventional computational techniques, like EC or other machine learning techniques, to tackle problems in finance. See for example, Tsang and Martinez-Jaramillo (2004) for a good introduction to the field.

Financial Forecasting

Financial forecasting is one of the most important fields in the computational finance area (Tsang & Martinez-Jaramillo, 2004) and EC has been used to solve a great variety of financial forecasting problems, such as, detection of stock price movements, volatility prediction, forecasting of foreign exchange markets, and so on.

Machine learning classifiers, like other forecasting techniques, extend past experience into the future. The aim is to analyze past data in order to identify patterns in the interest of creating a model to predict future events. In this section we will introduce some important works in the financial forecasting area, which take advantage of some of the EC distinctive features. First, the relevance of the interpreatability of the solution is illustrated; after that, some examples about the usefulness of genetating multiple solutions for the same problem are given. Then, some works that use EC as an optimization approach to solve forecasting problems are presented. Finally, the use of a great variety of representations is highlighted. Evolutionary techniques are able to produce interpretable solutions, this property is especially important for predictions, since the main goals of classification are: to generate an accurate classification model that is be able to predict unseen cases and to discover the predictive structure of the problem (Breiman, Friedman, Olshen, & Stone, 1984). Models for understanding provide information about the structural patterns in data that can be useful to recognize the variables' interactions. There are classification models that have good predictive power, however, these provide a poor representation of the solution; for example, ►Artificial Neural Networks (ANNs). Since EC techniques provide not just a good prediction but an interpretable solution, these have been used in financial problems to acquire knowledge of the event to predict. For example, Tsang, Yung, and Li (2004) trained a GP using past data from the financial stock markets to predict price movements of at least $r\%$ in a period of at most n times. The attributes used to train the GP were indicators from technical analysis. Due to the interpretability of the solution, the authors were able to analyze the most successful indicators in the result. In fact, some researchers have used EC in order to discover new financial indicators such as Allen and Karjalainen (1999), who made use of a GP system to infer technical trading rules from past. In the same vein, Bhattacharyya, Pictet, and Zumbach (2002) used GP to discover trading decision models from high-frequency foreign exchange (FX) markets data. In other research, Bhattacharyya et al. (2002) used GA for mining financial ►time-series to identify patterns, with the aim to discover trading decision models. In a different approach, Potvin, Soriano, and Vallée (2004) applied GP to automatically generate short-term trading rules on the stock markets, the authors used historical pricing and transaction volume data reported for 14 Canadian companies from the Toronto stock exchange market. Other approach called grammatical evolution (GE) (Brabazon & O'Neill, 2004) was applied to discover new technical trading rules, which can be used to trade foreign exchange markets. In that approach, each of the evolved programs represents a market trading system.

As it was mentioned earlier, EC techniques are able to generate a set of solutions for a single problem, this quality has been used to collect a set of results, with the aim of applying the most suitable solution according to the situation, for instance Lipinski (2004) analyzed high-frequency data, from the Paris Stock Exchange Market. In that model, stock market trading rules were combined into stock market trading experts, which defined the trading expertise. The author used a simple GA, a population-based incremental learning, the compact genetic algorithm, and the extended compact genetic algorithm to discover optimal trading experts in a specific situation, the author argues that the optimal solution depends on the specific situation on the stock market, which varies with time. EC plays an important role in the learning and continual adaptation to the changing environment.

Taking advantage of the EC's ability to generate multiple solutions, Garcia-Almanza and Tsang (2008) proposed an approach, called evolving comprehensible rules (ECR), to discover patterns in financial data sets to detect investment opportunities. ECR was designed to classify the minority class in imbalanced environments, which is particularly useful in financial forecasting because the number of profitable chances is scarce. The approach offers a range of solutions to suit the investor's risk guidelines and so, the user can choose the best trade-off between miss-classification and false alarm costs according to the investor's requirements. Another approach proposed by Ghandar et al. (2008) was designed to generate trading rules, the authors implemented an adaptive computational intelligent system by using an evolutionary algorithm and a fuzzy logic rule base representation. The data to train the system was composed just by volume and price. The authors' objective was to create a system to generate rules for buy recommendations in dynamic market conditions. An analysis of the results was provided by applying the system for portfolio construction in historical data for companies listed as part of the MSCI Europe Index from 1990 to 2005. The results showed that their approach was able to generate trading rules that beat traditional, fixed rule-based strategies, as the price momentum and alpha portfolios, but this also beat the market index.

Given that EC can be used as an optimization technique, it has been combined with other approaches. As an instance, Chen, Wang, and Zhang (1999) used a genetic algorithm to determine the number of input variables and the number of hidden layers in an ANN for forecasting foreign exchange rates of the Dollar/Deutsche mark. Chen and Lu (1999) used GP to optimize an ANN, this approach is called evolutionary neural trees (ENT). The objective was to forecast the high-frequency stock returns of the Heng–Sheng stock index. Schoreels, Logan, and Garibaldi (2004) investigated the effectiveness of an agent based trading system. The system employs a simple GA to optimize the trading decisions for every agent, the knowledge was based on a range of technical indicators generating trading signals. In Dempster, Payne, Romahi, and Thompson (2001) the authors aim to detect buy and sell signals in the exchange (FX) markets. The authors analyzed and compare the performance of a GP combined with a reinforcement learning system to a simple linear program characterizing a ►Markov decision process and a heuristic in high-frequency (intraday) foreign exchange trading. The authors considered eight popular technical indicators used by intraday FX traders, Based on simple trend-indicators such as moving averages as well as more complex rules. From experimental results the authors found that all methods were able to create significant in-sample and out-of-sample profits when transaction costs are zero. The GP approach generated profits for nonzero transaction costs, although none of the methods produce significant profits at realistic transaction costs.

As it can be seen from the previous paragraphs, EC techniques allow representing the solutions using different structures, such as, decision trees (Potvin et al. (2004)), finite states automats, graphs, grammar (Brabazon & O'Neill, 2004), networks, binary vectors (Lipinski, 2004) among may others. In fact, this characteristic lets us to choose the best representation for the problem.

Portfolio Optimization

Portfolio optimization is probably the most important task in finance. The most relevant aspects in finance are involved in such task: the determination of the price, the estimation of the volatility, the correlation among stocks, etc. The portfolio selection problem can be described in a simple way as the problem of choosing the assets and the proportion of such assets in an investor's portfolio that wants to maximize his profits and minimize the risk.

As its name suggest, Portfolio Optimization is an optimization problem and EC has proven to be very useful in difficult (sometimes intractable) optimization problems. In (Maringer, 2005), the author explains extensively the portfolio optimization problem and the possible heuristic approaches, including ant systems (AS), memetic algorithms (MAs), GAs, and ESs.

Multi-objective evolutionary optimization is an important field within EC and the portfolio optimization problem is one its more important applications in finance. Being a multi-objective optimization problem, EC provides plenty of opportunities and different approaches can be used for the portfolio optimization problem. For example, Hassan and Clack (2008) use a multi-objective GP to approach this problem. In Diosan (2005), the author compares different multi-objective evolutionary algorithms for the portfolio optimization problem.

The number of papers on portfolio optimization using any form of EC techniques is huge and still growing. For example, in (Loraschi et al., 1995) the authors use distributed genetic algorithms to approach the portfolio optimization problem, whereas in (Loraschi and Tettamanzi, 1995) and (Streichert, Ulmer, and Zell, 2004) the authors use EAs.

Financial Markets

Financial markets are mechanisms where buyers and sellers exchange goods like bonds, gold, options, currencies, etc. Some examples of such markets are the New York Stock Exchange, the Chicago Mercantil Exchange, and the NASDAQ Stock Market.

Financial markets are essential for financial systems and for the overall economy. Such markets represent one of the most efficient ways to allocate financial resources into companies, due to the low transaction costs and the public information available for buyers and sellers. However, bubbles and crashes are recurrent phenomena which have enormous repercussions to global economy. In fact, nowadays we can see as never before that one single crash in one market could lead to a worldwide slump on most of the remaining stock markets. Crises in financial markets could affect other aspects of the (real) economy; for example, interest rates, inflation, unemployment, etc. This, in turn could cause even more instability on the financial markets as we have witnessed recently. Moreover, market crashes occur with an unpleasant higher frequency than is predicted by the standard economic theory.

One of the most important research issues in financial markets is the explanation of the process that determines the asset prices and as a result the rate of return. There are many models that can be used to explain such process, like the capital asset pricing model (CAPM), the arbitrage pricing theory (APT) or the black-scholes option pricing. Unfortunately, such models do not explain, as one would expect, the behavior of prices in real markets. The contradictions between the existing theory and the empirical properties of the stock market returns are one of the motivations for some researchers to develop and use different approaches to study financial markets. An additional aspect on the study of financial markets is the complexity of the analytical models of such markets. Financial markets are also very complex to analyze the wide variety of participants and their ever-changing nature. Previous to the development of some new simulation techniques, very important simplifying (unrealistic) assumptions had to be made in order to allow tractability of the theoretical models.

Behavioral finance, agent-based computational economics (ACE) (Tesfatsion, 2002) and computational finance (Tsang & Martinez-Jaramillo, 2004) have risen as alternative ways to overcome some of the problems of the analytical models. AI and in particular EC have been used in the past to study some financial and economic problems. However, the development of a well established community, known as the ACE community, facilitates the study of phenomena in financial markets that was not possible in the past. Within such community, a vast number of works and a different number of approaches are being produced by numbers in order to solve or gain more understanding of some economic and financial problems.

The influential work of Arthur, Holland, LeBaron, Palmer, & Talyer, (1997) and previously the development of the concept of bounded rationality (Arthur, 1991; Simon, 1982) changed the way in which we conceive and model the economic agents. This change in conception, modified dramatically the possibilities to study some economic phenomena and in particular the Financial Markets. The new models of economic agents have changed, there is no need any more of fully rational representative agents, there is no need of homogeneous expectations and information symmetry. Furthermore, the development of artificially adapted agents (Holland & Miller, 1991) gives to the economics science a way forward into the study of economic systems.

Agent-based financial markets of different characteristics have been developed for the study of such markets in the last decade since the influential Santa Fe Artificial Market. (The Santa Fe Artificial Stock Market is a simulated stock market developed at the Santa Fe Institute. Such market was developed by team of highly reputed researchers, among them John Holland, the inventor of genetic algorithms (Holland, 1975).) (Arthur et al., 1997). Some of them differ from the original Santa Fe market on the type of agents used like Chen and Yeh (2001), Gode and Sunder (1992), Martinez-Jaramillo and Tsang (2009b); on market mechanism like Bak, Paczuski, and Shubik (1997), Gode and Sunder (1992). Other markets borrow ideas from statistical mechanics like Levy, M., Levy, H., and Solomon (1994) and Lux (1998). Some important research has been done modelling stock markets inspired on the minority game. (The Minority Game was first proposed by Yi-Cheng Zhang and Damien Challet (1997) inspired by El Farol bar problem introduced by Brian Arthur (1994).) like

Challet, Marsili, and Zhang (2000). There are financial simulated markets in which several stocks are traded like in Cincotti, Ponta, and Raberto (2005). However, there are some criticisms to this approach like the problem of calibration, the numerous parameters needed for the simulation program, the complexity of the simulation, etc.

Although they all differ in the sort of assumptions made, methodology and tools; these markets share the same essence: the macro behavior of such market (usually the price) should emerge endogenously as a result of the micro-interactions of the (heterogeneous) market participants. This approach is in opposition with the traditional techniques being used in Economics and Finance.

One of the most crucial aspects on the modelling of financial markets is the modelling of the market participants also known as "agents". Unfortunatelly, for the sake of mathematical tractability, theoretical models assume that all the market participants can be modelled by a representative agent. The *representative agent* is a common, yet very strong, assumption in the modeling of financial markets. This concept has been the source of controversy and strong criticisms. For example, in Kirman (1992), the author criticizes the *representative individual* approach in economics. Moreover, Lux and Ausloos (2002) declare:

▸ Unfortunately, standard modelling practices in economics have rather tried to avoid heterogeneity and interaction of agents as far as possible. Instead, one often restricted attention to the thorough theoretical analysis of the decisions of one (or few) *representative* agents

In order to overcome the limitations of such an assumption, some researchers has opted for less orthodox techniques like GAs and GPs to model the participants in financial markets. Such evolutionary techniques have been widely used to model the agents' behaviour and adaptation in financial markets. In order to understand the different approaches of the variety of artificial (simulated) financial markets, it is useful to describe the different types of markets on the basis of the framework proposed in LeBaron (2001). In such work, LeBaron identifies the key design issues present in every artificial financial market and describes some of the most important works until then. The main design issues identified in LeBaron (2001) are:

- Agents
- Market mechanism
- Assets
- Learning
- Calibration
- Time

In addition to the description of the different approaches in artificial financial markets by using the above described framework, there is a fairly detailed extension of it in Grothmann (2002) that is worth looking at. In such work the basic design issues proposed in LeBaron (2001) are extended and detailed. For a more complete and detailed guide to the application of EC techniques in artificial financial markets, see Martinez-Jaramillo and Tsang (2009).

Option Pricing

Derivatives (See Hull, 2008 for an introduction.) are financial instruments whose main purpose is to hedge risk; however, they can also be used with speculation purposes with potentially negative effects on the financial health of the companies. Derivatives markets are having an important expansion in recent years; futures, forwards, swaps, and options are the best known types of derivatives. Having said so, option pricing is an extremely important task in finance. The Black–Scholes model for option pricing is the reference analytical model as it has an important theoretical framework behind it. However, in practice prices show that there is a departure from the prices obtained with such model. One possible reason that could explain such departure is the assumptions being made in such model (the assumption of constant volatility and the assumption that prices follow a geometric Brownian motion). This is why GP has been used as an alternative to perform option pricing in Chen, Yeh, and Lee (1998), Fan, Brabazon, O'Sullivan, and O'Neill (2007), Yin, Brabazon, and O'Sullivan (2007). Interestingly, not only GP has been used to perform option pricing but also ACO has been explored to approach this important problem in finance (Kumar, Thulasiram, & Thulasiraman, 2008).

Credit Scoring, Credit Rating, and Bankruptcy Prediction

Credit rating and credit scoring are two examples of financial problems that have been traditionally approached through statistical analyzes. Credit rating is an estimate of a corporation's worthiness to be given a credit and is generally expressed in terms of an ordinal value; whereas credit scoring is a technique used express the potential risk of lending money to a given consumer in terms of a probability measure. Both techniques are therefore similar in their ends but applied to different domains.

The seminal work in the field of credit scoring is that of Altman (1968), who proposed the application of linear discriminant analysis (Fisher, 1936) to a set of measurements known as financial ratios, i.e., indicators of a corporation's financial health, that are obtained from the corporation's financial statements. One of the main applications of Altmans' method, also known as *Z*-score, is bankruptcy prediction. Understandably, a series of improvements have been achieved by means of applying more powerful classifiers, such as decision trees, genetic programming, neural networks and support vector machines, among others. References that apply such techniques or that make a literature review of their application are Atiya (2001), Huang, Chen, and Wang (2007), Ong, Huang, and Tzeng (2005), Shin and Lee (2002), and Martens, Baesens, Gestel, and Vanthienen (2007).

Another method to evaluate credit worthiness is the one provided by specialized agencies. The so-called credit ratings are nothing more than ordinal values expressing the financial history, current assets, and liabilities of entities such as individuals, organizations, or even sovereign countries, such that they represent their risk of defaulting a loan. Although each rating agency uses its own methodology and scale these are usually not disclosed, nevertheless, on the academic realm, several superseding techniques to ordinal regression have been applied. For example, Huang, Chen, H., Hsu, Chen, W. H., and Wu (2004) and Paleologo, Elisseeff, and Antonini (2010) have proposed computationally oriented methods to solve this problem.

Related to bankruptcy prediction, NNs have been the standard selection apart from the traditional statistical methods (discriminant analysis, logit and probit models). Quintana, Saez, Mochon, and Isasi (2008) explore the feasibility of using the evolutionary nearest neighbor classifier algorithm (ENPC) suggested by (Fernández & Isasi, 2004) in the domain of early bankruptcy prediction. They assess its performance comparing it to six other alternatives, their results suggest that this algorithm might be considered as a good choice. Another relevant work is Turku, Back, Laitinen, Sere, and Wezel (1996) in which the authors compare discriminant analysis, logit analysis, and GAs for the selection of the independent variables used for the prediction model. Finally, in Lensberg, Eilifsen, and McKee (2006), the authors use GP to study bankruptcy in Norwegian companies and find acceptable accuracy in addition to information about the usefulness of the variables used for the prediction task.

Cross References

►Evolutionary Algorithms
►Evolutionary Computation in Economics
►Evolutionary Computational Techniques in Marketing
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics, 51*, 245–271.

Altman, E. I. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *Journal of Finance, 23*(4), 589–609.

Arthur, W. B. (1991). Learning and adaptive economic behavior. Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *American Economic Review, 81*, 353–359.

Arthur, W. B. (1994). Inductive reasoning and bounded rationality: The El Farol problem. *American Economic Review, 84*, 406–411.

Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R. G., & Talyer, P. (1997). Asset pricing under endogenous expectations in an artificial stock market. In W. B. Arthur, S. Durlauf, & D. Lane (Eds.), *The economy as an evolving complex system II*. Reading, MA: Addison-Wesley.

Atiya, A. F. (2001). Bankruptcy prediction for credit risk using neural networks: A survey and new results. *IEEE Transactions on Neural Networks, 12*(4), 929–935.

Bak, P., Paczuski, M., & Shubik, M. (1997). Price variations in a stock market with many agents. *Physica A, 246*, 430–453.

Bhattacharyya, S., Pictet, O. V., & Zumbach, G. (2002). Knowledge-intensive genetic discovery in foreign exchange markets. *IEEE Transactions on evolutionary computation, 6*(2), 169–181.

Brabazon, A., & O'Neill, M. (2004). Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Computational Management Science, 1*(3), 311–327.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth, CA: Wadsworth International Group.

Challet, D., Marsili, M., & Zhang, Y. C. (2000). Modeling market mechanism with minority game. *Physica A, 276*, 284–315.

Challet, D., & Zhang, Y. C. (1997). Emergence of cooperation and organization in an evolutionary game. *Physica A, 246*, 407.

Chen, S. H., & Lu, C. F. (1999). Would evolutionary computation help in designs of artificial neural nets in forecasting financial time series? In *IEEE Proceedings of 1999 congress on evolutionary computation,* (pp. 275–280). New Jersey: IEEE Press.

Chen, S. H., Wang, H. S., & Zhang, B. T. (1999). Forecasting high-frequency financial time series with evolutionary neural trees: The case of hang-seng stock index. In H. R. Arabnia (Ed.), *Proceedings of the international conference on artificial intelligence, IC-AI ́99* (Vol. 2, pp. 437–443). Las Vegas, NV: CSREA Press.

Chen, S. H., & Yeh, C. H. (2001). Evolving traders and the business school with genetic programming: A new architecture of the agent-based artificial stock market. *Journal of Economic Dynamics and Control, 25*(3–4), 363–393.

Chen, S. H., Yeh, C. H., & Lee, W. C. (1998). Option pricing with genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, et al. (Eds.), *Proceedings of the third annual conference, genetic programming 1998:* (pp. 32–37). San Francisco: Morgan Kaufmann.

Cincotti, S., Ponta, L., & Raberto, M. (2005). A multi-assets artifcial stock market with zero-intelligence traders. In *WEHIA 2005*, Essex, UK.

Dempster, M. A. H., Payne, T. W., Romahi, Y., & Thompson, G. W. P. (2001). Computational learning techniques for intraday fx trading using popular technical indicators. *IEEE Transactions on Neural Networks, 12*, 744–754.

Diosan, L. (2005). A multi-objective evolutionary approach to the portfolio optimization problem. In *CIMCA '05: Proceedings of the international conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce Vol. 2 (CIMCA-IAWTIC'06)* (pp. 183–187). Washington DC: IEEE Computer Society.

Fan, K., Brabazon, A., O'Sullivan, C., & O'Neill, M. (2007). Option pricing model calibration using a real-valued quantum-inspired evolutionary algorithm. In *GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 1983–1990). New York: ACM.

Fernández, F., & Isasi, P. (2004). Evolutionary design of nearest prototype classifiers. *Journal of Heuristics, 10*(4), 431–454.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*, 179.

Garcia-Almanza, A. L., & Tsang, E. P. K. (2008). Evolving decision rules to predict investment opportunities. *International Journal of Automation and Computing, 5*(1), 22–31.

Ghandar, A., Michalewicz, Z., Schmidt, M., To, T. D., & Zurbrugg, R. (2008). Computational intelligence for evolving trading rules. *IEEE Transactions on Evolutionary Computation, 13*(1), 71–86.

Gode, D. K., & Sunder, S. (1992). *Allocative efficiency of markets with zero intelligence (z1) traders: Market as a partial substitute forindividual rationality*. GSIA working papers 1992-16, Carnegie Mellon University, Tepper School of Business.

Grothmann, R. (2002). *Multi-agent market modeling based on neural networks*. PhD thesis, University of Bremen, Germany.

Hassan, G., & Clack, C. D. (2008). Multiobjective robustness for portfolio optimization in volatile environments. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 1507–1514). New York: ACM.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Holland, J. H., & Miller, J. H. (1991). Artificial adaptive agents in economic theory. *The American Economic Review, 81*, 365–370.

Huang, C. L., Chen, M. C., & Wang, C. J. (2007). Credit scoring with a data mining approach based on support vector machines. *Expert Systems with Applications, 33*(4), 847–856.

Huang, Z., Chen, H., Hsu, C. J., Chen, W. H., & Wu, S. (2004). Credit rating analysis with support vector machines and neural networks: A market comparative study. *Decision Support Systems, 37*(4), 543–558.

Hull, J. (2008). *Options, futures and other derivatives. Prentice Hall Series in Finance*. New Jersey: Prentice Hall. 7th Edition.

Kirman, A. P. (1992). Whom or what does the representative individual represents? *The Journal of Economic Perspectives, 6*, 117–136.

Kumar, S., Thulasiram, R. K., & Thulasiraman, P. (2008). A bioinspired algorithm to price options. In *C3S2E '08: Proceedings of the 2008 C3S2E conference* (pp. 11–22). New York: ACM.

LeBaron, B. (2001). A builder's guide to agent based financial markets. *Quantitative Finance, 1*, 254–261.

Lensberg, T., Eilifsen, A., & McKee, T. E. (2006). Bankruptcy theory development and classification via genetic programming. *European Journal of Operational Research, 169*(2), 677–697; Feature cluster on scatter search methods for optimization.

Levy, M., Levy, H., & Solomon, S. (1994). A microscopic model of the stock market: cycles, booms and crashes. *Economics Letters, 45*, 103–111.

Lipinski, P. (2004). *Evolutionary data-mining methods in discovering stock market expertise from financial time series*. PhD thesis, University of Wroclaw, Wroclaw, Poland.

Loraschi, A., & Tettamanzi, A. (1995). An evolutionary algorithm for portfolio selection in a downside risk framework. *The European Journal of Finance*, 1994.

Loraschi, A., Tettamanzi, A., Tomassini, M., Svizzero, C., Scientifico, C., & Verda, P. (1995). Distributed genetic algorithms with an application to portfolio selection. In *Artificial neural nets and genetic* (pp. 384–387). Berlin: Springer.

Lux, T. (1998). The socio-economic dynamics of speculative markets: Interacting agents, chaos, and the fat tails of return distributions. *Journal of Economic Behavior and Organization, 33*, 143–165.

Lux, T., & Ausloos, M. (2002). Market fluctuations I: Scaling, multiscaling and their possible origins. In A. Bunde, J. Kropp, & H. J. Schellnhuber (Eds.), *Theories of disaster – scaling laws governing weather, body, and stock market dynamics* (pp. 373–409). Berlin: Springer.

Maringer, D. (2005). *Portfolio management with heuristic optimization. Advances in computational management science* (Vol. 8). Berlin: Springer.

Martens, D., Baesens, B., Gestel, T. V., & Vanthienen, J. (2007). Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research, 183*(3), 1466–1476.

Martinez-Jaramillo, S., & Tsang, E. P. K. (2009a). *Evolutionary computation and artificial financial markets. Studies in computational intelligence* (Vol. 185, pp. 137–179). Berlin: Springer.

Martinez-Jaramillo, S., & Tsang, E. P. K. (2009b). An heterogeneous, endogenous and coevolutionary gp-based financial market. *IEEE Transactions on Evolutionary Computation, 13*, 33–55.

Ong, C. S., Huang, J. J., & Tzeng, G. H. (2005). Building credit scoring models using genetic programming. *Expert Systems with Applications, 29*(1), 41–47.

Paleologo, G., Elisseeff, A., & Antonini, G. (2010). Subagging for credit scoring models. *European Journal of Operational Research. 201*(2), 490–499.

Potvin, J. Y., Soriano, P., & Vallée, M. (2004). Generating trading rules on the stock markets with genetic programming. *Computers and Operations Research, 31*(7), 1033–1047.

Quintana, D., Saez, Y., Mochon, A., & Isasi, P. (2008). Early bankruptcy prediction using enpc. *Applied Intelligence, 29*(2), 157–161.

Schoreels, C., Logan, B., & Garibaldi, J. M. (2004). Agent based genetic algorithm employing financial technical analysis for making trading decisions using historical equity market data. In *IAT '04: Proceedings of the intelligent agent technology, IEEE/WIC/ACM international conference* (pp. 421–424). Washington, DC: IEEE Computer Society.

Shin, K. S., & Lee, Y. J. (2002). A genetic algorithm application in bankruptcy prediction modeling. *Expert Systems with Applications, 23*(3), 321–328.

Simon, H. A. (1982). *Models of bounded rationality* (Vol. 2). Cambridge, MA: MIT Press.

Streichert, F., Ulmer, H., & Zell, A. (2004). Evaluating a hybrid encoding and three crossover operators on the constrained portfolio selection problem. In *Proceedings of the 2004 congress on evolutionary computation* (pp. 932–939). New Jersey: IEEE Press.

Tesfatsion, L. (2002). Agent-based computational economics: Growing economies from the bottom up. *Artificial Life, 8*, 55–82.

Tsang, E. P. K., & Martinez-Jaramillo, S. (2004). Computational finance. In *IEEE computational intelligence society newsletter* (pp. 3–8). New Jersey: IEEE Press.

Tsang, E. P. K., Yung, P., & Li, J. (2004). Eddie-automation, a decision support tool for financial forecasting. *Journal of Decision Support Systems, Special Issue on Data Mining for Financial Decision Making, 37*(4), 559–565.

Turku, B. B., Back, B., Laitinen, T., Sere, K., & Wezel, M. V. (1996). Choosing bankruptcy predictors using discriminant analysis, logit analysis, and genetic algorithms. In *Proceedings of the first international meeting on artificial intelligence in accounting, finance and tax* (pp. 337–356). Huelva: Spain.

Yin, Z., Brabazon, A., & O'Sullivan, C. (2007). Adaptive genetic programming for option pricing. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on genetic and evolutionary computation* (pp. 2588–2594). New York: ACM.

Evolutionary Computational Techniques in Marketing

Alma Lilia García-Almanza, Biliana Alexandrova-Kabadjova,
Serafín Martínez-Jaramillo
Bank of Mexico, Mexico, D.F.

Definition

Evolutionary Computation (EC) in marketing is a field that uses evolutionary techniques to extract and gather useful patterns with the objective of designing marketing strategies and discovering products and services of superior value which satisfy the customers' necessities. Due to the fierce competition by some companies for attracting more customers and the necessity of innovation, it is common to find numerous marketing problems being approached by EC techniques.

Motivation and Background

The objective of marketing is to identify the customers' needs and desires in order to guide the entire organization to serve best by designing products, services, and programs which satisfy customers (Kotler & Armstrong, 1996). Nowadays, the market competition is very strong, since customers can choose from several alternatives. For that reason, marketing teams are facing the necessity of creating intelligent business strategies. Thus, new artificial intelligent approaches for marketing have emerged; especially, evolutionary algorithms have been used to solve a variety of marketing problems such as the design of more attractive products and services for consumers, the analysis of populations to target potential clients, the design of new marketing strategies, and more.

Applications

Nowadays, it is very easy to capture and store large sets of data. However, such data must be processed and analyzed in order to obtain useful information to make marketing decisions. Since EC techniques can be used to extract patterns from data, these have been used in marketing for multiple purposes. In order to illustrate the application of EC in marketing, let us introduce some important works in this field.

Target potential clients

Bhattacharyya (2000) proposed a Genetic Algorithm (GA) in combination with a case-based reasoning system to predict customer purchasing behavior. The objective was to identify potential customers for a specific product or service. This approach was developed and tested with real cases from a worldwide insurance direct marketing company. An optimization mechanism was integrated into the classification system to select those customers who were most likely to acquire an insurance.

New Products design

As it was mentioned previously, one of the goals of marketing is to discover products of superior value and quality. To achieve this goal, Fruchter et al. (2006) resolved to design a product line rather than a single product. The authors argued that by offering a product line, the manufacturer can customize the products according to the necessities of different segments of the population, which would satisfy more customers. Since the amount of data about customer preferences was large, the optimization of the product line became very difficult. The authors used a GA to optimize the problem and the performance of the solutions was valued by measuring the manufacturer's profits. In the same vein, Liu and Ong (2008) used a GA to solve a problem of marketing segmentation, this approach was used to make strategy decisions for reaching effectively all customers. In other approach proposed by Balakrishnan and Jacob (1996), a GA was used to optimize the customer's preferences in new products' design. The authors explained that, to design a new product it is important to determine its attributes, such as color or shape. A study to gather the customers' preferences had to be carried out. Finally, a GA was used to select those attributes that satisfied a bigger number of customers.

Advertisement

Advertising is an important area of marketing; this is defined as the activity of attracting public attention to a product or business. Since personalized advertisement improves marketing efficiency, Kwon and Moon (2001) proposed a personalized prediction model to be used in email marketing. A circuit model combined with Genetic Programming (GP) was proposed to analyze customers' information. The result was a set of recommendation rules, which was tested over a general mass marketing. According to the authors, the model achieved a significant improvement in sales. In Naik, Mantrala, and Sawyer (1998), the authors used a GA combined with a Kalman filter procedure to determine the best media schedule for advertisement, which was constrained by a budget. This approach evaluated a large number of alternative media schedules to decide the best media planning solution.

Internet has become a very popular and convenient media to make businesses. Many products and services can be found easily in a very short time, increasing the competition between those providers. Since this kind of sales does not involve human interaction directly, it is essential to design new and better strategies to personalize the Web pages in order to contend in this media. As an instance, Abraham and Ramos (2003) proposed an ant clustering algorithm to discover Web usage patterns and a linear genetic program to analyze the visitor trends. The objective was to discover useful knowledge from interactions of the users with the Web. The knowledge was used to design adaptive Web sites, business and support services, personalization, network traffic flow analysis, and more.

According to Scanlon (2008), the company Staples used a software called IDDEA to redesign and relaunch its paper brand. This software was developed by Affinova Inc, and uses a GA to simulate the evolution of consumer markets where strong products survive and weak ones die out. The strongest possible design emerges after several generations. A panel of 750 consumers select their favorite options from each generation. The software analyze customers choices over multiple generations to identify preference patterns. Surveys include consumer profiles that comprised basic demographic information, customer beliefs and consumer habits. This allow them to understand how different designs attract different consumers. In another project, IDDEA was used to identify imagery and messaging that would be of interest to consumers. As can be seen from previous paragraphs, EC has been used to solve a variety of marketing problems.

Since EC is able to dcal with optimization, forecasting and data mining problems, among others, there is a great potential of usage in the field of marketing to optimize processes, to extract patterns of customers from large amount of data, to forecast purchasing tendencies, and many others.

Cross References

►Evolutionary Algorithms
►Evolutionary Computation in Economics
►Evolutionary Computation in Finance
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Abraham, A., & Ramos, V. (2003). Web Usage mining using artificial ant colony clustering and linear genetic programming, *Genetic programming, Congress on Evolutionary Computation (CEC)*, IEEE, 2003, 1384–1391.

Balakrishnan, P. V. S., & Jacob, V. S. (1996). Genetic algorithms for product design. *Management Science, 42*(8), 1105–1117.

Bhattacharyya, S. (2000). Evolutionary algorithms in data mining: Multi-objective performance modeling for direct marketing. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 465–473). New York: ACM.

Fruchter, G.E., Fligler, A., Winer, R. S. (2006). Optimal product line design: A genetic algorithm approach to mitigate cannibalization. *Journal of Optimization Theory and Applications*, 131(2), (pp. 227-244), Springer Netherlands.

Kotler, P., & Armstrong, G. (1996), *Principles of marketing, 7* ed., Prentice Hall, Englewood Cliffs NJ.

Kwon, Y.-K., & Moon, B.-R. (2001). Personalized email marketing with a genetic programming circuit model. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)* (pp. 1352–1358). San Francisco: Morgan Kaufmann.

Liu, H.-H., & Ong, C.-S. (2008). Variable selection in clustering for marketing segmentation using genetic algorithms. *Expert Systems Applications, 34*(1), 502–510.

Naik, P. A., Mantrala, M. K., & Sawyer, A. G. (1998). Planning media schedules in the presence of dynamic advertising quality. *Marketing Science, 17*(3), 214–235.

Scanlon, Jessie. "Staples' Evolution." BusinessWeek 29 Dec. 2008: 1-2. Web, http://www.businessweek.com/innovate/content/dec2008/id20081229_162381.htm

Evolutionary Computing

►Evolutionary Algorithms

Evolutionary Constructive Induction

►Evolutionary Feature Selection and Construction

Evolutionary Feature Selection

►Evolutionary Feature Selection and Construction

Evolutionary Feature Selection and Construction

Krzysztof Krawiec
Poznan University of Technology
Poznan, Poland

Synonyms

EFSC; Evolutionary constructive induction; Evolutionary feature selection; Evolutionary feature synthesis; Genetic attribute construction; Genetic feature selection

Definition

Evolutionary feature selection and construction (EFSC) is a bio-inspired methodology for explicit modification of input data of a learning system. It uses evolutionary computation (EC) to find a mapping from the original data representation space onto a *secondary* representation space. In evolutionary feature selection (EFS), that mapping consists in dropping off some of the features (►attributes) from the original representation, so the dimensionality of the resulting representation space is not greater than that of the original space. In evolutionary feature construction (EFC), evolutionary algorithm creates (synthesizes) new features (derived attributes) that complement and/or replace the original ones. Therefore, EFS may be considered as special case of EFC.

A typical EFSC algorithm maintains a population of solutions, each of them encoding a specific mapping. The best mapping found in evolutionary search becomes the data preprocessor for the classifier. Usually, EFSC takes place in training phase only, and the evolved mapping does not undergo further changes in the testing phase.

Though EFSC is technically a form of data preprocessing (see ►Data Preparation), some of its variants may as well involve an internal inductive process in the fitness function. Also, EFS and EFC may be considered as special cases of ►Feature Selection and ►Feature Construction, respectively. EFC is also partially inspired by ►Constructive Induction.

Motivation and Background

Real-world machine learning problems often involve a multitude of attributes, which individually have low informative content and cannot provide satisfactory performance of the learning system. This applies in particular to data-abundant domains like image analysis and signal processing. When faced with many low-quality attributes, induction algorithms tend to build classifiers that perform poorly in terms of classification accuracy. This problem may be alleviated by removing some features from the original representation space (*feature selection*) or introducing new features defined as informative expressions (arithmetic, logical, etc.) built of *multiple* attributes (*feature construction*).

Unfortunately, many learning algorithms lack the ability of discovering intricate dependencies between attributes, which is a necessary precondition for successful feature selection and construction. This gap is filled out by EFSC, which uses EC to get rid of superfluous attributes and to construct new features. To this extent, anticipated benefits from EFSC are similar to those of general ►Feature Selection and ►Feature Construction, and include reduced dimensionality of the input space, better predictive accuracy of the learning system, faster training and querying, and better readability of the acquired knowledge.

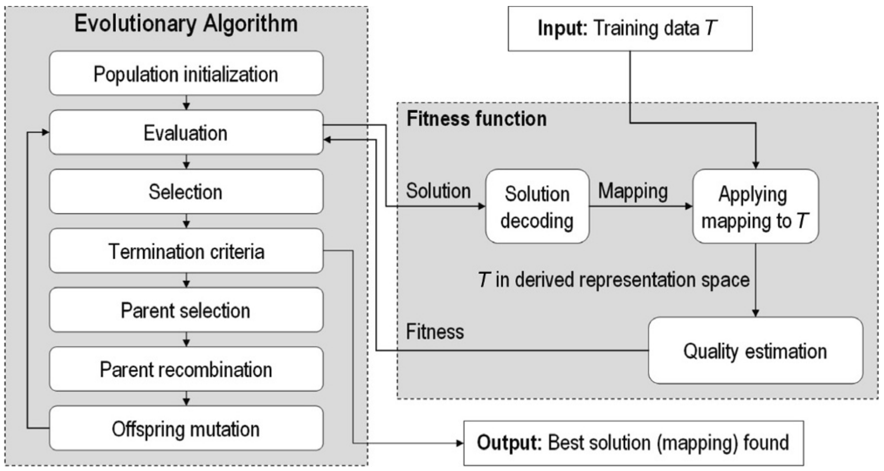
In general, both feature selection and feature construction may be conveniently formulated as an optimization problem with each solution corresponding to a particular feature subset (for feature selection) or to a particular definition of new features (for feature construction). The number of such solutions grows exponentially with the number of original features, which renders the exact search methods infeasible. Therefore, EC techniques with their ability of performing global parallel search with low risk of being trapped in local optima are particularly predisposed to solve these types of problems. Moreover, EC algorithms can optimize arbitrary function without demanding assumptions concerning solution space and objective function (like, for instance, the branch-and-bound algorithm). This is extremely important in the context of EFSC, where the so-called fitness landscape (the objective function spanned over the space of solutions) heavily depends on the training data, and it is therefore difficult to predict its properties.

The other strength of EC is the ease of adaptation to a specific task that usually boils down to the choice of solution representation and implementation of the fitness function. For instance, a subset of features in EFS may be directly encoded as a bit string solution in genetic algorithm (GA), where a bit at a particular position determines the selection or exclusion of the corresponding feature (Vafaie & Imam, 1994; Yang & Honavar, 1998). In EFC, definitions of constructed features may be conveniently represented as genetic programming (GP) expressions/procedures (Rizki, Zmuda, & Tamburino, 2002; Teller & Veloso, 1997). Also, unlike many other search algorithms, evolutionary algorithm can easily produce *many* solutions. This makes it a natural tool for, e.g., a parallel construction of multiple representations (feature subsets) that may be subsequently used in a compound classifier.

Structure of Learning System

Typically, EFSC uses a variant of evolutionary algorithm (usually GA for EFS or genetic programming for EFC) to maintain a population of solutions (individuals), each of them encoding a particular subset of features (for EFS) or definition of new features (for EFC). Solutions undergo mutations, crossing-over, and selective pressure that promotes the well-performing ones. Selective pressure is exerted by fitness function, which estimates solution's quality by measuring some properties of the secondary representation space (see Fig. 1). This usually involves three steps:

1. *Decoding* of solution (retrieving mapping from the encoded solution).
2. *Transforming* the training set into the secondary representation space according to the mapping.



Evolutionary Feature Selection and Construction. Figure 1. **Evolutionary feature selection and construction**

3. Estimating the *quality* of the secondary representation space which, after appropriate conversion (e.g., scaling), becomes solution's fitness.

Technically, step 3 usually boils down to one of two methods. *Filter approach* relies on measures that characterize the desired properties of training data in the secondary space (e.g., class separability), abstracting from any particular induction algorithm. *Wrapper approach* estimates the predictive ability that may be attained in the secondary representation space by a *specific* induction algorithm, usually by partitioning the training set into several subsets and performing multiple train-and-test experiment (e.g., cross-validation). In both cases, the particular implementation depends on the type of task being solved (classification or regression, predominantly the former one). Wrapper approach, though computationally more expensive, takes into account inductive and representational biases specific for induction algorithm, and often prove superior in terms of classification accuracy.

The result of a typical EFSC procedure is the best solution found in the evolutionary run, i.e., the superior representation mapping with respect to fitness function. This mapping serves as a preprocessor of input data and is subsequently used to induce the final classifier from the training set. The trained classifier together with the preprocessing provided by the mapping is the final outcome of the EFSC-enriched training process and may be used for classification of new examples.

EFS and EFC are predominantly applied to supervised learning from examples and attribute-value representation of training data. The above scheme remains relatively unchanged across various EFS and EFC approaches reported in literature, with main differences discussed in following.

Evolutionary Feature Selection

EFS is the simplest variant of EFSC. In this case, a solution encodes the indices of attributes that should be removed from the original representation (or, alternatively, which should be left in the resulting secondary representation). This leads to straightforward encoding characteristic for GA, with each solution being a bit string as long as the number of original attributes. EFS may be thus easily implemented using off-shelf generic software packages and involves relatively straightforward fitness function. However, more sophisticated approaches have been also considered, like evolving GP individuals to asses the quality of and rank feature subsets (Neshatian & Zhang, 2009).

Evolutionary feature weighting (EFW) is a direct generalization of EFS, where the evolutionary search weighs features instead of selecting them. Solutions in EFW are real-valued vectors evolved by evolutionary algorithm or evolutionary strategy. EFW requires use of a special wrapper fitness function that can take attribute weights into account. In (Komosiński & Krawiec, 2000), EFW has been used with a nearest neighbor-based wrapper fitness function to weigh features for a medical diagnosing problem.

Evolutionary Feature Construction

EFC requires sophisticated evolutionary representation of solutions to encode definitions of new features, and usually employs genetic programming for that purpose. Each GP solution encodes an expression tree that uses the original attributes and numeric constants as leaves (terminals), and functions from a predefined vocabulary as internal tree nodes (nonterminals). The value returned by such an expression when applied to an example is interpreted as the new feature. Function set usually encompasses simple arithmetics (typically $+$, $-$, $*$, $/$) and elementary functions (like *sin*, *cos*, *log*, *exp*). The evolved features replace or extend the original ones. As a single new feature is usually insufficient to provide satisfactory discriminative ability, it is common to encode several GP trees within each solution.

EFC may be conveniently adopted to image analysis or computer vision problems, or any other type of machine learning task that involves a large numbers of attributes. Commonly, an EFC algorithm evolves GP solutions that construct higher-level features from low-level image attributes (Krawiec & Bhanu, 2005) or implement advanced feature detectors (Howard, Roberts, & Ryan, 2006; Puente, 2009; Quintana, Poli, & Claridge, 2006). Alternatively, solutions encode chains of operations that process the entire image globally according to the goal specified by the fitness function. Many other variants of this approach have been studied in literature, involving, e.g., solutions represented as graphs (Teller & Veloso, 1997) or sequences of operations (linear genetic programming, (Bhanu et al., 2005)).

Applications

Real-world applications of EFSC are numerous and include medical and technical diagnosing, computer network intrusion detection, genetics, air quality forecasting, brain-computer interfaces, seismography, robotics, face recognition, handwriting recognition, vehicle detection in visual, infrared, and radar modality, image segmentation, satellite imaging, and stereovision. The conceptually simpler EFS has been implemented in several machine learning and neural-network software packages (WEKA, Statistica Neural Networks). EFC usually requires a more sophisticated and application-specific implementation. However, for standard learning-from-example tasks, it may be conveniently implemented by extending off-shelf libraries, like WEKA (Waikato Environment for Knowledge Analysis, http://www.cs.waikato.ac.nz/ml/weka/) and ECJ (Evolutionary Computation in Java, http://cs.gmu.edu/~eclab/projects/ecj/). More examples of real-world applications of EFSC may be found in (Langdon, Gustafson, & Koza, 2009).

Future Directions

Recent work on EFC employs various extensions of EC. It has been demonstrated that an EFC task may be decomposed into several semi-independent subtasks using cooperative coevolution, a variant of evolutionary algorithm that maintains several populations with solutions encoding partial solutions to the problem (Krawiec & Bhanu, 2005). Other recent work demonstrates that fragments of GP expressions encoding feature definitions may help to discover good features in other learning tasks (Jaśkowski, Krawiec, & Wieloch, 2007). With time, EFC becomes more and more unified with GP-based classification, where solutions are expected to perform the complete classification or regression task rather than to implement only feature definitions.

The online genetic programming bibliography (Langdon et al., 2009) provides quite complete coverage of state of the art in evolutionary feature selection and construction. A concise review of contemporary genetic programming research involving feature construction for image analysis and object detection may be found in (Krawiec, Howard, & Zhang, 2007). A more extensive and systematic study of different evolutionary approaches to feature construction is presented in (Bhanu et al., 2005).

Cross References

►Constructive Induction
►Data Preparation
►Feature Selection

Recommended Reading

Bhanu, B., Lin, Y., & Krawiec, K. (2005). *Evolutionary synthesis of pattern recognition systems*. New York: Springer-Verlag.
Howard, D., Roberts, S. C., & Ryan, C. (2006). Pragmatic genetic programming strategy for the problem of vehicle detection in

airborne reconnaissance. *Pattern Recognition Letters*, *27*(11), 1275–1288.

Jaśkowski, W., Krawiec, K., & Wieloch, B. (2007). Knowledge reuse in genetic programming applied to visual learning. In Dirk Thierens et al. (eds.), GECCO '07: In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, (Vol 2, pp. 1790–1797), London, 2007. ACM Press.

Komosiński, M., & Krawiec, K. (2000). Evolutionary weighting of image features for diagnosing of CNS tumors. *Artificial Intelligence in Medicine*, *19*(1), 25–38.

Krawiec, K., & Bhanu, B. (2005). Visual learning by coevolutionary feature synthesis. *IEEE Transactions on System, Man, and Cybernetics – Part B*, *35*(3), 409–425.

Krawiec, K., Howard, D., & Zhang, M. (2007). Overview of object detection and image analysis by means of genetic programming techniques. In *Proceedings of frontiers in the convergence of bioscience and information technologies 2007 (fbit2007), Jeju, Korea, october 11–13, 2007* (pp. 779–784). IEEE CS Press.

Langdon, W., Gustafson, S., & Koza, J. (2009). *The genetic programming bibliography*. ([online] http://www.cs.bham.ac.uk/ wbl/biblio/)

Neshatian, K., & Zhang, M. (2009). Genetic programming for feature subset ranking in binary classification problems. In L. Vanneschi, S. Gustafson, A. Moraglio, I. vanoe De Falco, & M. Ebner (Eds.), *Genetic programming* (pp. 121–132). Springer.

Puente, C., Olague, G., Smith, S. V., Bullock, S. H., González-Botello, M. A., & Hinojosa-Corona, A. (2009). A novel GP approach to synthesize vegetation indices for soil eros ion assessment. In M. Giacobini et al. (Eds.), *Applications of evolutionary computing* (pp. 375–384). Springer.

Quintana, M. I., Poli, R., & Claridge, E. (2006) Morphological algorithm design for binary images using genetic programming. *Genetic Programming and Evolvable Machines*, *7*(1), 81–102.

Rizki, M. M., Zmuda, M. A., & Tamburino, L. A. (2002). Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, *6*(6), 594–609.

Teller, A., & Veloso, M. (1997). PADO: A new learning architecture for object recognition. In K. Ikeuchi & M. Veloso (Eds.), *Symbolic visual learning* (pp. 77–112). New York: Oxford Press.

Vafaie, H., & Imam, I. F. (1994). Feature selection methods: genetic algorithms vs. greedy-like search. In *Proceedings of international conference on fuzzy and intelligent control systems*.

Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. *IEEE Transactions on Intelligent Systems*, *13*(2), 44–49.

Evolutionary Feature Synthesis

►Evolutionary Feature Selection and Construction

Evolutionary Fuzzy Systems

Carlos Kavka
University of Trieste
Trieste
Italy

Definition

An evolutionary fuzzy system is a hybrid automatic learning approximation that integrates ►fuzzy systems with ►evolutionary algorithms, with the objective of combining the optimization and learning abilities of evolutionary algorithms together with the capabilities of fuzzy systems to deal with approximate knowledge. Evolutionary fuzzy systems allow the optimization of the knowledge provided by the expert in terms of linguistic variables and fuzzy rules, the generation of some of the components of fuzzy systems based on the partial information provided by the expert, and in some cases even the generation of fuzzy systems without expert information. Since many evolutionary fuzzy systems are based on the use of genetic algorithms, they are also known as *genetic fuzzy systems*. However, many models presented in the scientific literature also use genetic programming, evolutionary programming, or evolution strategies, making the term *evolutionary fuzzy systems* more adequate. Highly related is the concept of *evolutionary neuro-fuzzy systems*, where the main difference is that the representation is based on neural networks. Recently, the related concept of *evolving fuzzy systems* has been introduced, where the main objective is to apply evolutionary techniques to the design of fuzzy systems that are adequate to the control of nonstationary processes, mainly on real-time applications.

Motivation and Background

One of the most interesting properties of a fuzzy system is its ability to represent expert knowledge by using linguistic terms of everyday common use, allowing the description of uncertainty, vagueness, and imprecision in the expert knowledge. The linguistic terms, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy theory concepts.

The usual approach to build a fuzzy system consists in the definition of the membership functions and the rule base in terms of expert knowledge. Compared with other rule-based approaches, the process of extracting knowledge from experts and representing it formally is simpler, since linguistic terms can be defined to match the terms used by the experts. In this way, rules are defined establishing relations between the input and output variables using these linguistic terms. However, even if there is a clear advantage of using the terms defined as ▶fuzzy sets, the knowledge extraction process is still difficult and time consuming, usually requiring a very difficult manual fine tuning process. It should be noted that no automatic framework to determine the parameters of the components of the fuzzy system exists yet, generating the need for methods that provide adaptability and learning ability for the design of fuzzy systems.

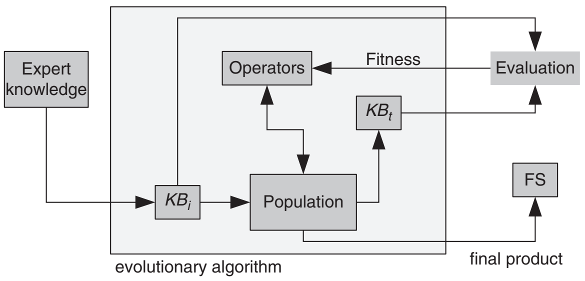
Since it is very easy to map a fuzzy system into a feedforward neural network structure, it is not surprising that many methods based on neural network learning have been proposed to automate the fuzzy system building process (Hoffmann, 2001; Karr & Gentry, 1993) The combined approach provides advantages from both worlds: the low level learning and computational power of neural networks is joined together with the high level human-like thinking and reasoning of fuzzy systems. However, this approach can still face some problems, such as the potential risk of its learning algorithms to get trapped in local minimum, the possible need for restriction of the membership functions to follow some mathematical properties (like differentiability), and the difficulties of inserting or extracting knowledge in some approaches, where the obtained linguistic terms can exhibit a poor semantic due to the usual black-box processing of many neural networks models.

Evolutionary algorithms provide a set of properties that make them ideal candidates for the optimization and design of fuzzy systems, and in fact, there are many methods that have been proposed in the literature to design or tune the different components of fuzzy systems. Evolutionary systems exhibit robust performance and global search characteristics, while requiring only a simple quality measure from the environment. There is no need for gradient information or input/output patterns. Other strengths come from its parallel nature: instead of selecting a single solution and refining it, in most evolutionary methods, a set of alternative solutions is considered and evolved in parallel.

Structure of the Learning System

The learning process defined by an evolutionary fuzzy system starts from the knowledge provided by the expert, which can include all or just some of the components of the knowledge base of a fuzzy system. The evolutionary algorithm that is behind this learning approach can perform the optimization of all the parameters that are provided by the expert, plus the generation of the missing components of the fuzzy system based on the partial specifications provided by the expert.

The model shown in Fig. 1 presents a general architecture of the learning and optimization process in evolutionary fuzzy systems. An initial knowledge base KB_i is built based on the knowledge provided by the expert. Note that KB_i could be (and usually is) a incompletely specified knowledge base. Based on this initial expert knowledge, the evolutionary algorithm creates a population of individuals, which can represent complete fuzzy systems or just a few components of them. The evaluation of the individuals is performed by creating a temporary knowledge base KB_t, which can also be complete or not. By using the information in KB_t, combined with the initial knowledge base KB_i, the individuals are evaluated by determining the error in the approximation of patterns if there are examples available, computing the reinforcement signal (typical situation in control problems), or in any other way depending on the problem characteristics (Babuska, 1998; Cordon, Gomide, Herrera, Hoffmann, & Magdalena, 2004). The



Evolutionary Fuzzy Systems. Figure 1. The general model of the evolutionary fuzzy systems learning and optimization

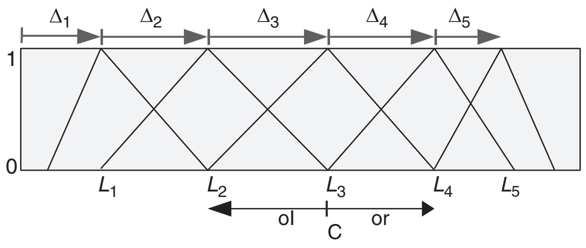
result of the evaluation is typically a single fitness measure, which provides the necessary information for the selection and the variational operators of the evolutionary algorithm. These operators, which can be standard or defined specifically for the problem, combine and mute the individuals based on the fitness value and their specific parameters. The process is repeated till a predefined criterion is fulfilled, obtaining as a final result the fuzzy system *FS*.

Depending on the information provided by the expert, the learning or optimization process performed by the evolutionary fuzzy system can be applied to the database, the fuzzy rule base or both of them. These three approaches are described below.

Optimization and Learning of the Fuzzy Database

In this case, it is assumed that the fuzzy rule base is known and provided by the expert. The initial knowledge base KB_i contains the fuzzy rule base, and if provided, the initial approximation of the parameters of antecedents and/or consequents. Since the expert has to define the rule base, and in order to do that, he/she needs to know the labels of the linguistic terms used for the antecedents and consequents, it is usual that the number of fuzzy sets is predefined and kept constant during the evolution.

The representation of the individuals contains only the parameters of the fuzzy sets associated to the input linguistic variables, and the fuzzy sets associated to the output variables in the case of a Mamdani fuzzy system, or the associated lineal approximators in the case of a Takagi-Sugeno fuzzy system. Other parameters could also be specified if necessary (scale factors, etc.). Usually, individuals are represented as a fixed length string that is defined as the concatenation of all parameters of the input and output fuzzy sets or approximators. Of course, the representation for the fuzzy sets depends on their particular class: for example, three values are required to represent triangular fuzzy sets, four values to represent trapezoidal fuzzy sets, and two for sigmoidal fuzzy sets. As an example, Fig. 2 shows that three values are necessary to represent a triangular fuzzy set: the center, the left width, and the right width, labeled as *c*, *ol*, and *od*, respectively. From this example, it can be seen that 15 values are required in order to represent the 5 fuzzy sets associated to this single linguistic variable.



Evolutionary Fuzzy Systems. Figure 2. A linguistic variable represented with five fuzzy sets

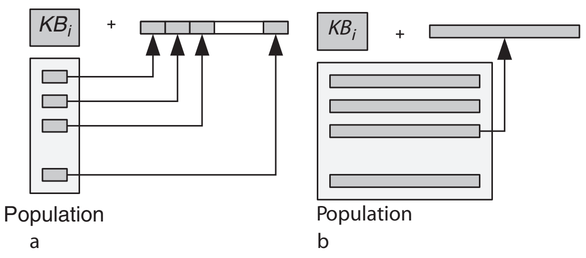
However, it is usual to apply fuzzy logic concepts (Zadeh, 1988) to simplify the representation, with the implied reduction in the search space, and also, to enhance the interpretability (Casillas, Cordon, Herrera, & Magdalena, 2003) of the resulting fuzzy system. As an example, it is desirable that the partition associated to a linguistic variable fulfills the completeness property, which establishes that for each point in the input domain, the summation of the membership values of all membership functions must be equal to 1. It is also desirable that the position of the fuzzy sets remains always the same during the evolution, for example in Fig. 2, it means that it is expected that the fuzzy set L_1 will be always at the left of L_2, L_2 always at the left of L_3, and so on. A representation that considers these two requirements can be defined by representing the whole partition specifying the distance from the center of a fuzzy set to the center of the next one (Hoffmann, 2001). The representation of five fuzzy sets then requires only five values (labeled in the figure as Δ_i), which reduces largely the search space and keeps the order of fuzzy sets, while fulfilling the completeness property. Most implementations use real values to represent the parameters.

The operators of the evolutionary algorithm can be standard operators or can be defined specifically based on the selected representation. As an example, operators that modify the width of fuzzy sets, shift the centers, or perform other operations on the fuzzy set representations, linear approximators, or other parameters have been defined in the scientific literature.

Optimization and Learning of the Fuzzy Rule Base

In this case, the fuzzy rule base is not known, or only an initial approximation to it is provided. The other parameters of the knowledge base are known and provided by the expert. The three most usual approximations are

1. Michigan approximation: Each individual of the population codifies a single rule (Bonarini, 1996), which means that each individual by itself cannot represent a complete solution to the problem. The knowledge base for evaluation KB_t is built based on the information defined in KB_i and the rules defined by all the individuals from the population combined together (see Fig. 3a). Rules are penalized or rewarded based on its performance during the evaluation. The fuzzy system is then built through the competition of a set of independent rules that have to be learned to collaborate during the evolution.
2. Pittsburgh approximation: Each individual represents the complete rule base. If dynamic creation and removal of rules is allowed, it is necessary to define special variational operators to deal with variable length individuals. Compared with the Michigan approach the evaluation is simpler, since by just combining each individual with KB_i it is possible to build KB_t for evaluation (see Fig. 3b). However, usually, the search space is larger when compared with the Michigan approach.
3. Iterative approximation: Each individual codifies a single rule (Cordon, Herrera, & Hoffmann, 2001) like in the Michigan approach. However, in each iteration of the algorithm, only the best rule is selected discarding all the others. This selection is based by considering the properties of the rule, such as for example, its covering degree on a set of examples. The algorithm is then competitive and not cooperative. It is usually necessary to apply algorithms to refine the fuzzy rule set obtained at the end of the evolutionary process, which can include operations, such as for example, the removal of similar rules.



Evolutionary Fuzzy Systems. Figure 3. The evaluation of individuals in the (a) Michigan and (b) Pittsburgh approaches

The representation in all of these approximations usually consists of individuals that contain references to the fuzzy sets already defined in KB_i. The representation of each individual can be a sequence of integers where each one is an index to the fuzzy sets associated to the corresponding linguistic variable. As an example, the fuzzy rule base could be represented as a matrix where each cell corresponds to the intersection of the input fuzzy sets, containing the index of the output fuzzy set associated to the rule. It is also possible to represent the fuzzy rule base as a decision table or simply as a list of rules. In these last two cases, the representation can have variable length, allowing to represent fuzzy rule sets with variable size.

The fitness calculation depends on the selected approximation. On a Pittsburgh approximation, the fitness corresponds to the evaluation of the complete fuzzy system on the corresponding problem. It is also possible to include in the fitness calculation other factors, such as for example, penalization for fuzzy rule bases that contains many rules or fuzzy rules with superposed application areas, etc. On a Michigan or Iterative model, the fitness indicates the degree of adequacy of the rule measured independently, considering also in the Michigan model its degree of cooperation with the other rules in the population.

The definition of the variational operators depends of course on the selected approximation. If the representation allows it, standard operators of crossover and mutation can be used. However, it can be convenient (or necessary) to define specific operators. As an example, variational operators can consider factors such as the time period since the rule has been used for the last time, its overall contribution to the final result, its performance when evaluated on the set of examples, etc.

Optimization and Learning of the Complete Knowledge Base

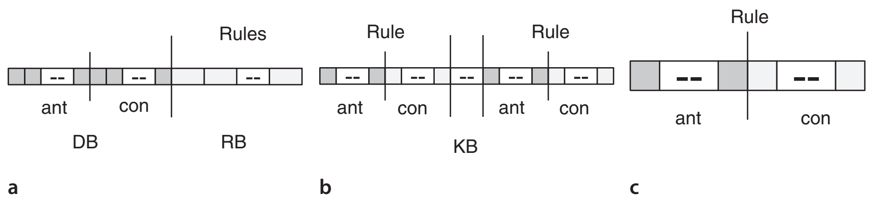
This case is a combination of the two models described before. The knowledge base KB_i contains the initial approximation to the definition of the antecedents and consequents, and the initial approximation to the fuzzy rule base as provided by the expert. Note that KB_i can also be empty if it is expected that the algorithm must generate all the parameters of the fuzzy system by itself.

The representation of the individuals contains all the parameters that define a knowledge base in order to allow its learning or optimization. The three most used representation schemes are shown in Fig. 4. In the first scheme, each individual contains the representation of all fuzzy sets, and the representation of all fuzzy rules using indexes to refer to the corresponding fuzzy sets. In the second scheme, each individual is structured as a set of rules, where each one specifies its own input and output fuzzy sets by directly including the parameters that define them. The representation (a) is adequate for descriptive fuzzy systems, since the rules contain references to the fuzzy sets used in their definition and can be shared by all of them. The representation (b) is adequate for approximative fuzzy systems, where each rule defines its own fuzzy sets. These two representations are adequate for the Pittsburgh approximation, while the third one (c) is adequate for the Michigan and the Iterative approximation. Of course, there can be many variations of this representations. For example, the input space partition can be predefined or obtained through fuzzy clustering algorithms, and if this partition is not expected to go under optimization, then it is not necessary to include the parameters of the input fuzzy sets in the representation.

Since this model is a combination of the two previous models, everything that was mentioned before concerning the fitness function and the variational operators also applies in this context. However, the fact that all parameters of the knowledge base are included in the representation allows to define more powerful variational operators. As an example, it is possible to define operators that decide the creation of new fuzzy sets, the elimination of some of them, and at the same time, the adaptation of the associated fuzzy rules, when for example, it is detected that there are areas in the input space that are not well covered, many rules with superimposed areas, etc. It is also possible to apply genetic programming techniques (Pedrycz, 2003), which are usually used to modify the structure of the fuzzy system, adding, removing, or combining sections of the fuzzy system with the objective of generating the most adequate structure.

Final Remarks

Clearly, the integration of fuzzy systems with evolutionary algorithms allows to overcome the limitations of each model considered independently, obtaining a powerful hybrid approach, which allows to learn and optimize fuzzy systems based on expert knowledge. Previous sections have discussed in general terms the evolutionary learning model. However, in order to get more details about particular implementations, it is recommended to read the publications referenced in the next section. The presentation from Karr & Gentry (1993) is interesting, not only because it provides a nice introduction and application of evolutionary fuzzy systems, but it has the additional value of being one of the first publications in the area. The presentation of Hoffmann (2001) is an excellent introduction to evolutionary fuzzy systems used for control applications. The other publications present details on evolutionary fuzzy systems (Babuska 1998; Bonarini 1996; Cordon et al., 2001; Juang Lin & Lin 2000; Lee & Takagi 1993), including representations based on neural networks (Hoffmann, 2001; Karr & Gentry, 1993), evolution strategies (Alpaydtn, Dundar, & Balktr, 2002), genetic programming (Pedrycz, 2003) and applications of evolutionary fuzzy systems to the domain of recurrent



Evolutionary Fuzzy Systems. Figure 4. Representations for the complete knowledge base adequate for (a) descriptive and (b) approximative fuzzy systems in the Pittsburgh approximation, and (c) representation of a single independent rule adequate for Michigan and Iterative approximations

fuzzy systems (Kavka, Roggero, & Schoenauer, 2005). The paper by Cordon et al. (2004) provides a very comprehensive reference list about the main developments on evolutionary fuzzy systems.

It should be stressed that a very important aspect to consider in the definition of evolutionary fuzzy systems is the interpretability of the resulting fuzzy systems (Casillas et al., 2003). Even if it has been mentioned that it is possible to design an evolutionary fuzzy system without expert information, by allowing the evolutionary algorithm to define all the components of the knowledge base by itself, it must always be considered that the interpretability of the results is essential. Designing a system that solves the problem, but that works as a black box, can be adequate in other contexts, but it is not desirable at all in the context of evolutionary fuzzy systems. An evolutionary fuzzy system algorithm must provide the means so that the expert knowledge defined in fuzzy terms can be considered and used appropriately during the evolution, and also, it must guarantee an adequate interpretability degree of the resulting fuzzy system.

Recommended Reading

Alpaydtn, G., Dundar, G., & Balktr, S. (2002). Evolution-based design of neural fuzzy networks using self-adapting genetic parameters. *IEEE Transactions of Fuzzy Systems, 10*(2), 211–221.

Babuska, R. (1998). *Fuzzy modeling for control*. Norwell, MA: Kluwer Academic Press.

Bonarini, A. (1996). Evolutionary learning of fuzzy rules: Competition and cooperation. In W. Pedrycz (Ed.), *Fuzzy modeling: Paradigms and practice*. Norwell, MA: Kluwer Academic Press.

Casillas, J., Cordon, O., Herrera, F., & Magdalena, L. (Eds.). (2003). *Interpretability issues in fuzzy modeling. Series: Studies in fuzziness and soft computing* (Vol. 128)

Cordon, O., Gomide, F., Herrera, F., Hoffmann, F., & Magdalena, L. (2004). Ten years of genetic fuzzy systems: Current framework and new trends. *Fuzzy Sets and Systems, 141*, 5–31.

Cordon, O., Herrera, F., & Hoffmann, F. (2001). *Genetic fuzzy systems*. Singapore: World Scientific Publishing.

Hoffmann, F. (2001). Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE, 89*(9), 1318–1333.

Juang C. F., Lin, J. Y., & Lin, C. T. (2000). Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. *IEEE Transactions on Systems, Man and Cybernetics, 30*(2), 290–302.

Karr, C. L., & Gentry, E. J. (1993). Fuzzy control of PH using genetic algorithms. *IEEE Transactions on Fuzzy Systems, 1*(1), 46–53.

Kavka, C., Roggero, P., & Schoenauer, M. (2005). Evolution of Voronoi based fuzzy recurrent controllers. In *Proceedings of GECCO* (pp. 1385–1392). NeW York: ACM Press.

Lee, M., & Takagi, H. (1993). Integrating design stages of fuzzy systems using genetic algorithms. In *Proceedings of the second IEEE international conference on fuzzy systems* (pp. 612–617).

Pedrycz, W. (2003). Evolutionary fuzzy modeling. *IEEE Transactions of Fuzzy Systems, 11*(5), 652–665.

Zadeh, L. (1988). Fuzzy logic. *IEEE Computer, 21*(4), 83–93.

Evolutionary Games

Moshe Sipper
Ben-Gurion University
Beer-Sheva, Israel

Definition

Evolutionary algorithms are a family of algorithms inspired by the workings of evolution by natural selection, whose basic structure is to

1. Produce an initial *population* of individuals, these latter being candidate solutions to the problem at hand
2. Evaluate the *fitness* of each individual in accordance with the problem whose solution is sought
3. *While* termination condition not met *do*
   a. *Select* fitter individuals for reproduction
   b. *Recombine* (*crossover*) individuals
   c. *Mutate* individuals
   d. *Evaluate* fitness of modified individuals
4. *End while*

Evolutionary games is the application of evolutionary algorithms to the evolution of game-playing strategies for various games, including chess, backgammon, and Robocode.

Motivation and Background

Ever since the dawn of artificial intelligence in the 1950s, games have been part and parcel of this lively field. In 1957, a year after the Dartmouth Conference that marked the official birth of AI, Alex Bernstein designed a program for the IBM 704 that played two amateur games of chess. In 1958, Allen Newell, J.C. Shaw, and Herbert Simon introduced a more sophisticated chess program (beaten in thirty-five moves by a ten-year-old beginner in its last official game played in 1960).

Arthur L. Samuel of IBM spent much of the 1950s working on game-playing AI programs, and by 1961 he had a checkers program that could play at the master's level. In 1961 and 1963, Donald Michie described a simple trial-and-error learning system for learning how to play Tic-Tac-Toe (or Noughts and Crosses) called MENACE (for Matchbox Educable Noughts and Crosses Engine). These are but examples of highly popular games that have been treated by AI researchers since the field's inception.

Why study games? This question was answered by Susan L. Epstein, who wrote:

▸ There are two principal reasons to continue to do research on games... First, human fascination with game playing is long-standing and pervasive. Anthropologists have cataloged popular games in almost every culture... Games intrigue us because they address important cognitive functions... The second reason to continue game-playing research is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards (Epstein, 1999).

Studying games may thus advance our knowledge in both cognition and artificial intelligence, and, last but not least, games possess a competitive angle which coincides with our human nature, thus motivating both researcher and student alike.

Even more strongly, Laird and van Lent proclaimed that,

▸ ...interactive computer games are the killer application for human-level AI. They are the application that will soon need human-level AI, and they can provide the environments for research on the right kinds of problems that lead to the type of the incremental and integrative research needed to achieve human-level AI (Laird & van Lent, 2000).

Recently, evolutionary algorithms have proven a powerful tool that can automatically "design" successful game-playing strategies for complex games (Azaria & Sipper, 2005a,b; Hauptman & Sipper, 2005b, 2007a,b; Shichel et al., 2005; Sipper et al., 2007).

Structure of the Learning System

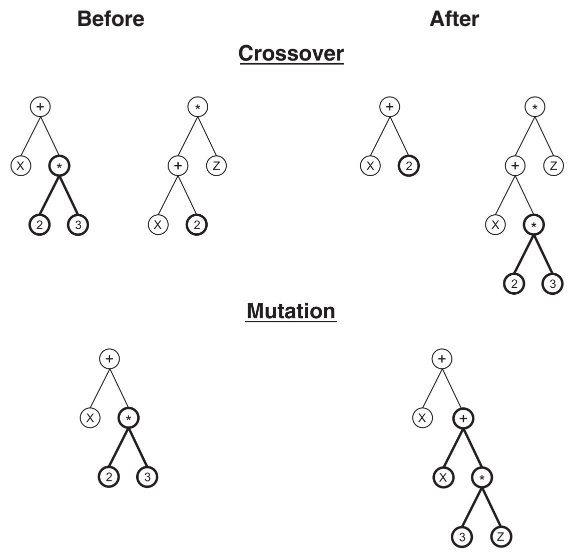
Genetic Programming

Genetic Programming is a subclass of evolutionary algorithms, wherein a *population* of individual programs is evolved, each program comprising *functions* and *terminals*. The functions are usually arithmetic and logic operators that receive a number of arguments as input and compute a result as output; the terminals are zero-argument functions that serve both as constants and as sensors, the latter being a special type of function that queries the domain environment.

The main mechanism behind genetic programming is precisely that of a generic evolutionary algorithm (Sipper, 2002; Tettamanzi & Tomassini, 2001), namely, the repeated cycling through four operations applied to the entire population: evaluate-select-crossover-mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a *fitness* value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, (as in nature) variability exists, and through the mechanism of natural (or, in our case, artificial) selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

Specifically, first a genetic operator is chosen at random; then, depending on the operator, one or two individuals are selected from the current population using a *selection operator*, one example of which is *tournament selection*: Randomly choose a small subset of individuals, and then select the one with the best fitness. After the probabilistic selection of better individuals the chosen genetic operator is used to construct the next generation. The most common operators are

- Reproduction (unary): Copy one individual to the next generation with no modifications. The main purpose of this operator is to preserve a small number of good individuals.
- Crossover (binary): Randomly select an internal node in each of the two individuals and swap the subtrees rooted at these nodes. An example is shown in Fig. 1.



Evolutionary Games. Figure 1. Genetic operators in genetic programming. LISP programs are depicted as trees. Crossover (*top*): Two subtrees (marked in *bold*) are selected from the parents and swapped. Mutation (*bottom*): A subtree (marked in *bold*) is selected from the parent individual and removed. A new subtree is grown instead

- Mutation (unary): Randomly select a node from the tree, delete the subtree rooted at that node, and then "grow" a new subtree in its stead. An example is shown in Fig. 1 (the growth operator as well as crossover and mutation are described in detail in Koza, 1992).

The generic genetic programming flowchart is shown in Fig. 2. When one wishes to employ genetic programming, one needs to define the following six desiderata:

1. Program architecture
2. Set of terminals
3. Set of functions
4. Fitness measure
5. Control parameters
6. Manner of designating result and terminating run

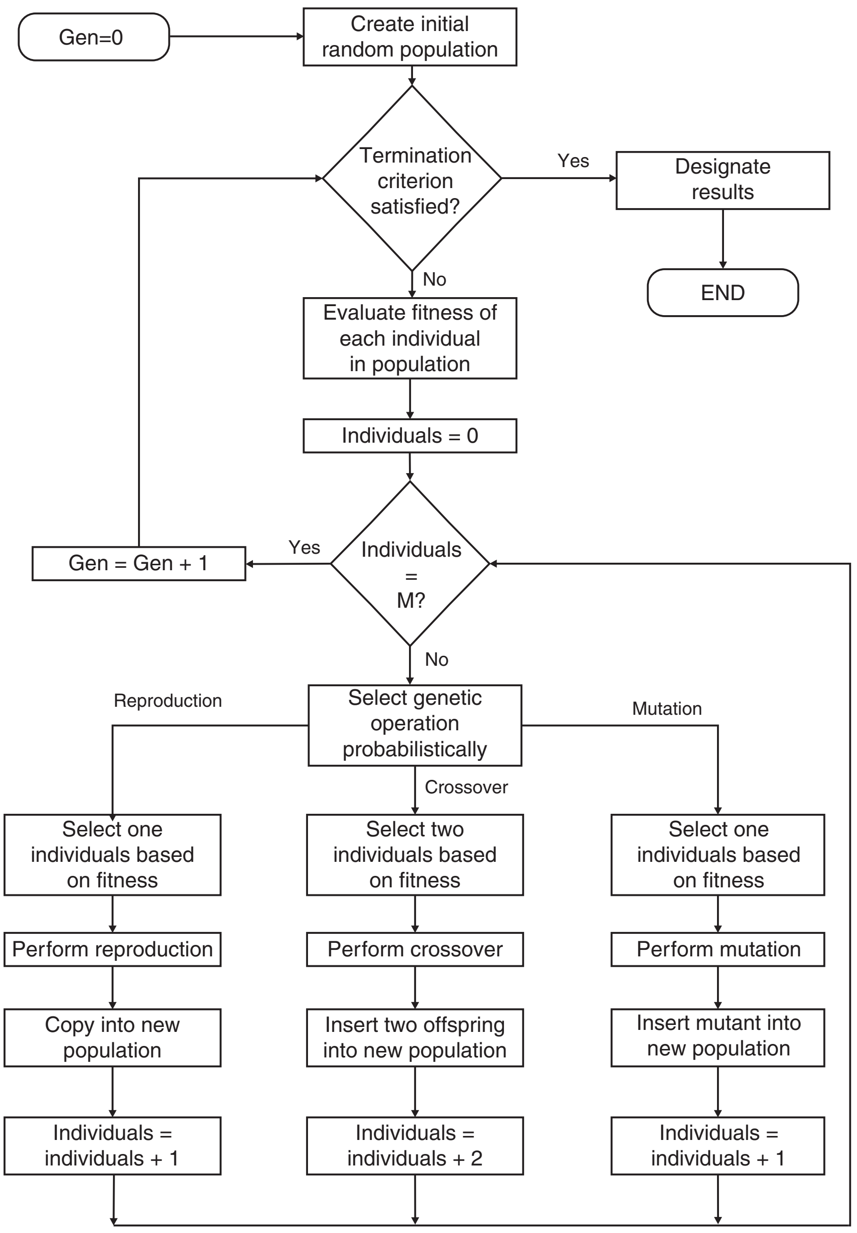
Evolving Game-Playing Strategies

Recently, we have shown that complex and successful game-playing strategies can be attained via genetic programming. We focused on three games (Azaria & Sipper, 2005a,b; Hauptman & Sipper, 2005b, 2007a,b; Shichel et al., 2005; Sipper et al., 2007):

1. *Backgammon.* Evolves a full-fledged player for the non-doubling-cube version of the game (Azaria & Sipper, 2005a,b; Sipper et al., 2007).
2. *Chess* (endgames). Evolves a player able to play endgames (Hauptman & Sipper, 2005b, 2007a,b; Sipper et al., 2007). While endgames typically contain but a few pieces, the problem of evaluation is still hard, as the pieces are usually free to move all over the board, resulting in complex game trees – both deep and with high branching factors. Indeed, in the chess lore much has been said and written about endgames.
3. *Robocode.* A simulation-based game in which robotic tanks fight to destruction in a closed arena (robocode.alphaworks.ibm.com). The programmers implement their robots in the Java programming language, and can test their creations either by using a graphical environment in which battles are held, or by submitting them to a central Web site where online tournaments regularly take place. Our goal here has been to evolve Robocode players able to rank high in the international league (Shichel et al., 2005; Sipper et al., 2007).

A strategy for a given player in a game is a way of specifying which choice the player is to make at every point in the game from the set of allowable choices at that point, given all the information that is available to the player at that point (Koza, 1992). The problem of discovering a strategy for playing a game can be viewed as one of seeking a computer program. Depending on the game, the program might take as input the entire history of past moves or just the current state of the game. The desired program then produces the next move as output. For some games one might evolve a complete strategy that addresses every situation tackled. This proved to work well with Robocode, which is a dynamic game, with relatively few parameters and little need for past history.

In a two-player game, such as chess or backgammon, players move in turn, each trying to win against the opponent according to specific rules (Hong, Huang, &


Gen=0
Create initial random population
Termination criterion satisfied?
Yes
Designate results
END
No
Evaluate fitness of each individual in population
Individuals = 0
Individuals = M?
Yes
Gen = Gen + 1
No
Select genetic operation probabilistically
Reproduction
Crossover
Mutation
Select one individuals based on fitness
Select two individuals based on fitness
Select one individuals based on fitness
Perform reproduction
Perform crossover
Perform mutation
Copy into new population
Insert two offspring into new population
Insert mutant into new population
Individuals = individuals + 1
Individuals = individuals + 2
Individuals = individuals + 1

Evolutionary Games. Figure 2. Generic genetic programming flowchart (based on Koza, 1992). M is the population size, and Gen is the generation counter. The termination criterion can be the completion of a fixed number of generations or the discovery of a good-enough individual

Lin, 2001). The course of the game may be modeled using a structure known as an adversarial game tree (or simply game tree), in which nodes are the positions in the game and edges are the moves. By convention, the two players are denoted as MAX and MIN, where MAX is the player who moves first. Thus, all nodes at odd-numbered tree levels are game positions where MAX moves next (labeled MAX nodes). Similarly, nodes on even levels are called MIN nodes, and represent positions in which MIN (opponent) moves next.

The complete game tree for a given game is the tree starting at the initial position (the root) and containing all possible moves (edges) from each position. *Terminal nodes* represent positions where the rules of the game determine whether the result is a win, a draw, or a loss. Although the game tree for the initial position is an explicit representation of all possible paths of the game, therefore theoretically containing all the information needed to play perfectly, for most (nontrivial) games it is extremely large, and constructing it is not feasible. For example, the complete chess game tree consists of roughly 10^{43} nodes (Shannon, 1950).

When the game tree is too large to be generated completely, only a partial tree (called a search tree) is generated instead. This is accomplished by invoking a *search algorithm*, deciding which nodes are to be developed at any given time and when to terminate the search (typically at nonterminal nodes due to time constraints). During the search, some nodes are evaluated by means of an *evaluation function* according to given heuristics. This is done mostly at the leaves of the tree. Furthermore, search can start from any position and not just at the beginning of the game.

Because we are searching for a winning strategy, we need to find a good next move for the current player, such that no matter what the opponent does thereafter, the player's chances of winning the game are as high as possible. A well-known method called the *minimax* search (Campbell & Marsland, 1983; Kaindl, 1988) has traditionally been used, and it forms the basis for most methods still in use today. This algorithm performs a depth-first search (the depth is usually predetermined), applying the evaluation function to the leaves of the tree, and propagating these values upward according to the minimax principal: at MAX nodes, select the maximal value, and at MIN nodes – the minimal value. The value is ultimately propagated to the position from which the search had started.

With games such as backgammon and chess one can couple a current-state evaluator (e.g., board evaluator) with a next-move generator. One can then go on to create a minimax tree, which consists of all possible moves, counter moves, counter countermoves, and so on; for real-life games, such a tree's size quickly becomes prohibitive. The approach we used with backgammon and chess is to derive a very shallow, single-level tree, and evolve "smart" evaluation functions. Our artificial player is thus created by combining an evolved board evaluator with a simple program that generates all next-move boards (such programs can easily be written for backgammon and chess).

In what follows, we describe the definition of the six items necessary in order to employ genetic programming, as delineated in the previous section: program architecture, set of terminals, set of functions, fitness measure, control parameters, and manner of designating result and terminating run. Due to lack of space we shall elaborate upon one game – Robocode – and only summarize the major results for backgammon and chess.

Example: Robocode

Program Architecture A Robocode player is written as an event-driven Java program. A main loop controls the tank activities, which can be interrupted on various occasions, called *events*. The program is limited to four lines of code, as we were aiming for the HaikuBot category, one of the divisions of the international league with a four-line code limit. The main loop contains one line of code that directs the robot to start turning the gun (and the mounted radar) to the right. This insures that within the first gun cycle, an enemy tank will be spotted by the radar, triggering a *ScannedRobotEvent*. Within the code for this event, three additional lines of code were added, each controlling a single actuator and using a single numerical input that was supplied by a genetic programming-evolved subprogram. The first line instructs the tank to move to a distance specified by the first evolved argument. The second line instructs the tank to turn to an azimuth specified by the second evolved argument. The third line instructs the gun (and radar) to turn to an azimuth specified by the third evolved argument (Fig. 3).

Terminal and Function Sets We divided the terminals into three groups according to their functionality (Shichel et al., 2005) (Fig. 4):

1. Game-status indicators: A set of terminals that provide real-time information on the game status, such as last enemy azimuth, current tank position, and energy levels.
2. Numerical constants: Two terminals, one providing the constant 0 and the other being an ephemeral random constant (ERC). This latter terminal is initialized to a random real numerical value in the range [-1, 1], and does not change during evolution.

```
Robocode Player's Code Layout

while (true)
    TurnGunRight(INFINITY); //main code loop
...
OnScannedRobot(){
    MoveTank(<GP#1>);
    TurnTankRight(<GP#2>);
    TurnGunRight(<GP#3>);
}
```

Evolutionary Games. Figure 3. Robocode player's code layout (HaikuBot division)

Energy()	Returns the remaining energy of the player
Heading()	Returns the current heading of the player
X()	Returns the current horizontal position of the player
Y()	Returns the current vertical position of the player
MaxX()	Returns the horizontal battlefield dimension
MaxY()	Returns the vertical battlefield dimension
EnemyBearing()	Returns the current enemy bearing, relative to the current player's heading
EnemyDistance()	Returns the current distance to the enemy
EnemyVelocity()	Returns the current enemy's velocity
EnemyHeading()	Returns the current enemy heading, relative to the current player's heading
EnemyEnergy()	Returns the remaining energy of the enemy
Constant()	An ERC (Ephemeral Random Constant) in the range [-1,1]
Random()	Returns a random real number in the range [-1,1]
Zero()	Returns the constant 0

a

Add(F, F)	Add two real numbers
Sub(F, F)	Subtract two real numbers
Mul(F, F)	Multiply two real numbers
Div(F, F)	Divide first argument by second, if denominator non-zero, otherwise return zero
Abs(F)	Absolute value
Neg(F)	Negative value
Sin(F)	Sine function
Cos(F)	Cosine function
ArcSin(F)	Arcsine function
ArcCos(F)	Arccosine function
IfGreater(F, F, F, F)	If first argument greater than second, return value of third argument, else return value of fourth argument
IfPositive(F, F, F)	If first argument is positive, return value of second argument, else return value of third argument
Fire(F)	If argument is positive, execute fire command with argument as fire-power and return 1; otherwise, do nothing and return 0

b

Evolutionary Games. Figure 4. Robocode representation. (a) Terminal set (b) Function set (F: Float)

3. Fire command: This special function is used to curtail one line of code by not implementing the fire actuator in a dedicated line.

Fitness Measure We explored two different modes of learning: using a fixed external opponent as teacher, and coevolution – letting the individuals play against each other; the former proved better. However, not just one but three external opponents were used to measure performance; these adversaries were downloaded from the HaikuBot league (robocode.yajags.com). The fitness value of an individual equals its average fractional score (over three battles).

Control Parameters and Run Termination The major evolutionary parameters (Koza, 1992) were population size – 256, generation count – between 100 and 200, selection method – tournament, reproduction probability – 0, crossover probability – 0.95, and mutation probability – 0.05. An evolutionary run terminates when fitness is observed to level off. Since the game is highly nondeterministic a "lucky" individual might attain a higher fitness value than better overall individuals. In order to obtain a more accurate measure for the evolved players, we let each of them do battle for 100 rounds against 12 different adversaries (one at a time). The results were used to extract the top player – to be submitted to the international league.

Results We submitted our top player to the HaikuBot division of the international league. At its very first tournament it came in third, later climbing to first place of 28 (robocode.yajags.com/20050625/haiku-1v1.html). All other 27 programs, defeated by our evolved strategy, were written by humans. For more details on GP-Robocode see Shichel et al., (2005) and Azaria, Hauptman, and Shichel (2007).

Backgammon and Chess: Major Results

Backgammon We pitted our top evolved backgammon players against *Pubeval*, a free, public-domain board evaluation function written by Tesauro. The program – which plays well – has become the *de facto* yardstick used by the growing community of backgammon-playing program developers. Our top evolved player was able to attain a win percentage of 62.4% in a tournament against Pubeval, about 10% higher (!) than the previous top method. Moreover, several evolved strategies were able to surpass the 60% mark, and most of them outdid all previous works. For more details on GP-Gammon, see Azaria and Sipper (2005a) and Azaria et al. (2007).

Chess (endgames) We pitted our top evolved chess-endgame players against two very strong external opponents: (1) A program we wrote ("Master") based upon consultation with several high-ranking chess players (the highest being Boris Gutkin, ELO 2400, International Master); (2) CRAFTY – a world-class chess program, which finished second in the 2004 World Computer Speed Chess Championship (www.cs.biu.ac.il/games/). Speed chess ("blitz") involves a time-limit per move, which we imposed both on CRAFTY and on our players. Not only did we thus seek to evolve good players, but ones who play well *and fast*. Results are shown in Table 1. As can be seen, GP-EndChess manages to hold its own, and even win, against these top players. For more details on GP-EndChess see Azaria et al., (2007) and Hauptman and Sipper (2005b).

Evolutionary Games. Table 1 Percent of wins, advantages, and draws for the best GP-EndChess player in the tournament against two top competitors

	%Wins	%Advs	%Draws
Master	6.00	2.00	68.00
CRAFTY	2.00	4.00	72.00

Deeper analysis of the strategies developed (Hauptman & Sipper, 2005a) revealed several important shortcomings, most of which stemmed from the fact that they used deep knowledge and little search (typically, they developed only *one* level of the search tree). Simply increasing the search depth would not solve the problem, since the evolved programs examine each board very thoroughly, and scanning many boards would increase time requirements prohibitively. And so we turned to evolution to find an optimal way to overcome this problem: How to add more search at the expense of less knowledgeable (and thus less time-consuming) node evaluators, while attaining better performance. In Hauptman and Sipper (2007b) *we evolved the search algorithm itself*, focusing on the *Mate-In-N* problem: find a key move such that even with the best possible counterplays, the opponent cannot avoid being mated in (or before) move *N*. We showed that our evolved search algorithms successfully solve several instances of the Mate-In-N problem, for the hardest ones developing 47% less game-tree nodes than CRAFTY. Improvement is thus not over the basic alpha-beta algorithm, but over a world-class program using all standard enhancements (Hauptman & Sipper, 2007b).

Finally, in Hauptman and Sipper (2007a), we examined a strong evolved chess-endgame player, focusing on the player's emergent capabilities and tactics in the context of a chess match. Using a number of methods we analyzed the evolved player's building blocks and their effect on play level. We concluded that evolution has found combinations of building blocks that are far from trivial and cannot be explained through simple combination – thereby indicating the possible emergence of complex strategies.

Cross References

►Evolutionary Computation
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Azaria, Y., & Sipper, M. (2005a). GP-Gammon: Genetically programming backgammon players. *Genetic Programming and Evolvable Machines, 6*(3), 283–300.

Azaria, Y., & Sipper, M. (2005b). GP-Gammon: Using genetic programming to evolve backgammon players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Proceedings of 8th European conference on genetic programming (EuroGP2005), LNCS* (Vol. 3447, pp. 132–142). Heidelberg: Springer.

Campbell, M. S., & Marsland, T. A. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence, 20*, 347–367.

Epstein, S. L. (1999). Game playing: The next moves. In *Proceedings of the sixteenth National conference on artificial intelligence* (pp. 987–993). Menlo Park, CA: AAAI Press.

Hauptman, A., & Sipper, M. (2005a). Analyzing the intelligence of a genetically programmed chess player. In *Late breaking papers at the 2005 genetic and evolutionary computation conference*, GECCO 2005.

Hauptman, A., & Sipper, M. (2005b). GP-EndChess: Using genetic programming to evolve chess endgame players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Proceedings of 8th European conference on genetic programming (EuroGP2005), LNCS* (Vol. 3447, pp. 120–131). Heidelberg: Springer.

Hauptman, A., & Sipper, M. (2007a). Emergence of complex strategies in the evolution of chess endgame players. *Advances in Complex Systems, 10*(Suppl. 1), 35–59.

Hauptman, A., & Sipper, M. (2007b). Evolution of an efficient search algorithm for the mate-in-N problem in chess. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, & A. I. Esparcia-Alcázar (Eds.), *Proceedings of 10th European conference on genetic programming (EuroGP2007), LNCS* (Vol. 4445, pp. 78–89). Heidelberg: Springer.

Hong, T.-P., Huang, K.-Y., & Lin, W.-Y. (2001). Adversarial search by evolutionary computation. *Evolutionary Computation, 9*(3), 371–385.

Kaindl, H. (1988). Minimaxing: Theory and practice. *AI-Magazine, 9*(3), 69–76.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Laird, J. E., & van Lent, M. (2000). Human-level AI's killer application: Interactive computer games. In *AAAI-00: Proceedings of the 17th National conference on artificial intelligence* (pp. 1171–1178). Cambridge, MA: MIT Press.

Shannon, C. E. (1950). Automatic chess player. *Scientific American, 48*, 182.

Shichel, Y., Ziserman, E., & Sipper, M. (2005). GP-Robocode: Using genetic programming to evolve robocode players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Proceedings of 8th European conference on genetic programming (EuroGP2005), LNCS* (Vol. 3447, pp. 143–154). Heidelberg: Springer.

Sipper, M. (2002). *Machine nature: The coming age of bio-inspired computing*. New York: McGraw-Hill.

Sipper, M., Azaria, Y., Hauptman, A., & Shichel, Y. (2007). Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37*(4), 583–593.

Tettamanzi, A., & Tomassini, M. (2001). *Soft computing: Integrating evolutionary, neural, and fuzzy systems*. Berlin: Springer.

Evolutionary Grouping

►Evolutionary Clustering

Evolutionary Kernel Learning

Christian Igel
Ruhr-Universität Bochum
Bochum, Institute für Neuroinformatik Germany

Definition

Evolutionary kernel learning stands for using ►evolutionary algorithms to optimize the ►kernel function for a kernel-based learning machine.

Motivation and Background

In kernel-based learning algorithms the kernel function determines the scalar product and thereby the metric in the feature space in which the learning algorithm operates. The kernel is usually not adapted by the ►kernel method itself. Choosing the right kernel function is crucial for the training accuracy and generalization capabilities of the learning machine. It may also influence the runtime and storage complexity during learning and application.

Finding an appropriate kernel is a ►model selection problem. The kernel function is selected from an a priori fixed class. When a parameterized family of kernel functions is considered, kernel adaptation reduces to finding an appropriate parameter vector. In practice, the most frequently used method to determine these values is grid search. In simple grid search the parameters are varied independently with a fixed step-size through a range of values and the performance of every combination is measured. Because of its computational complexity, grid search is only suitable for the adjustment of a few parameters. Further, the choice of the discretization of the search space may be crucial. Gradient-based approaches are perhaps the most elaborate techniques for adapting real-valued kernel parameters, see the articles by Chapelle, Vapnik, Bousquet, and Mukherjee (2002) and Glasmachers and Igel (2005) and references therein. To use these methods, however, the class of kernel functions must have a differentiable structure. They are also not directly applicable if the score function for assessing the parameter performance is not differentiable. This excludes some reasonable performance measures. Evolutionary kernel learning does not suffer from these limitations. Additionally, it allows for ▶multi-objective optimization (MOO).

Structure of Learning System

Canonical evolutionary kernel learning can be described as an evolutionary algorithm (EA) in which the individuals encode kernel functions, see Fig. 1. These individuals are evaluated by determining the task-specific performance of the kernel they represent. Two special aspects must be considered when designing an EA for kernel learning. First, one must decide how to assess the performance (i.e., the fitness) of a particular kernel. That is, model selection criteria have to be defined depending on the problem at hand. Second, one must also specify the subset of possible kernel functions in which the EA should search. This leads to the questions of how to encode these kernels and which variation operators to employ.

Assessing Fitness: Model Selection Criteria

The following presents some performance indices that have been considered for kernel selection. They can be used alone or in linear combination for single-objective optimization. In MOO a subset of these criteria can be used as different objectives.

It is important to note that, although many of these measures are designed to improve ▶generalization, kernel learning can lead to ▶overfitting if only limited data is used in the model selection process (e.g., in every generation the same small data sets are used to assess performance). Regularization (e.g., in a Bayesian framework) can be used to prevent overfitting. If enough data are available, it is advisable to monitor the generalization behavior of kernel learning using independent data. For example, external data can be used for the early stopping of evolutionary kernel learning.

Accuracy on Sample Data The most straightforward way to evaluate a model is to consider its performance on sample data. The empirical risk given by the error on the training data could be considered, but it does not measure generalization. To estimate the generalization performance, the accuracy on data not used for training is evaluated. In the simplest case, the available data is split into a training and validation set, with the first used for learning and the second for subsequent performance assessment. A theoretically sound and simple method is ▶cross-validation (CV). Cross-validation makes better use of the data, but it is more computationally demanding. In practice, it yields very good results.

If ▶classification is considered, it may be reasonable to split the classification error into false negative

```
initialize parent population of individuals,
each encoding kernel and perhaps additional parameters
while termination criterion is not met
    create off spring individuals from parents
    using variation operators
    train and evaluate kernel machine encoded by individuals
    using sample data
    select new parent population based on evaluation
```

Evolutionary Kernel Learning. Figure 1. Canonical evolutionary kernel learning algorithm

and false positive rates and to view ▸sensitivity and ▸specificity as two separate objectives (Suttorp & Igel, 2006).

Measures Derived from Bounds on the Generalization Performance Statistical learning theory allows one to compute estimates of and bounds on the expected generalization error of learning machines. These values can be utilized as criteria for model selection, although then the assumptions of the underlying theorems from statistical learning theory are typically violated and the terms "bound" and "unbiased estimate" become misleading.

For example, radius-margin bounds were used to evolve kernels for ▸support vector machines (SVMs) for classification (Igel, 2005). Furthermore, the number of support vectors (SVs) was optimized in combination with the empirical risk (Igel, 2005). The fraction of SVs is an upper bound on the leave-one-out error (e.g., Chapelle et al., 2002).

Number of Input Variables Variable selection refers to the ▸feature selection problem of choosing input variables that are best suited for the learning task. Masking a subset of variables can be viewed as modifying the kernel. By considering only a subset of feature dimensions the computational complexity of the learning machine decreases. When deteriorating feature dimensions are removed, the overall performance may increase. Reducing the number of input variables is therefore a common objective, which can be achieved using single-objective (Eads et al., 2002; Fröhlich, Chapelle, & Schölkopf, 2004; Jong, Marchiori, & van der Vaart, 2004; Miller, Jerebko, Malley, & Summers, 2003) or multi-objective (Pang & Kasabov, 2004; Shi, Suganthan, & Deb, 2004) evolutionary kernel learning.

Space and Time Complexity of the Classifier In some applications, it can be desirable to have fast kernel methods (e.g., for meeting real-time constraints). Thus, the execution time may be considered in the performance assessment during evolutionary kernel learning.

The space and time complexity of SVMs scales with the number of SVs. This is an additional reason to consider minimization of the number of SVs as an objective in evolutionary model selection for SVMs (Igel, 2005; Suttorp & Igel, 2006).

Multi-Objective Optimization The design of a learning machine is usually a MOO problem. For example, accuracy and complexity can be viewed as multiple, and probably conflicting, objectives. The goal of MOO is to approximate a diverse set of Pareto-optimal solutions (i.e., solutions that cannot be improved in one objective without getting worse in another one), which provide insights into the trade-offs between the objectives. Evolutionary multi-objective algorithms have become popular for MOO. Applications of multi-objective evolutionary kernel learning combining some of these performance measures listed above can be found in the work of Igel (2005), Pang and Kasabov (2004), and Shi et al. (2004).

Encoding and Variation Operators

The sheer complexity of the space of possible kernel functions makes it necessary to restrict the search to a particular class of kernel functions. This restriction essentially determines the representation and the operators used in evolutionary kernel learning.

When a parameterized family of mappings is considered, the kernel parameters can be encoded more or less directly in a real-valued EA. This is a frequently used representation, for example for *Gaussian kernel functions*.

For variable selection a binary encoding can be appropriate. One can fix a kernel $k : X \times X \to \mathbb{R}$, where $k(\vec{x}, \vec{z})$ solely depends on some distance measure between $\vec{x}, \vec{z} \in X$. In the binary encoding each bit then indicates whether a particular input variable is considered when computing the distance (Pang and Kasabov, 2004; Shi et al., 2004).

Kernels can be built from other kernels. For example, if k_1 and k_2 are kernel functions on X then $ak_1(\vec{x}, \vec{z}) + bk_2(\vec{x}, \vec{z})$ and $a\exp(-bk_1(\vec{x}, \vec{z}))$ for $\vec{x}, \vec{z} \in X, a, b \in \mathbb{R}^+$ are also kernels on X. This suggests a representation in which the individuals encode expressions that evaluate to kernel functions.

Given these different search spaces, it is not surprising that the aspects of all major branches of evolutionary computation have been used in evolutionary kernel learning: genetic algorithms (Fröhlich et al., 2004), genetic programming (Howley & Madden, 2005), evolution strategies (Igel, 2005), and evolutionary programming (Runarsson & Sigurdsson, 2004).

In general, kernel methods assume that the kernel (or at least the ►Gram matrix ►(kernel matrix) in the training process) is ►positive semidefinite (psd). Therefore, it is advisable to restrict the search space such that only psd functions evolve. Other ways of dealing with the problem of ensuring positive semidefiniteness are to ignore it (Howley & Madden, 2005) or to construct a psd Gram matrix from the matrix $\vec{M}$ induced by the training data and a non-psd "kernel" function. The latter can be achieved by subtracting the smallest eigenvalue of $\vec{M}$ from its diagonal entries.

Gaussian Kernels Gaussian kernel functions are prevalent. Their general form is $k(\vec{x},\vec{z}) := \exp\left(-(\vec{x}-\vec{z})^{\mathrm{T}}\vec{A}(\vec{x}-\vec{z})\right)$ for $\vec{x},\vec{z} \in \mathbb{R}^n$ and symmetric positive definite (pd) matrix $\vec{A} \in \mathbb{R}^{n\times n}$. When adapting $\vec{A}$, the issue of ensuring that the optimization algorithm generates only pd matrices $\vec{A}$ arises. This can be achieved by an appropriate parametrization of $\vec{A}$. Often the search is restricted to matrices of the form $\gamma\vec{I}$, where $\vec{I}$ is the unit matrix and $\gamma \in \mathbb{R}^+$ is the only adjustable parameter. However, allowing more flexibility has proven to be beneficial in certain applications (e.g., see Chapelle et al., 2002; Friedrichs & Igel, 2005; Glasmachers & Igel, 2005). It is straightforward to consider diagonal matrices with positive elements to allow for independent scaling factors weighting the input components. However, only by dropping this restriction one can achieve invariance against both rotation and scaling of the input space. A real-valued encoding that maps onto the set of all symmetric pd matrices can be used such that all modifications of the parameters result in feasible kernels, see the articles by Friedrichs and Igel (2005), Glasmachers and Igel (2005), and Suttorp and Igel (2006) for different parametrizations.

Optimizing Additional Hyperparameters One of the advantages of evolutionary kernel learning is that it can be easily augmented with an optimization of additional hyperparameters of the kernel method. The most prominent example is to encode not only the kernel but also the regularization parameter when doing model selection for SVMs.

Application Example

Notable applications of evolutionary kernel learning include the design of classifiers in bioinformatics (Mersch, Glasmachers, Meinicke, & Igel, 2007; Pang & Kasabov, 2004; Shi et al., 2004). Let us consider the work by Mersch et al. (2007) as an instructive example. Here, the parameters of a sequence kernel are evolved to improve the prediction of gene starts in DNA sequences. The kernel can be viewed as a weighted sum of 64 kernels, each measuring similarity with respect to a particular tri-nucleotide sequence (codon). The 64 weights $w_1,\ldots,w_{64}$ are optimized together with an additional global kernel parameter σ and a regularization parameter C for the SVM. Each individual stores $\vec{x} \in \mathbb{R}^{66}$, where $(w_1,\ldots,w_{64},\sigma,C)^{\mathrm{T}} = (\exp(x_1),\ldots,\exp(x_{64}),|x_{65}|,|x_{66}|)^{\mathrm{T}}$. An evolution strategy is applied, using additive multi-variate Gaussian mutation and weighted global recombination for variation and rank-based selection. The fitness is determined by a 5-fold cross-validation. The evolved kernels lead to higher classification rates and the adapted weights reveal the importance of particular codons for the task at hand.

Cross References

►Neuroevolution

Recommended Reading

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning, 46*(1), 131–159.

Eads, D. R., Hill, D., Davis, S., Perkins, S. J., Ma, J., Porter, R. B., et al. (2002). Genetic algorithms and support vector machines for time series classification. In B. Bosacchi, D. B. Fogel, & J. C. Bezdek (Eds.), *Applications and science of neural networks, fuzzy systems, and evolutionary computation V*, Proceedings of the SPIE (Vol. 4787) (pp. 74–85). SPIE–The International Society for Optical Engineering. Bellington, WA

Friedrichs, F., & Igel, C. (2005). Evolutionary tuning of multiple SVM parameters. *Neurocomputing, 64*(C), 107–117.

Fröhlich, H., Chapelle, O., & Schölkopf, B. (2004). Feature selection for support vector machines using genetic algorithms. *International Journal on Artificial Intelligence Tools, 13*(4), 791–800.

Glasmachers, T., & Igel, C. (2005). Gradient-based adaptation of general gaussian kernels. *Neural Computation, 17*(10), 2099–2105.

Howley, T., & Madden, M. (2005). The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review, 24*(3), 379–395.

Igel, C. (2005). Multi-objective model selection for support vector machines. In C. A. Coello Coello, E. Zitzler, & A. Hernandez

Aguirre (Eds.), *Proceedings of the third international conference on evolutionary multi-criterion optimization (EMO 2005)*, LNCS (Vol. 3410) (pp. 534–546). Berlin: Springer.

Jong, K., Marchiori, E., & van der Vaart, A. (2004). Analysis of proteomic pattern data for cancer detection. In G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, et al. (Eds.), *Applications of evolutionary computing*, LNCS (Vol. 3005, pp. 41–51). Berlin: Springer.

Mersch, B., Glasmachers, T., Meinicke, P., & Igel, C. (2007). Evolutionary optimization of sequence kernels for detection of bacterial gene starts. *International Journal of Neural Systems, 17*(5), 369–381.

Miller, M. T., Jerebko, A. K., Malley, J. D., & Summers, R. M. (2003). Feature selection for computer-aided polyp detection using genetic algorithms. In A. V. Clough & A. A. Amini (Eds.), *Medical imaging 2003: Physiology and function: Methods, systems, and applications*, Proceedings of the SPIE (Vol. 5031) (pp. 102–110).

Pang, S., & Kasabov, N. (2004). Inductive vs. transductive inference, global vs. local models: SVM, TSVM, and SVMT for gene expression classification problems. In *International joint conference on neural networks (IJCNN 2004)* (Vol. 2, pp. 1197–1202). Washington, DC: IEEE Press.

Runarsson, T. P., & Sigurdsson, S. (2004). Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing – Letters and Reviews, 3*(3), 59–68.

Shi, S. Y. M., Suganthan, P. N., & Deb, K. (2004). Multi-class protein fold recognition using multi-objective evolutionary algorithms. In *IEEE symposium on computational intelligence in bioinformatics and computational biology* (pp. 61–66). Washington, DC: IEEE Press.

Suttorp, T., & Igel, C. (2006). Multi-objective optimization of support vector machines. In Y. Jin (Ed.), *Multi-objective machine learning*. Studies in computational intelligence (Vol. 16, pp. 199–220). Berlin: Springer.

Evolutionary Robotics

Phil Husbands
University of Sussex
Brighton, UK

Synonyms

Embodied evolutionary learning; Evolution of agent behaviors; Evolution of robot control

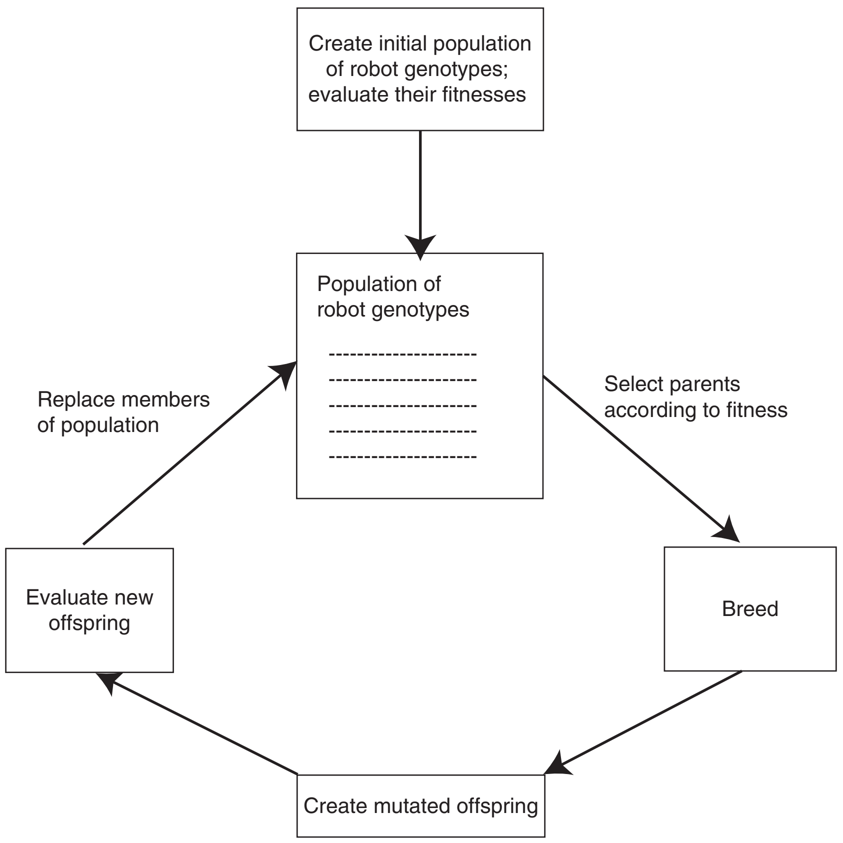
Definition

Evolutionary robotics involves the use of ►evolutionary computing techniques to automatically develop some or all of the following properties of a robot: the control system, the body morphology, and the sensor and motor properties and layout. Populations of artificial genomes (usually lists of characters and numbers) encode properties of autonomous mobile robots required to carry out a particular task or to exhibit some set of behaviors. The genomes are mutated and interbred creating new generations of robots according to a Darwinian scheme in which the fittest individuals are most likely to produce offspring. Fitness is measured in terms of how good a robot's behavior is according to some evaluation criteria; this is usually automatically measured but may, in the manner of eighteenth century pig breeders, be based on the experimenters' judgment.

Motivation and Background

Turing's (1950) paper, *Computing Machinery and Intelligence*, is widely regarded as one of the seminal works in artificial intelligence. It is best known for what came to be called the Turing test – a proposal for deciding whether or not a machine is intelligent. However, tucked away toward the end of Turing's wide ranging discussion of issues arising from the test is a far more interesting proposal. He suggests that worthwhile intelligent machines should be adaptive, should learn and develop, but concedes that designing, building, and programming such machines by hand is probably completely infeasible. He goes on to sketch an alternative way of creating machines based on an artificial analog of biological evolution. Each machine would have hereditary material encoding its structure, mutated copies of which would form offspring machines. A selection mechanism would be used to favor better adapted machines – in this case, those that learned to behave most intelligently. Turing proposed that the selection mechanism should largely consist of the experimenter's judgment.

It was not until more than 40 years after their publication that Turing's long forgotten suggestions became reality. Building on the development of principled evolutionary search algorithm by, among others, Holland (1975), researchers at CNR, Rome, Case Western University, the University of Sussex, EPFL, and elsewhere independently demonstrated methodologies and practical techniques to evolve, rather than design, the control systems for primitive autonomous intelligent machines (Beer & Gallagher, 1992; Cliff, Harvey, & Husbands, 1993; de Garis, 1990; Floreano & Mondada,


Create initial population of robot genotypes; evaluate their fitnesses
Population of robot genotypes
Select parents according to fitness
Breed
Create mutated offspring
Evaluate new offspring
Replace members of population

Evolutionary Robotics. Figure 1. General scheme employed in evolutionary robotics

1994; Husbands & Harvey, 1992; Parisi & Nolfi, 1993). Thus, the field of *Evolutionary Robotics* was born in the early 1990s. Initial motivations were similar to Turing's: the hand design of intelligent adaptive machines intended for operation in natural environments is extremely difficult, would it be possible to wholly or partly automate the process?

Today, the field of evolutionary robotics has expanded in scope to take in a wide range of applications, including promising new work on autonomous flying machines (Floreano, Husbands, & Nolfi, 2008), as well as research aimed at exploring specific scientific issues – for instance, principles from neuroscience or questions in cognitive science (Harvey, Di Paolo, Wood, Quinn, & Tuci, 2005; Philippides, Husbands, Smith, & O'Shea, 2005). Such work is able to exploit the fact that evolutionary robotics operates with fewer assumptions about neural architectures and behavior generating mechanisms than other methods; this means that whole general classes of designs and processes can be explored.

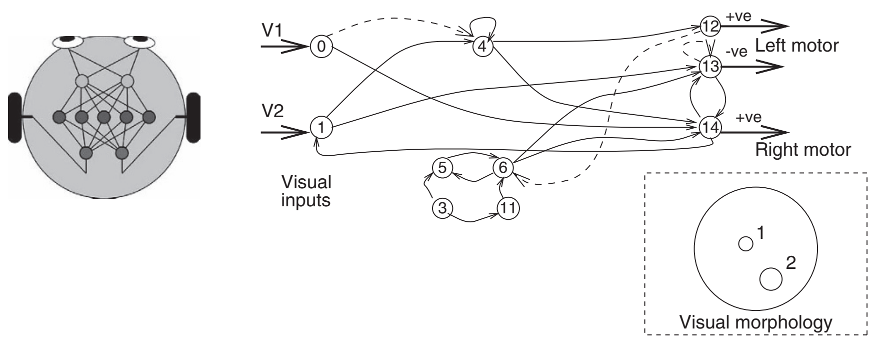
Structure of the Learning System

The key elements of the evolutionary robotics approach are

- An artificial genetic encoding specifying the robot control systems/body plan/sensor properties etc., along with a mapping to the target system
- A method for measuring the fitness of the robot behaviors generated from these genotypes
- A way of applying selection and a set of "genetic" operators to produce the next generation from the current

The structure of the overall evolutionary process is captured in Fig. 1. The general scheme is like that of any application of an evolutionary search algorithm. However, many details of specific parts of the process, particularly the evaluation step, are peculiar to evolutionary robotics.

The more general parts of the evolutionary process (selection, breeding, genetic operators such as mutation and crossover, replacement, and population structure)



Evolutionary Robotics. Figure 2. Evolved neurocontrollers. On the left a simple fixed architecture feedforward network is illustrated. The connection weights, and sometimes the neuron properties, are put under evolutionary control. On the right a more complex architecture is illustrated. In this case, the whole architecture, including the number of neurons and connections, is under evolutionary control, along with connection and neuron properties and the morphology of a visual sensor that feeds into the network

are also found in most other applications of evolutionary computing and, just as in those other applications, there are many well-documented ways of implemented each (De Jong, 2006; Eiben & Smith, 2003). Hence, this section focuses on genetic encoding and evaluation as a route to more evolutionary robotics specific issues. For a much fuller treatment of the subject, see Floreano et al. (2008) and Nolfi and Floreano (2000).

Genetic Encoding

While, as already mentioned, many aspects of the robot design can potentially be under genetic control, *at least* the control system always is. By far the most popular form of controller is some sort of neural network. These range from straightforward feedforward networks of simple elements (Floreano & Mondada, 1994) to relatively complex, dynamic and plastic recurrent networks (Beer & Gallagher 1992; Floreano & Urzelai 2000; Philippides, Husbands, Smith, & O'Shea, 2005), as illustrated in Fig. 2. In the simplest case, a fixed architecture network is used to control a robot whose sensors feed into the network which in turn feeds out to the robot motors. In this scenario, the parameters of the network (connection weights and relevant properties of the units such as thresholds or biases) are coded as a fixed length string of numerical values.

A more complex case, which has been explored since the very early days of evolutionary robotics (Cliff et al., 1993), involves the evolution of the network architecture as well as the properties of the connections and units. Typically, the size of the network (number of units and connections) and its architecture (wiring diagram) are unconstrained and free to evolve. This involves more complex encodings which can grow and shrink, as units and connections are added or lost, while allowing a coherent decoding of connections between units. These range from relatively simple strings employing blocks of symbols that encode a unit's properties and connections relative to other units (Cliff et al.) to more indirect schemes that make use of growth processes in some geometric space (Philippides et al., 2005) or employ genetic programming-like tree representations in which whole subbranches can be added, deleted, or swapped over (Gruau, 1995).

The most general case involves the encoding of control network and body and sensor properties. Various kinds of developmental schemes have been used to encode the construction of body morphologies from basic building blocks, both in simulation and in the real world. The position and properties of sensors can also be put under evolutionary control. Sometimes one complex encoding scheme is used for all aspects of the robot under evolutionary control, and sometimes the different aspects are put on separate genotypes.

Fitness Evaluation

The fitness of members of the population is measured, via an evaluation mechanism, in terms of the robot behaviors produced by the control system, or control system plus robot morphology that it encodes. Fitness evaluation, therefore, consists of translating the genome in question into a robot instantiation and then measuring the aspects of the resulting behavior. In the earliest work aimed at using evolutionary techniques to develop neurocontrollers for particular physical robots, members of a population were downloaded in turn onto the robot and their behavior was monitored and measured either automatically by clever experimental setups (Floreano & Mondada, 1994; Harvey, Husbands, & Cliff, 1994) or manually by an observer (Gruau & Quatramaran, 1997). The machinery of the evolutionary search algorithm was managed on a host computer while the fitness evaluations were undertaken on the target robot.

One drawback of evaluating fitness on the robot is that this cannot be done any quicker than in real time, making the whole evolutionary process rather slow. However, in the early work in the field this approach was taken because it was felt that it was unlikely that simulations could be made accurate enough to allow proper transfer of evolved behavior onto the real robot. However, a careful study of accurate physics-based simulations of a Khepera robot, with various degrees of noise added, proved this assumption false (Jakobi, Husbands, & Harvey, 1995). This led to the development of Jakobi's minimal simulation methodology (Jakobi, 1998a), whereby computationally very efficient simulations are built by modeling only those aspects of the robot–environment interaction deemed important to the desired behavior and masking everything else with carefully structured noise (so that evolution could not come to rely on any of those features). These ultrafast, ultralean simulations have successfully been used with many different forms of robot and sensing, with very accurate transfer of behavior from simulation to reality. An alternative approach uses plastic controllers that further adapt through self-organization to help smooth out the differences between an inaccurate simulation and the real world (Urzelai & Floreano, 2001). Instead of evolving connection weights, in this approach "learning rules" for adapting connection strengths are evolved – this results in controllers that continually adapt to changes in their environment. For details of further approaches, see Floreano et al. (2008). Much evolutionary robotics work now makes use of simulations; without them it would be impossible to do the most ambitious work on the concurrent evolution of controllers and body morphology (Lipson & Pollack, 2000) (to be briefly described later). However, although simulation packages and techniques have developed rapidly in the past few years, there will still inevitably be discrepancies between simulation and reality, and the lessons and insights of the work outlined above should not be forgotten.

An interesting distinction can be made between implicit and explicit fitness functions in evolutionary robotics (Nolfi & Floreano, 2000). In this context, an explicit fitness function rewards specific behavioral elements – such as traveling in a straight line – and hence shapes the overall behavior from a set of specific behavioral primitives. Implicit fitness functions operate at a more indirect, abstract level – fitness points are given for completing some task but they are not tied to specific behavioral elements. Implicit fitness functions might involve components such as maintaining energy levels or covering as much ground as possible, components that can be achieved in many different ways. In practice, it is quite possible to define a fitness function that has both explicit and implicit elements.

Advantages

Potential advantages of this methodology include

- The ability to explore potentially unconstrained designs that have large numbers of free variables. A *class* of robot systems (to be searched) is defined rather than specific, fully defined robot designs. This means fewer assumptions and constraints are necessary in specifying a viable solution.
- The ability to use the methodology to fine-tune the parameters of an already successful design.
- The ability, through the careful design of fitness criteria and selection techniques, to take into account multiple, and potentially conflicting, design criteria and constraints (e.g., efficiency, cost, weight, power consumption, etc.).
- The possibility of developing highly unconventional and minimal designs.
- The ability to explicitly take into account robustness and reliability as major driving force behind the fitness measure, factors that are particularly important for certain applications.

Applications

For a detailed survey of applications of evolutionary robotics, see Floreano et al. (2008); this section gives a brief overview of some areas covered by the methodology to give a better idea of the techniques involved and to indicate the scope of the field.

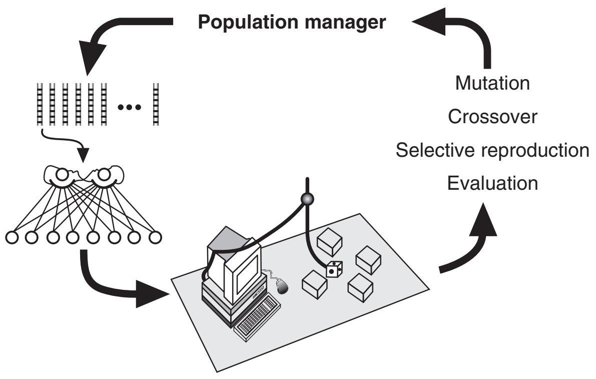
Prominent early centers for research in this area were EPFL and Sussex University, both of which are still very active in the field. Much of the early EPFL work used the miniature Khepera robot (Mondada, Franzi, & Ienne, 1993), which became a popular tool in many areas of robotics research. In its simplest form, it is a two-wheeled cylindrical robot with a ring of IR sensors around its body. The first successful evolutionary robotics experiments at EPFL employed the setup illustrated in Figs. 3 and 4. A population of bit strings encoded the connection weights and node thresholds for a simple fixed architecture feedforward neural network. Each member of the population was decoded into a particular instantiation of a neural network controller which was then downloaded onto the robot (Floreano & Mondada, 1994). This controlled the robot for a fixed period of time as it moved around the environment shown in Fig. 4.

The following simple fitness function was used to evolve obstacle avoidance behaviors:

$$F = V + (1 - \sqrt{DV}) + (1 - I)$$

where V is the average rotation speed of opposing wheels, DV is the difference between signed speed values of opposing wheels, and I is the activation value of the IR sensor with the highest input (readings are high if an obstacle is close to a sensor). Maximizing this function ensures high speed, a tendency to move in straight lines, and avoidance of walls and obstacles in the environment. After about 36 h of real-world evolution using this setup, controllers were evolved that successfully generated efficient motion around the course, avoiding collisions with the walls.

At the same time as this work was going on at EPFL, a series of pioneering experiments on evolving visually guided behaviors were being performed at Sussex University (Cliff et al., 1993; Harvey et al., 1994) in which discrete-time dynamical recurrent neural networks and visual sampling morphologies were concurrently evolved to allow a gantry robot (as well as other more standard mobile robots) to perform various visually guided tasks. An early instantiation of the Sussex gantry robot is shown in Fig. 5.

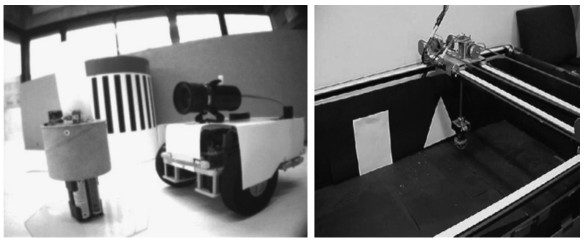


Evolutionary Robotics. Figure 3. **Setup for early EPFL evolutionary robotics experiments with the Khepera robot (see text for details). Used with permission**

Evolutionary Robotics. Figure 4. **The simple environment used for evolving obstacle avoidance behaviors with a Khepera robot. Used with permission**

A CCD camera points down toward a mirror angled at 45°. The mirror can rotate around an axis perpendicular to the camera's image plane. The camera is suspended from the gantry allowing motion in the X, Y, and Z dimensions. This effectively provides an equivalent to a wheeled robot with a forward facing camera when only the X and Y dimensions of translation are used (see Fig. 5).

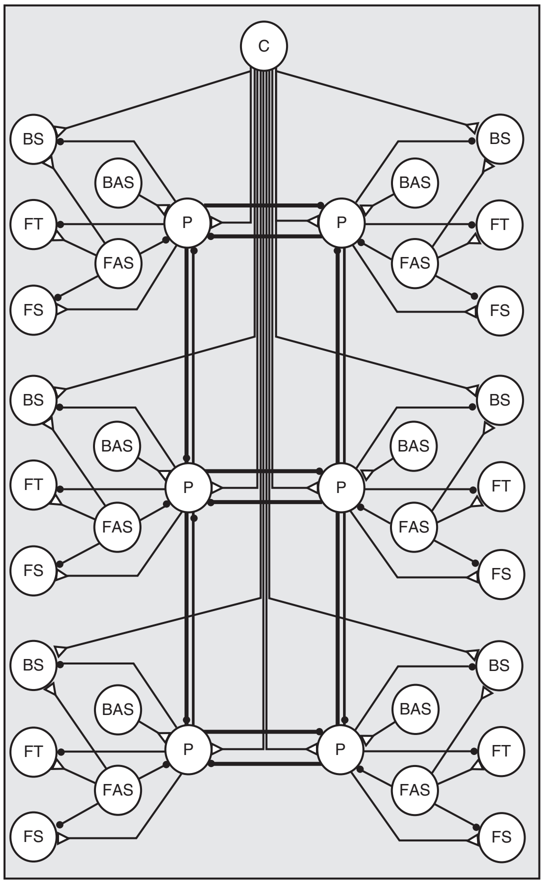
The apparatus was initially used in a manner similar to the real-world EPFL evolutionary robots setup illustrated in Fig. 3. A population of strings encoding robot controllers and visual sensing morphologies are stored on a computer to be downloaded one at a time onto the robot. The exact position and orientation of the camera head can be accurately tracked and used in the fitness evaluations. A number of visually



Evolutionary Robotics. Figure 5. An early version of the Sussex gantry robot (*right*) was a "hardware simulation" of a robot such as that shown on the left. It allowed real-world evolution of visually guided behaviors in an easily controllable experimental setup (see text for further details)

guided navigation behaviors were successfully achieved, including navigating around obstacles and discriminating between different objects. In the experiment illustrated in Fig. 5, starting from a random position and orientation the robot has to move to the triangle rather than the rectangle. This has to be achieved irrespective of the relative positions of the shapes and under very noisy lighting conditions. The architecture and all parameters of recurrent neural network controllers were evolved in conjunction with visual sampling morphologies – only genetically specified patches from the camera image were used (by being fed to input neurons according to a genetic specification), the rest of the image is thrown away. This resulted in extremely minimal systems only using 2 or 3 pixels of visual information, yet still able to very robustly perform the task under highly variable lighting conditions. Behaviors were evolved in an incremental way, with more complex capabilities being evolved from populations of robots that were successful at some simpler task (for details see Harvey et al. (1994) and Harvey, Husbands, Cliff, Thompson, & Jakobi (1997)). The highly minimal yet very robust systems developed highlighted the potential for evolutionary robotics techniques in areas such as space exploration where there is a great pressure to minimize resources while maintaining reliability (Hobbs, Husbands, & Harvey, 1996).

Since this early work, many different behaviors have been successfully evolved on a wide range of robots (Floreano et al., 2008; Nolfi & Floreano, 2000) There is not enough room to give an adequate summary of the



Evolutionary Robotics. Figure 6. Schematic diagram of a distributed neural network for the control of locomotion as used by Beer et al. Excitatory connections are denoted by open triangles, and inhibitory connections are denoted by filled circles. C, command neuron; P, pacemaker neuron; FT, foot motor neuron; FS and BS, forward swing and backward swing motor neurons; FAS and BAS, forward and backward angle sensors. Reproduced with permission

whole field, so a few interesting subareas are highlighted below.

Over the past 15 years or so, there has been a growing body of work on evolving controllers for various kinds of walking robots – a nontrivial sensorimotor coordination task. Early work in this area concentrated on evolving dynamical network controllers for simple simulated insects (often inspired by cockroach studies), which were required to walk in uncomplicated environments (e.g., de Garis, 1990; Beer & Gallagher, 1992).

The promise of this work soon led to versions of this methodology being used on real robots. Probably, the first success in this direction was by Lewis, Fagg, and Solidum (1992) who evolved a neural controller for a simple hexapod robot, using coupled oscillators built from continuous-time, leaky-integrator, artificial neurons. The robot was able to execute an efficient tripod gait on flat surfaces. All evaluations were done on the actual robot with each leg connected to its own pair of coupled neurons, leg swing being driven by one neuron and leg elevation by the other. These pairs of neurons were cross-connected, in a manner similar to that used in the neural architecture shown in Fig. 6, to allow coordination between the legs. This architecture for locomotion, introduced by Beer, Chiel, and Sterling (1989), was based on the studies of cockroaches and has been much used ever since. Gallagher, Beer, Espenschiel, and Quinn (1996) used a generalization of it to evolve controllers for generating locomotion in a hexapod robot. This machine was more complex than Lewis et al.'s, with a greater number of degrees of freedom per leg. In this work, each leg was controlled by a fully connected network of five continuous-time, leaky-integrator neurons, each receiving a weighted sensory input from that leg's angle sensor. The connection weights and neuron time constants and biases were under genetic control. This produced efficient tripod gaits for walking on flat surfaces. In order to produce a wider range of gaits operating at a number of speeds such that rougher terrain could be successfully negotiated, a slightly different distributed architecture, more inspired by stick insect studies, was found to be more effective (Beer, Quinn, Chiel, & Ritzmann, 1997).

Jakobi (1998b) successfully used his minimal simulation techniques to evolve controllers for an 8-legged robot. Evolution in simulation took less than 2 h on what would today be regarded as a very slow computer, and then transferred successfully to the real robot. Jakobi evolved modular controllers based on Beer's continuous recurrent network architecture to control the robot as it engaged in walking about its environment, avoiding obstacles and seeking out goals. The robot could smoothly change gait, move backward and forward, and even turn on the spot. More recently, related approaches have been successfully used to evolve controllers for more mechanically sophisticated robots such as the Sony Aibo (Tllez, Angulo, & Pardo, 2006). In the last few years, there has also been successful work on evolving coupled oscillator style neural controllers for the highly unstable dynamic problem of biped walking. Reil and Husbands (2002) showed that accurate physics-based simulations using physics-engine software could be used to develop controllers able to generate successful bipedal gaits. Reil and colleagues have now significantly developed this technology to exploits its commercial possibilities in the animation and games industries (see www.naturalmotion.com for further details). Vaughan has taken related work in another direction. He has successfully applied evolutionary robotics techniques to evolve a simulation of a 3D ten-degree of freedom bipedal robot. This machine demonstrates many of the properties of human locomotion. By using passive dynamics and compliant tendons, it conserves energy while walking on a flat surface. Its speed and gait can be dynamically adjusted and it is capable of adapting to discrepancies in both its environment and its body's construction (Vaughan, Di Paolo & Harvey, 2004). In general, the evolutionary development of neural network walking controllers, with their intricate dynamics, produces a wider range of gaits and generates smoother, more adaptive locomotion than the more standard use of finite state machine based systems employing parameterized rules governing the timing and coordination of individual leg movements.

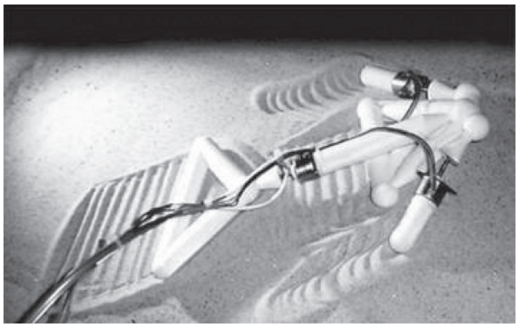
Early single robot research was soon expanded to handle interactions between multiple robots. Floreano and Nolfi did pioneering work on the coevolution of predator–prey behaviors in physical robots (Floreano & Nolfi, 1997). The fitness of the prey robot was measured by how quickly it could catch the prey; the fitness of the prey was determined by how long it could escape the predator. Two Khepera robots were used in this experiment, each had the standard set of proximity sensors but the predator also has a vision system and the prey was able to move twice as fast as the predator. A series of interesting chasing and evasion strategies emerged. Later Quinn, Smith, Mayley, and Husbands (2003) demonstrated the evolution of coordinated cooperative behavior in a group of robots. A group of robots equipped only with IR proximity sensors were required to move as far as possible as a coordinated group starting from a random configuration. The task was solved by the robots adopting and then maintaining a specific formation.

Analysis of the best evolved solution showed that it involved the robots adopting different roles, with the identical robots collectively "deciding" which robot would perform each role. Given the minimal sensing constraints, the evolved system would have proved extremely difficult to have designed by hand. For discussion of other multiple robot behaviors, see Floreano et al. (2008).

In the work described so far, control systems have been evolved for pre-existing robots: the brain is constrained to fit a particular body and set of sensors. Of course in nature, the nervous system evolved simultaneously with the rest of the organism. As a result, the nervous system is highly integrated with the sensory apparatus and the rest of the body: the whole operates in a harmonious and balanced way – there are no distinct boundaries between the control system, the sensors, and the body.

Karl Sims started to explore the concurrent evolution of the brain and the body in his highly imaginative work involving simulated 3D "creatures" (Sims, 1994). In this work, the creatures coevolved under a competitive scenario in which they were required to try and gain control of a resource (a cube) placed in the centre of an arena. Both the morphology of the creatures and the neural system controlling their actuators were under evolutionary control.

Lipson and Pollack (2000), working at Brandeis University, pushed the idea of fully evolvable robot hardware about as far as is reasonably technologically feasible at present. In an important piece of research, directly inspired by Sims' earlier simulation work, autonomous "creatures" were evolved in simulation out of basic building blocks (neurons, plastic bars, and actuators). The bars could connect together to form arbitrary truss structures with the possibility of both rigid and articulated substructures. Neurons could be connected to each other and to the bars whose length they would then control via a linear actuator. Machines defined in this way were required to move as far as possible in a limited time. The fittest individuals were then fabricated robotically using rapid manufacturing technology (plastic extrusion 3D printing) to produce results such as that shown in Fig. 7. They thus achieved autonomy of design and construction using evolution in a "limited universe" physical simulation coupled to automatic fabrication. The highly unconventional designs thus realized performed as well in reality as in simulation. The success of this work points the way to new possibilities in developing energy efficient fault tolerant machines.


Evolutionary Robotics. Figure 7. **A fully automatically evolved robot developed on the Golem project (see text for details). Used with permission**

Pfeifer and colleagues at Zurich University have explored issues central to the key motivation for fully evolvable robot hardware: the balanced interplay between body morphology, neural processing, and generation of adaptive behavior and have developed a set of design principles for intelligent systems in which these issues take centre stage (Pfeifer & Bongard, 2007).

Future Directions

Major ongoing challenges – methodological, theoretical, and technological – include finding the best way to incorporate development and lifetime plasticity within the evolutionary framework (this involves trends coming from the emerging field of epigenetic robotics), understanding better what the most useful building blocks are for evolved neurocontrollers, and finding efficient ways to scale work on concurrently evolving bodies and brains.

There are very interesting developments in the evolution of group behaviors and the emergence of communication (Di Paolo, 1998; Floreano, Mitri, Magnenat, & Keller, 2007; Quinn, 2001), the use of evolutionary robotics as a tool to illuminate problems in cognitive science (Beer, 2003; Harvey et al., 2005) and neuroscience (Di Paolo, 2003; Philippides et al., 2005;

Seth, 2005), in developing flying behaviors (Floreano, Hauert, Leven, & Zufferey, 2007; Shim & Husbands, 2007), and in robots that have some form of self-model (Bongard, Zykov, & Lipson, 2006), to name but a few.

Cross References

►Co-Evolutionary Learning
►Evolutionary Artificial Neural Networks
►Genetic Algorithms
►Robot Learning

Recommended Reading

Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent (with commentary and response). *Adaptive Behavior*, *11*(4), 209–243.

Beer, R. D., Chiel, H. J., & Sterling, L. S. (1989). Heterogeneous neural networks for adaptive behavior in dynamic environments. In D. Touretzky (Ed.), *Neural information processing systems* (vol.1, pp. 577–585). San Francisco, CA: Morgan Kauffman.

Beer, R. D., & Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behaviour. *Adaptive Behaviour*, *1*, 94–110.

Beer, R. D., Quinn, R. D., Chiel, H. J., & Ritzmann, R. E. (1997). Biologically-inspired approaches to robotics. *Communications of the ACM*, *40*(3), 30–38.

Bongard, J., Zykov, V., & Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, *314*, 1118–1121.

Cliff, D., Harvey, I., & Husbands, P. (1993). Explorations in evolutionary robotics. *Adaptive Behavior*, *2*, 73–110.

de Garis, H. (1990). Genetic programming: Evolution of time dependent neural network modules which teach a pair of stick legs to walk. In *Proceedings of the 9th European conference on artificial intelligence* (pp. 204–206). Stockholm, Sweden.

De Jong, K. A. (2006). *Evolutionary computation: A unified approach.* Cambridge, MA: MIT Press.

Di Paolo, E. (1998). An investigation into the evolution of communication. *Adaptive Behavior, 6*(2), 285–324.

Di Paolo, E. A. (2003). Evolving spike-timing dependent plasticity for single-trial learning in robots. *Philosophical Transactions of the Royal Society A*, *361*, 2299–2319.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Berlin: Springer.

Floreano, D., Hauert, S., Leven, S., & Zufferey, J. C. (2007). Evolutionary swarms of flying robots. In D. Floreano (Ed.), *Proceedings of the international symposium on flying insects and robots*, (pp. 35–36). Monte Verita, Switzerland: EPFL.

Floreano, D., Husbands, P., & Nolfi, S. (2008). Evolutionary robotics. In B. Siciliano, & O. Khatib (Eds.), *Springer handbook of robotics* (Chap. 61). (pp.1423–1451). Berlin: Springer.

Floreano, D., Mitri, S., Magnenat, S., & Keller, L. (2007). Evolutionary conditions for the emergence of communication in robots. *Current Biology*, *17*, 514–519.

Floreano, D., & Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J. Meyer, & S. W. Wilson (Eds.), *From animals to animats III: Proceedings of the third international conference on simulation of adaptive behavior* (pp. 402–410). Cambridge, MA: MIT Press-Bradford Books.

Floreano, D., & Nolfi, S. (1997). Adaptive behavior in competing co-evolving species. In P. Husbands, & I. Harvey (Eds.), *Proceedings of the 4th European conference on artificial life* (pp. 378–387). Cambridge, MA: MIT Press.

Floreano, D., & Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, *13*(4–5), 431–443.

Gallagher, J., Beer, R., Espenschiel, M., & Quinn, R. (1996). Application of evolved locomotion controllers to a hexapod robot. *Robotics and Autonomous Systems*, *19*(1), 95–103.

Gruau, F. (1995). Automatic definition of modular neural networks. *Adaptive Behavior*, *3*(2), 151–183.

Gruau, F., & Quatramaran, K. (1997). Cellular encoding for interactive evolutionary robotics. In P. Husbands, & I. Harvey (Eds.), *Proceedings of the 4th European conference on artificial life.* Cambridge, MA: The MIT Press/Bradford Books

Harvey, I., Di Paolo, E., Wood, R., Quinn, M., & Tuci, E. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, *11*(1–2), 79–98.

Harvey, I., Husbands, P., & Cliff, D. T. (1994). Seeing the light: Artificial evolution, real vision. In D. T. Cliff, P. Husbands, J. A. Meyer, & S. Wilson (Eds.), *From animals to animats 3: Proceedings of the third international conference on simulation of adaptive behaviour, SAB94* (pp. 392–401). Cambridge, MA: MIT Press.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., & Jakobi, N. (1997). Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, *20*, 205–224.

Hobbs, J., Husbands, P., & Harvey, I. (1996). Achieving improved mission robustness. In *4th European Space Agency workshop on advanced space technologies for robot applications – ASTRA'96*, Noordwijk, The Netherlands ESTEC.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Husbands, P., & Harvey, I. (1992). Evolution versus design: Controlling autonomous mobile robots. In *Proceedings of 3rd annual conf. on artificial intelligence, simulation and planning in high autonomy systems* (pp. 139–146) Los Alimitos, CA: IEEE Computer Society Press.

Jakobi, N. (1998a). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behaviour*, *6*, 325–368.

Jakobi, N. (1998b). Running across the reality gap: Octopod locomotion evolved in a minimal simulation. In P. Husbands, & J. A. Meyer, (Eds.), *Evolutionary robotics: First European workshop, EvoRobot98* (pp. 39–58). Berlin: Springer.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulations in evolutionary robotics. In F. Moran et al. (Eds.), *Proceedings of 3rd European conference on artificial life* (pp. 704–720). Berlin: Springer.

Lewis, M. A., Fagg, A. H., & Solidum, A. (1992). Genetic programming approach to the construction of a neural network for a walking robot. In *Proceedings of IEEE international conference on robotics and automation* (pp. 2618–2623). Washington, DC: IEEE Press.

Lipson, H., & Pollack, J. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, *406*, 974–978.
Mondada, F., Franzi, E., & Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In T. Yoshikawa, & F. Miyazaki (Eds.), *Proceedings of the third international symposium on experimental robotics* (pp. 501–513). Berlin: Springer.
Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: The biology, Intelligence, and technology of self-organizing machines*. Cambridge, MA: MIT Press/Bradford Books.
Parisi, D., & Nolfi, S. (1993). Neural network learning in an ecological and evolutionary context. In V. Roberto (Ed.), *Intelligent perceptual systems* (pp. 20–40). Berlin: Springer.
Pfeifer, R., & Bongard, J. (2007). *How the body shapes the way we think: A new view of intelligence*. Cambridge, MA: MIT Press.
Philippides, A., Husbands, P., Smith, T., & O'Shea, M. (2005). Flexible couplings: Diffusing neuromodulators and adaptive robotics. *Artificial Life*, *11*(1&2), 139–160.
Quinn, M. (2001). Evolving communication without dedicated communication channels. In J. Kelemen, & P. Sosik. (Eds.), *Proceedings of the 6th European conference on artificial life*, ECAL'01 (pp. 357–366). Berlin: Springer.
Quinn, M., Smith, L., Mayley, G., & Husbands, P. (2003). Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London, Series A: Mathematical, Physical and Engineering Sciences*, *361*, 2321–2344.
Reil, T., & Husbands, P. (2002). Evolution of central pattern generators for bipedal walking in real-time physics environments. *IEEE Transactions of Evolutionary Computation*, *6*(2), 10–21.
Seth, A. K. (2005). Causal connectivity analysis of evolved neural networks during behavior. *Network: Computation in Neural Systems*, *16*(1), 35–54.
Shim, Y. S., & Husbands, P. (2007). Feathered flyer: Integrating morphological computation and sensory reflexes into a physically simulated flapping-wing robot for robust flight Manoeuvre. In *Proceedings of ECAL* LNCS (Vol. 4648, pp. 756–765). Berlin: Springer.
Sims, K. (1994). Evolving 3D morphology and behavior by competition. In R. Brooks, & P. Maes, (Eds.), *Proceedings of artificial life IV* (pp. 28–39). Cambridge, MA: MIT Press.
Tllez, R., Angulo, C., & Pardo, D. (2006). Evolving the walking behaviour of a 12 DOF quadruped using a distributed neural architecture. In *2nd International workshop on biologically inspired approaches to advanced information technology (Bio-ADIT'2006)* LNCS (Vol. 385, pp. 5–19). Berlin: Springer.
Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.
Urzelai, J., & Floreano, D. (2001). Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments. *Evolution Computing*, *9*, 495–524.
Vaughan, E., Di Paolo, E. A., & Harvey, I. (2004). The evolution of control and adaptation in a 3D powered passive dynamic walker. In J. Pollack, M. Bedau, P. Husbands, T. Ikegami, & R. Watson, (Eds.), *Proceedings of the ninth international conference on the simulation and synthesis of living systems artificial life IX*, (pp. 139–145). Cambridge, MA: MIT Press.

Evolving Neural Networks

►Neuroevolution

Example

►Instance

Example-Based Programming

►Inductive Programming

Expectation Maximization Algorithm

►Expectation-Maximization Algorithm

Expectation Maximization Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Synonyms

EM Clustering

The EM algorithm (Dempster, Laird, & Rubin 1977) finds maximum likelihood estimates of parameters in probabilistic models. EM is an iterative method which alternates between two steps, expectation (E) and maximization (M). For clustering, EM makes use of the finite Gaussian mixtures model and estimates a set of parameters iteratively until a desired convergence value is achieved. The mixture is defined as a set of K probability distributions and each distribution corresponds to one cluster. An instance is assigned with a membership probability for each cluster.

The EM algorithm for partitional clustering works as follows:

1. Guess initial parameters: mean and standard deviation (if using normal distribution model).

2. Iteratively refine the parameters with *E* and *M* steps. In the *E* step: compute the membership possibility for each instance based on the initial parameter values. In the *M* step: recompute the parameters based on the new membership possibilities.
3. Assign each instance to the cluster with which it has the highest membership possibility.

Cross References

►Expectation-Maximization Algorithm

Recommended Reading

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 39*(1), 1–38.

Expectation Propagation

Tom Heskes
Radboud University Nijmegen
Nijmegen, The Netherlands

Synonyms

EP

Definition

Expectation propagation is an algorithm for Bayesian machine learning (see ►Bayesian Methods). It tunes the parameters of a simpler approximate distribution (e.g., a Gaussian) to match the exact posterior distribution of the model parameters given the data. Expectation propagation operates by propagating messages, similar to the messages in (loopy) belief propagation (see ►Graphical Models). Whereas messages in belief propagation correspond to exact belief states, messages in expectation propagation correspond to approximations of the belief states in terms of expectations, such as means and variances. It is a deterministic method especially well-suited to large databases and dynamic systems, where exact methods for Bayesian inference fail and ►Monte Carlo methods are far too slow.

Motivation and Background

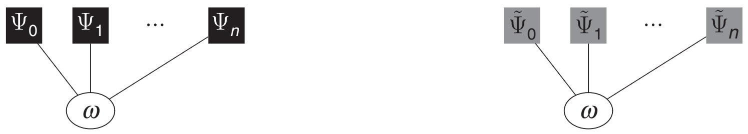
One of the main problems for ►Bayesian methods are their computational expense: computation of the exact posterior, given the observed data, typically requires the solution of high-dimensional integrals that have no analytical expressions. Approximation algorithms are needed to approximate this posterior as accurately as possible. These techniques for approximate inference can be subdivided in two categories: deterministic approaches and stochastic sampling (Monte Carlo) methods. Having the important advantage that (under certain conditions) they give exact results in the limit of an infinite number of samples, *Monte Carlo methods* are the method of choice in Bayesian statistics. However, in particular when dealing with large databases, the time needed for stochastic sampling to obtain a reasonably accurate approximation of the exact posterior can be prohibitive. This explains the need for faster, deterministic approaches, such as the Laplace approximation, *variational approximations*, and expectation propagation.

Expectation propagation was first described by Thomas Minka in his thesis (Minka, 2001). It can be viewed as a generalization and reformulation of the earlier ADATAP algorithm of Manfred Opper and Ole Winther (2001). Expectation propagation quickly became one of the most popular deterministic approaches for approximate Bayesian inference. Expectation propagation improves upon assumed density filtering, a classical method from stochastic control, by iteratively refining local approximations instead of computing them just once. Furthermore, it encompasses loopy belief propagation, a popular method for approximate inference in probabilistic ►graphical models, as a special case. Where loopy belief propagation is restricted to models of discrete variables only, expectation propagation applies to a much wider class of probabilistic graphical models with discrete and continuous variables and complex interactions between them.

Structure of Learning System

Bayesian Machine Learning

In the Bayesian framework for machine learning, you should enumerate all reasonable models of the data and assign a prior belief $P(w)$ to each of these models w. In the discrete case, the w are the different models, in the continuous case, the w are the continuous valued parameters (usually vectors). Then, upon observing the data D, you compute the likelihood $P(D|w)$ to evaluate how probable the data was under each of these models.



Expectation Propagation. Figure 1. (left-hand side) A so-called factor graph corresponding to the i.i.d. assumption in Bayesian machine learning. Each box corresponds to a factor or term. A circle corresponds to a variable. Factors are connected to the variables that they contain. Ψ_0 corresponds to the prior, $\Psi_1 \ldots \Psi_n$ are the likelihood terms for the n data points. (right-hand side) Factor graph of the approximating distribution. The original terms have been replaced by term approximations

The product of the prior and the likelihood gives you, up to a normalization constant, the posterior probability $P(w|D)$ over models given the data:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)},$$

where the normalization term $P(D)$ is called the probability of the data or "evidence." This posterior probability incorporates all you have learned from the data D regarding the models w under consideration. As indicated above, exact calculation of this posterior probability is often infeasible, because the normalization term requires the solution of intractable sums or integrals.

In its simplest setting, the data D consists of n observations, $x_1, \ldots, x_n$, which are assumed to be independent and identically-distributed (i.i.d.). The posterior probability then factorizes into $n + 1$ terms, one for each observation and one for the prior. With definitions $\Psi_0(w) \equiv P(w)$ and $\Psi_i(w) \equiv P(x_i|w)$, we can rewrite

$$P(w|D) = \frac{P(w)\prod_{i=1}^{n} P(x_i|w)}{P(D)} \equiv \frac{\prod_{i=0}^{n} \Psi_i(w)}{P(D)}.$$

This factorization is visualized in the so-called factor graph in Fig. 1. We use it as a running example in the following section.

Assumed Density Filtering

Expectation propagation can be interpreted as an iterative refinement of assumed density filtering. In assumed density filtering, we add terms one-by-one and project in each step back to the "assumed density." For example, suppose that our prior probability $P(w) = \Psi_0(w)$ is a (known) Gaussian distribution over model parameters w, the terms corresponding to the data points are non-Gaussian, and we aim to find an appropriate Gaussian approximation $Q(w)$ to the exact (non-Gaussian) posterior $P(w|D)$. Our first approximation is the prior itself. Assumed-density filtering now proceeds by adding terms one at a time, where at each step we approximate the resulting distribution as closely as possible by a Gaussian. The pseudo-code is given in Algorithm 1, where $Q_{0:i}(w)$ denotes the approximation obtained after incorporating the prior and the first i observations.

If we use the Kullback–Leibler divergence as the distance measure from the non-Gaussian (but normalized) product of $Q_{0:i-1}(w)$ and $\Psi_i(w)$ and the Gaussian approximation, projection becomes "moment matching"; the result of the projection is the Gaussian that has the same mean and covariance matrix as the non-Gaussian product.

Expectation Propagation

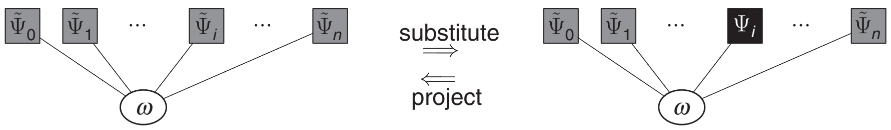
When in assumed density filtering, we add the term $\Psi_i(w)$, the Gaussian approximation changes from $Q_{0:i-1}(w)$ to $Q_{0:i}(w)$. We will call the quotient of the two the *term approximation* (here and in the following we ignore normalization constants):

$$\tilde{\Psi}_i(w) = \frac{Q_{0:i}(w)}{Q_{0:i-1}(w)}.$$

In our running example, term approximations are quotients between two different Gaussian densities and therefore have a Gaussian form themselves. Since the prior $\Psi_0(w)$ is a Gaussian density, $\tilde{\Psi}_0(w) = \Psi_0(w)$. The approximation $Q_{0:n}(w)$ is equal to the product of all

Algorithm 1 Assumed density filtering

```
1: $Q_0(w) = \Psi_0(w)$
2: for $i = 1$ to $n$ do
3:   $Q_{0:i}(w) = \text{Project\_to\_Gaussian}(Q_{0:i-1}(w)\Psi_i(w))$
4: end for
```



Expectation Propagation. Figure 2. Visualization of expectation propagation when recomputing the term approximation for observation *i*

term approximations and is visualized on the righthand side of Fig. 1. In assumed density filtering, the resulting approximation depends on the ordering in which the terms have been added. For example, if the terms had been added in reverse order, their term approximations might have been (slightly) different.

Expectation propagation now generalizes assumed density filtering by iteratively refining these term approximations. When successful, the final approximation will be independent of the ordering. Pseudo-code of expectation propagation is given in Algorithm 2. In step 1 through 5, the term approximations are initialized; in step 6 through 12, these term approximations are iteratively refined until they no longer change. In step 8, we take out the previous term approximation from the current approximation. In step 9, we put back in the exact term and project back to a Gaussian, like we did in assumed density filtering. It is easy to check that the approximation $Q(w)$ after the first loop equals the approximation $Q_{0:n}(w)$ obtained with assumed density filtering. The recalculation of the term approximation corresponding to observation i is visualized in Fig. 2.

Computational Aspects

With expectation propagation, we have to do a little more bookkeeping than with assumed density filtering: we have to keep track of the term approximations. One loop of expectation propagation is about as expensive as running assumed density filtering. Typically, about five iterations are sufficient for convergence.

The crucial operation is in step 3 of Algorithm 1 and step 9 of Algorithm 2. Here we have to compute the moments of the (non-Gaussian) probability distribution on the right-hand side. In most cases, we do not have analytical expressions for these moments and have to compute them numerically, e.g., using Gaussian quadrature. We then obtain the moments (mean and covariance matrix) of the new approximation $Q(w)$. Divisions and multiplications correspond to a simple subtraction and addition of so-called canonical parameters. For the Gaussian, these canonical parameters are the inverse of the covariance matrix (precision matrix) and the product of the precision matrix and the mean. The bottom-line is that we go back and forth between distributions in terms of moments and in terms of canonical parameters. For a Gaussian, this requires computing the inverse of the covariance matrix, which is roughly on the order of d^3, where d is the dimension of w. A practical point of concern is that matrix inversion is numerically instable, in particular for matrices that are close to singular, which can lead to serious round-off errors.

Convergence Issues

Sadly enough, expectation propagation is not guaranteed to converge to a fixed point. If it does, this fixed point can be shown to correspond to an extremum of the so-called Bethe free energy, an approximation of the "evidence" $\log P(D)$, under particular consistency and normalization constraints (Heskes, Opper, Wiegerinck,

Algorithm 2 Expectation propagation

```
 1: $\tilde{\Psi}_0(w) = \Psi_0(w)$
 2: for $i = 1$ to $n$ do
 3:     $\tilde{\Psi}_i(w) = 1$
 4: end for
 5: $Q(w) = \prod_{i=0}^{n} \tilde{\Psi}_i(w)$
 6: while not converged do
 7:     for $i = 1$ to $n$ do
 8:         $Q_{-i}(w) = \frac{Q(w)}{\tilde{\Psi}_i(w)}$
 9:         $Q(w) = \text{Project\_to\_Gaussian}(Q_{-i}(w)\Psi_i(w))$
10:         $\tilde{\Psi}_i(w) = \frac{Q(w)}{Q_{-i}(w)}$
11:     end for
12: end while
```

Winther, & Zoeter, 2005; Heskes & Zoeter, 2002; Minka, 2001, 2005). These constraints relate to the projection step in Algorithm 2: after convergence, the moments of $Q(w)$ should be equal to the moments of the distribution obtained by taking out a term approximation and putting back the corresponding exact term. This should hold for all i.i.d. observations $i = 1, \ldots, n$ in the factor graph of Fig. 1: so we conclude that, after convergence, the moments ("expectations") of all distributions constructed in this way should be the same. Expectation consistent approximations are based on the exact same idea and indeed turn out to be equivalent to expectation propagation (Heskes et al., 2005).

When expectation propagation does not converge, we can try "damping": instead of replacing the old term approximation by the new one, we replace it by a log-convex combination of the old and the new one. In many cases, damping with a step size 0.1 makes expectation propagation converge, at the expense of requiring more iterations. However, even damping with an infinitesimally small step size is not guaranteed to lead to convergence. In those cases, we can try to minimize the Bethe free energy more explicitly with a so-called double-loop algorithm (Heskes & Zoeter, 2002): in the outer loop we compute a convex bound on the Bethe free energy, which we then minimize in the inner loop with an algorithm very similar to standard expectation propagation. Double-loop algorithms are an order of magnitude slower than standard expectation propagation.

Generalizations

The running example above serves to illustrate the main idea, but is of course rather restrictive. Expectation propagation can be applied with any member of the exponential family as approximating distribution (Minka, 2001; Seeger, 2005). The crucial operations are the projection step and the transformation from moment to canonical form: if these can be performed efficiently and robustly, expectation propagation is into play.

In many interesting cases, the model to be learned (here represented as a single variable w) contains a lot of structure. This structure can be exploited by expectation propagation to make it more efficient. For example, when a term only contains a subset of the elements of w, so does its term approximation. Also, we might take as the approximating distribution a distribution that factorizes over the elements of w, instead of a "full" distribution coupling all elements. For a Gaussian, this would amount to a diagonal instead of a full covariance matrix. Such a factorization will lead to lower memory requirements and faster computation, perhaps at the expense of reduced accuracy. More advanced approximations include Tree-EP, where the approximating structure is a tree, and generalized expectation propagation, which generalizes expectation propagation to include higher-order interactions in the same way as generalized belief propagation generalizes loopy belief propagation (Welling, Minka, & Teh, 2005).

Power expectation propagation (Minka, 2005) generalizes expectation propagation by considering a different distance measure in the projection step. Instead of taking the Kullback–Leibler divergence, we can take any so-called α-divergence. $\alpha = 1$ corresponds to the Kullback–Leibler divergence, $\alpha = -1$ to the Kullback–Leibler divergence with the two probabilities interchanged. In the latter case, we obtain a variational method called variational Bayes.

Programs and Data

Code for expectation propagation applied for Gaussian process classification can be found at http://www.kyb.tuebingen.mpg.de/bs/people/csatol/ogp/, and http://www.gaussianprocess.org/gpml/code/matlab/doc/classification.html. Kevin Murphy's Bayes Net toolbox (http://bnt.sourceforge.net) can provide a good starting point to write your own code for expectation propagation.

Applications

Expectation propagation has been applied for, among others, Gaussian process classification (Csató, 2002), inference in Bayesian networks and Markov random fields, text classification with Dirichlet models and processes (Minka & Lafferty, 2002), ▶logistic regression models for rating players (Herbrich & Graepel, 2006), and inference and learning in hybrid and nonlinear dynamic Bayesian networks (Heskes & Zoeter, 2002).

Future Directions

From an application point of view, expectation propagation will probably become one of the standard techniques for approximate Bayesian machine learning, much like the Laplace approximation and Monte Carlo methods. Future research may involve questions like

- When does expectation propagation converge? Can we design variants that are guaranteed to converge?
- What "power" to use in power expectation propagation for what kind of purposes?
- Can we adapt expectation propagation to handle approximating distributions that are not part of the exponential family?

Cross References

►Gaussian Distribution
►Gaussian Process
►Graphical Models

Recommended Reading

Csató, L. (2002). *Gaussian processes – iterative sparse approximations*. PhD thesis, Aston University, Birmingham, UK.
Herbrich, R., & Graepel, T. (2006). *TrueSkill: A Bayesian skill rating system*. (Tech. Rep. No. MSR-TR-2006-80). Cambridge, UK: Microsoft Research.
Heskes, T., Opper, M., Wiegerinck, W., Winther, O., & Zoeter, O. (2005). Approximate inference with expectation constraints. *Journal of Statistical Mechanics: Theory and Experiment, 11*, P11015-1–P11015-24.
Heskes, T., & Zoeter, O. (2002). Expectation propagation for approximate inference in dynamic Bayesian networks. In A. Darwiche & N. Friedman (Eds.), *Proceedings of the 18th conference on uncertainty in artificial intelligence* (pp. 216–223). San Francisco: Morgan Kaufmann.
Minka, T. (2001). *A family of algorithms for approximate Bayesian inference*. PhD thesis, Cambridge, MA: MIT.
Minka, T. (2005). *Divergence measures and message passing*. (Tech. Rep. NO. MSR-TR-2005-173), Cambridge, UK: Microsoft Research.
Minka, T., & Lafferty, J. (2002). Expectation-propogation for the generative aspect model. In A. Darwiche & N. Friedman (Eds.), *Proceedings of the 18th conference on uncertainty in artificial intelligence* (pp. 352–359). San Francisco: Morgan Kaufmann.
Opper, M., & Winther, O. (2001). Tractable approximations for probabilistic models: The adaptive Thouless-Anderson-Palmer mean field approach. *Physical Review Letters, 86*, 3695–3699.
Seeger, M. (2005). *Expectation propagation for exponential families* (Tech. Rep.). Berkeley, CA: University of California.
Welling, M., Minka, T., & Teh, Y. (2005). Structured region graphs: Morphing EP into GBP. In F. Bacchus & T. Jaakkola (Eds.), *Proceedings of the 21st conference on uncertainty in artificial intelligence (UAI)* (pp. 609–614). Arlington, VA: AUAI Press.

Expectation-Maximization Algorithm

Synonyms

EM Algorithm; Expectation Maximization Algorithm

Expectation-Maximization (EM) was described by Arthur Dempster, Nan Laird, and Donald Rubin in a classic 1977 paper in the Journal of the Royal Statistical Society. The EM algorithm is used for finding maximum likelihood estimates of parameters in stochastic models, where the model depends on unobserved latent or hidden variables. EM iterates between performing expectation (E) and maximization (M) steps. Each expectation step involves the computation of the expectation of the likelihood of all model parameters by including the hidden variables as if they were observed. Each maximization step involves the computation of the maximum likelihood estimates of the parameters by maximizing the expected likelihood found during the expectation step. The parameters produced by the maximization step are then used to begin another expectation step, and the process is repeated.

It can be shown that an EM iteration will not decrease the observed data likelihood function. However, there is no guarantee that the iteration converges to a maximum likelihood estimator.

"Expectation-maximization" has developed to be a general recipe and umbrella term for a class of algorithms that iterates between a type of expectation and maximization step. The Baum–Welch algorithm is an example of an EM algorithm specifically suited to HMMs.

Experience Curve

►Learning Curves in Machine Learning

Experience-Based Reasoning

►Case-Based Reasoning

Explanation

In ►Minimum Message Length, an *explanation* is a code with two parts, where the first part is an *assertion* code and the second part is a *detail* code.

Explanation-Based Generalization for Planning

►Explanation-Based Learning for Planning

Explanation-Based Learning

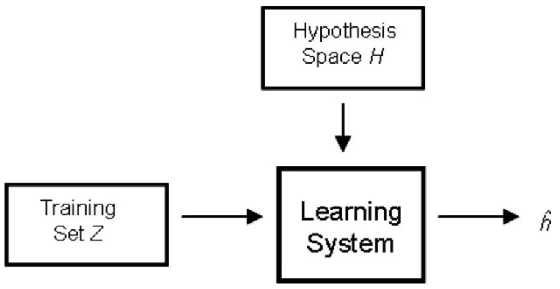
Gerald DeJong[1], Shiau Hong Lim[2]
[1]University of Illinois at Urbana
Urbana, IL, USA
[2]University of Illinois,
IL, USA

Synonyms

Analytical learning; Deductive learning; EBL; Utility problem

Definition

Explanation-Based Learning (EBL) is a principled method for exploiting available domain knowledge to improve ►supervised learning. Improvement can be in speed of learning, confidence of learning, accuracy of the learned concept, or a combination of these. In modern EBL the domain theory represents an expert's approximate knowledge of complex systematic world behavior. It may be imperfect and incomplete. Inference over the domain knowledge provides *analytic* evidence that compliments the empirical evidence of the training data. By contrast, in original EBL the domain theory is required to be much stronger; inferred properties are guaranteed. Another important aspect of modern EBL is the interaction between domain knowledge and labeled training examples afforded by explanations. Interaction allows the nonlinear combination of evidence so that the resulting information about the target concept can be much greater than the sum of the information from each evidence source taken independently.



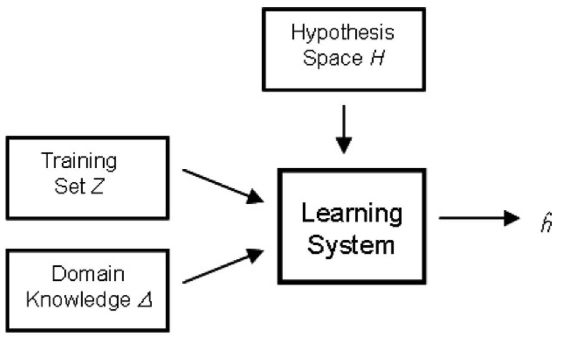
Explanation-Based Learning. Figure 1. Conventional learner

Motivation and Background

A conventional machine learning system is illustrated in Fig. 1. A hypothesis $\hat{h}$ is selected from a space of candidates H using a training set of labeled examples Z as evidence. It is common to assume that the examples are drawn from some space of well-formed inputs X according to some fixed but unknown distribution $\mathcal{D}$. The quality of $\hat{h}$ is to be judged against different examples similarly selected and labeled. The correct label for an example is specified by some ideal *target concept*, c^*. This is typically some complex world process whose outcome is of interest. The target concept, c^*, will generally not be a member of space of acceptable candidates, H. Rather, the learner tries to find some $\hat{h}$ which is acceptably similar to c^* over $X_{\mathcal{D}}$ and can serve as a computationally tractable stand-in.

Of course, good performance of $\hat{h}$ on Z (its training performance) alone is insufficient. The learner must achieve some statistical guarantee of good performance on the underlying distribution (test performance). If H is too rich and diverse or if Z is too impoverished, a learner is likely to ►overfit the data; it may find a pattern in the training data that does not hold in the underlying distribution $X_{\mathcal{D}}$. Test performance will be poor despite good training performance.

An Explanation-Based Learner employs its domain theory, Δ (Fig. 2) as an additional source of information. This domain theory must not be confused with ►learning bias, which is present in all learners. Determinations (Russell & Grosof, 1987) provide an extreme



Explanation-Based Learning. Figure 2. EBL learner

illustration. These are logical expressions that make strong claims about the world but only after seeing a training example. EBL domain theories are used only to explain. An inferred expression is not guaranteed to hold but only provides analytic evidence.

An explanation for some $z \in Z$ is immediately and easily generalized: The structure of the explanation accounts for why *z*s assigned classification label should follow from its features. All other examples that meet these conditions are assigned the same classification by the generalized explanation for the same reasons.

Early approaches to EBL (e.g., DeJong & Mooney, 1986; Mitchell, 1997; Mitchell, Keller, & Kedar-Cabelli, 1986; Russell & Norvig, 2003) were undone by two difficult problems: (1) unavoidable imperfections in the domain theory and (2) the utility problem. The former stems from assuming a conventional semantics for the domain theory. It results in a brittleness and an under-reliance on the training data. Modern EBL is largely a reaction to this difficulty. The utility problem is a consequence of an ill-defined hypothesis space and, as will be discussed later, can be avoided in a straightforward manner.

Structure of Learning System

Explanations and Their Generalization

An *explanation* for a training example is any causal structure, derivable from Δ, which justifies why this training example might merit its teacher-assigned classification label. A *generalized explanation* is the structure of an explanation without the commitment to any particular example. The explanation and generalization processes are relatively straightforward and not significantly different from the original EBL algorithms.

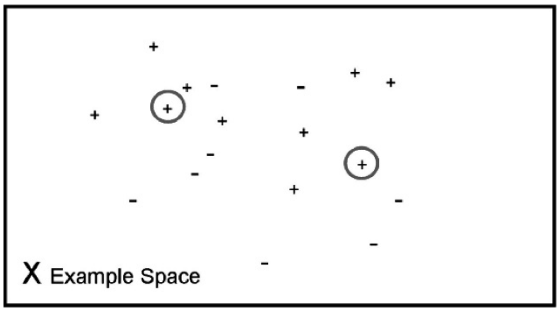
The weakness of early EBL is in viewing the components of Δ as constraints. This leads to a view of explanations and their generalizations as *proofs*. Real-world brittleness due to the qualification problem (McCarthy, 1980) follows inevitably. In modern EBL, Δ is seen as approximating the underlying world constraints (DeJong, 2006; Kimmig, De Raedt, & Toivonen, 2007). The domain theory is fundamentally a statistical device. Its analytic evidence and the empirical evidence of the training examples both provide a bridge to the real world.

The domain theory introduces new predicates and specifies their significant potential interactions. From a statistical point of view, these are named latent (hidden) features together with a kind of grammar for constructing alternative estimators for them. In short, the domain theory compactly specifies a large set of conceptual structures that an expert believes may be useful making sense of the domain. If the expert is correct, then patterns of interest will become computationally much more accessible via analytic inference.

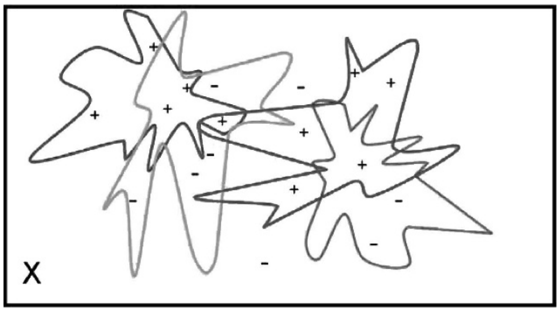
One flexible and useful form of a domain theory is sound inference over a set of first-order symbolic logic sentences. In such domain theories, the explanation mechanism can be identical to logical deduction although using a paraconsistent inference mechanism; inference must be well behaved despite inconsistencies in the theory. Generalized explanations are simply "theorems" of Δ that relate a classification label to the values of observable example features. But since the sentences of the theory only approximate world constraints, derivation alone, even via sound inference, is not sufficient evidence to believe a conclusion. Thus, a generalized explanation is only a conjecture. Additional training examples beyond those used to generate each explanation help to estimate the utility of these generalizations.

But analytic mechanisms need not be limited to symbolic logic-like inference. For example, one EBL approach is to distinguish handwritten Chinese characters (Lim, Wang, & DeJong, 2007) employing a Hough transform as a component of the domain theory. There, an explanation conjectures (hidden) glyph "strokes" to explain how the observed pixels of the training images may realize the image's character label.

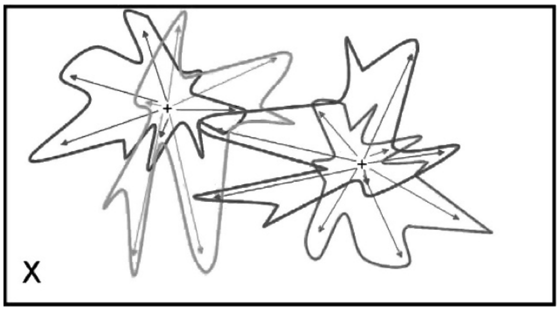
Whatever the form of the analytic inferential mechanism, multiple, quite incompatible explanations can be generated; the same training label can be explained



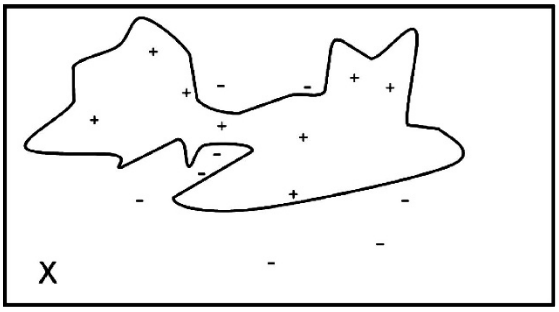
Explanation-Based Learning. Figure 3. An example space with two designated positive training items



Explanation-Based Learning. Figure 5. Explanations are evaluated with other training examples



Explanation-Based Learning. Figure 4. Four constructed explanations are sufficient to cover the positive examples



Explanation-Based Learning. Figure 6. An element from *H* that approximates the weighted explanations

using different input features and postulating different interactions. Such explanations will generalize to cover quite different subsets of X. Figure 3 shows a small training set with two positive examples highlighted. While the explanation process can be applied to all examples both positive and negative, these two will be used to illustrate. In this illustration, just two explanations are constructed for each of the highlighted training examples. Figure 4 shows the generalized extensions of these four explanations in the example space. The region enclosed by each contour is meant to denote the subset of X conjectured to merit the same classification as the explained example. Explanations make no claim about the labels for examples outside their extension.

Evaluation and Hypothesis Selection

Additional training examples that fall within the extension of a generalized explanation help to evaluate it empirically. This is shown in Fig. 5. The estimated utility of a generalized explanation reflects (1) the generalized explanation's empirical accuracy on these training examples, (2) the inferential effort required to derive the explanation (see DeJong, 2006), and (3) the redundancies and interactions with other generalized explanations (higher utility is estimated if its correct predictions are less commonly shared by other generalized explanations).

The estimated utilities define an EBL classifier as a mixture of the generalized explanations each weighted by its estimated utility:

$$\hat{c}_{\mathrm{EBL}}(x) = \sum_{g \in GE(Z,\Delta)} u_g \cdot g(x),$$

where $GE(Z, \Delta)$ denotes the generalized explanations for Z from Δ and u_g is the estimated utility for g. This corresponds to a voting scheme where each generalized explanation that claims to apply to an example casts a vote in proportion to its estimated utility. The votes are normalized over the utilities of voting generalized explanations. The mixture scheme is similar to that of sleeping experts (Freund, Schapire, Singer, & Warmuth, 1997). This EBL classifier approximates the target concept c^*. But unlike the approximation chosen by a conventional learner, $\hat{c}_{\mathrm{EBL}}$ reflects the information of Δ in addition to Z.

The final step is to select a hypothesis $\hat{h}$ from H. The EBL concept $\hat{c}_{\mathrm{EBL}}$ is used to guide this choice. Figure 6 illustrates the selection of a $\hat{h} \in H$, which is a good approximation to a utility-blended mixture of Fig. 5. This final step, selecting a hypothesis from H, is important but was omitted in original EBL. These systems employed generalized explanations directly. Unfortunately, such classifiers suffer from a difficulty known as the *utility problem* (Minton, 1990). Note this is a slightly different use of the term *utility*, referring to the performance of an application system. This system can be harmed more than helped by concepts such as $\hat{c}_{\mathrm{EBL}}$, even if these concepts provide highly accurate classification. Essentially, the average cost of evaluating an EBL concept may outweigh the average benefit that it provides to the application system. It is now clear that this utility problem is simply the manifestation of a poorly structured hypothesis space. Note that, in general, an EBL classifier itself will not be an element of the space of acceptable hypotheses H. Previous approaches to the utility problem (Etzioni, 1993; Gratch & DeJong, 1992; Greiner & Jurisica, 1992; Minton, 1990) identify and disallow offending EBL concepts. However, the root cause is addressed by employing the EBL concept as a guidance in selecting a $\hat{h} \in H$ rather than using $\hat{c}_{\mathrm{EBL}}$ directly. Without this last step, H is completely ignored. But H embodies all of the information in the learning problem about what makes an acceptable hypothesis. The "utility problem" is simply the manifestation of leaving out this important information.

Literature

The roots and motivation for EBL extend at least to the MACROPs of STRIPS (Fikes, Hart, & Nilsson, 1972). The importance of explanations of training examples was first suggested in DeJong (1981). The standard references for the early EBL work are Mitchell et al. (1986) and DeJong and Mooney (1986). When covering EBL, current textbooks give somewhat refined versions of this early approach (Mitchell, 1997; Russell & Norvig, 2003). Important related ideas include determinations (Russell & Grosof, 1987), chunking (Laird, Rosenbloom, & Newell, 1986), and knowledge compilation (Anderson, 1986). EBL's ability to employ first-order theories make it an attractive compliment to learning Horn theories with ►Inductive Logic Programming (Bruynooghe, De Raedt, & De Schreye, 1989; Hirsh, 1987; Pazzani & Kibler, 1992; Zelle & Mooney, 1993). The problem of imperfect domain theories was recognized early, and there have been many approaches (Cohen, 1992; Flann & Dietterich, 1989; Genest, Matwin, & Plante, 1990; Ourston & Mooney, 1994; Thrun & Mitchell, 1993; Towell, Craven, & Shavlik, 1991). But with modern statistical learning ascending to the dominant paradigm of the field, interest in analytic approaches waned. The current resurgence of interest is largely driven by placing EBL in a modern statistically sophisticated framework that nonetheless is still able to exploit a first-order expressiveness (DeJong, 2006; Kimmig et al., 2007; Lim et al., 2007; Sun & DeJong, 2005)

Cross References

►Explanation-Based Learning for Planning
►Speedup Learning

Recommended Reading

Anderson, J. (1986). Knowledge compilation: The general learning mechanism. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning II* (pp. 289–310). San Mateo, CA: Morgan Kaufmann.

Bruynooghe, M., De Raedt, L., & De Schreye, D. (1989). Explanation based program transformation. In *IJCAI* (pp. 407–412).

Cohen, W. W. (1992). Abductive explanation-based learning: A solution to the multiple inconsistent explanation problem. *Machine Learning, 8*, 167–219.

DeJong, G. (1981). Generalizations based on explanations. In *IJCAI'81, the seventh international joint conference on artificial intelligence* (pp. 67–69). Vancover, BC.

DeJong, G. (2006). Toward robust real-world inference: A new perspective on explanation-based learning. In *ECML06, the seventeenth European conference on machine learning* (pp. 102–113). Heidelberg: Springer.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*(2), 145–176.

Etzioni, O. (1993). A structural theory of explanation-based learning. *Artificial Intelligence, 60*(1), 93–139.

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

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

Freund, Y., Schapire, R. E., Singer, Y., & Warmuth, M. K. (1997). Using and combining predictors that specialize. In *Twenty-ninth annual ACM symposium on the theory of computing* (pp. 334–343). El Paso, TX.

Genest, J., Matwin, S., & Plante, B. (1990). Explanation-based learning with incomplete theories: A three-step approach. In proceedings of the seventh international conference on machine learning (pp. 286–294).

Gratch, J., & DeJong, G. (1992). Composer: A probabilistic solution to the utility problem in speed-up learning. In *AAAI* (pp. 235–240).

Greiner, R., & Jurisica, I. (1992). A statistical approach to solving the EBL utility problem. In *National conference on artificial intelligence* (pp. 241–248). San Jose, CA.

Hirsh, H. (1987). Explanation-based generalization in a logic-programming environment. In *IJCAI* (pp. 221–227). Milan, Italy.

Kimmig, A., De Raedt, L., & Toivonen, H. (2007). Probabilistic explanation based learning. In *ECML'07, the eighteenth European conference on machine learning* (pp. 176–187).

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning, 1*(1), 11–46.

Lim, S. H., Wang, L.-L., & DeJong, G. (2007). Explanation-based feature construction. In *IJCAI'07, the twentieth international joint conference on artificial intelligence* (pp. 931–936)

McCarthy, J. (1980). Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence, 13*, 27–39.

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*(2–3), 363–391.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*(1), 47–80.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence, 66*(2), 273–309.

Pazzani, M. J., & Kibler, D. F. (1992). The utility of knowledge in inductive learning. *Machine Learning, 9*, 57–94.

Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Russell, S. J., & Grosof, B. N. (1987). A declarative approach to bias in concept learning. In *AAAI* (pp. 505–510). Seattle, WA.

Sun, Q., & DeJong, G. (2005). Feature kernel functions: Improving svms using high-level knowledge. In *CVPR (2)* (pp. 177–183)

Thrun, S., & Mitchell, T. M. (1993). Integrating inductive neural network learning and explanation-based learning. In *IJCAI* (pp. 930–936). Chambery, France.

Towell, G. G., Craven, M., & Shavlik, J. W. (1991). Constructive induction in knowledge-based neural networks. In proceedings of the eighth international conference on machine learning (pp. 213–217)

Zelle, J. M., & Mooney, R. J. (1993). Combining Foil and EBG to speed-up logic programs. In *IJCAI* (pp. 1106–1113). Chambery, France.

Explanation-Based Learning for Planning

Subbarao Kambhampati[1], Sungwook Yoon[2]
[1]Arizona State University, Tempe, AZ, USA
[2]Palo Alto, CA, USA

Synonyms

Explanation-based generalization for planning; Speedup learning for planning

Definition

►Explanation-based learning (EBL) involves using prior knowledge to explain ("prove") why the training example has the label it is given, and using this explanation to guide the learning. Since the explanations are often able to pinpoint the features of the example that justify its label, EBL techniques are able to get by with much fewer number of training examples. On the flip side, unlike general classification learners, EBL requires prior knowledge (aka "domain theory/model") in addition to labeled training examples – a requirement that is not easily met in some scenarios. Since many planning and problem solving agents do start with declarative domain theories (consisting at least descriptions of actions along with their preconditions and effects), EBL has been a popular learning technique for planning.

Dimensions of Variation

The application of EBL in planning varies along several dimensions: whether the learning was for improving the speed and quality of the underlying planner (Etzioni, 1993; Kambhampati, 1994; Kambhampati, Katukam, & Qu, 1996; Minton et al., 1989; Yoon, Fern, & Givan, 2008) or acquire the domain model (Levine & DeJong, 2006); whether it was was done from successes (Kambhampati, 1994; Yoon et al. 2008) or failures (Ihrig

& Kambhampati, 1997; Minton et al., 1989); whether the explanations were based on complete/correct (Kambhampati et al., 1996; Minton et al., 1989) or partial domain theories (Yoon et al.), whether learning is based on single (Kambhampati, 1994; Kambhampati et al.; Minton et al., 1989) or multiple examples (Estlin & Mooney, 1997; Flann & Dietterich, 1989) (where, in the latter case, inductive learning is used in conjunction with EBL) and finally whether the planner whose performance EBL aims to improve is a means-ends analysis one (Minton et al., 1989), partial-order planner (Estlin & Mooney, 1997) or a heuristic search planner (Yoon et al.).

EBL techniques have been used in planning both to improve search and to reduce domain modeling burden (although the former has received more attention by far). In the former case, EBL is used to learn "control knowledge" to speedup the search process (Kambhampati et al., 1996; Minton et al., 1989), or to improve the quality of the solutions found by the search process (Estlin & Mooney, 1997). In the latter case EBL is used to develop domain models (e.g., action models) (Levine & DeJong, 2006).

EBL for search improvement involves either remembering and reusing successful plans, or learning search control rules to avoid failing search branches. Other variations include learning effective indexing of stored cases from retrieval failures (Ihrig & Kambhampati, 1997) and learning "adjustments" to the default heuristic used by the underlying search.

Another important issue is the degree of completeness/correctness of the underlying background theory used to explain examples. If the theory is complete and correct, then learning is possible from a single example. This type of EBL has been called "analytical learning." When the theory is partia, EBL still is effective in narrowing down the set of potentially relevant features of the training example. These features can then be used within an inductive learner. Within planning, EBL has been used in the context of complete/correct as well as partial domain models.

A final dimension of variation that differentiated a large number of research efforts is the type of underlying planner. Initially, EBL was used on top of means-ends analysis planners (cf. PRODIGY, Minton et al., 1989). Later work focused on partial order planners (e.g., Estlin & Mooney, 1997; Kambhampati et al., 1996). More recently, the focus has been on forward search state-space planners (Yoon et al., 2008).

Learning from Success: Explanation-Based Generalization

When learning from successful cases (plans), the training examples comprise of successful plans, and the explanations involve proofs showing that the plan, as it is given, is able to support the goals. Only the parts of the plan that take part in this proof are relevant for justifying the success of the plan. The plan is thus "generalized" by removing extraneous actions that do not take part in the proof. Object identifiers and action orderings are also generalized as long as the generalization doesn't affect the proof of correctness (Kambhampati, 1994). The output of the learning phase is thus a variablized plan containing a subset of the constraints (actions, orderings, object identity constraints) of the original plan. This is then typically indexed and used as a macro-operator to speed-up later search.

For example, given a planning problem of starting with an initial state where five blocks, A, B, C, D and E are on table, and the problem requires that in the goal state A must be on B and C must be on D, and a plan P that is a sequence of actions *pickup A, stack A on B, pickup E, putdown E, Pickup C, stack C on D*, the explanation-based learner might output the generalization "*do in any order* {*pickup x, stack x on y*} {*pick up z, stack z on w*}" for the generalized goals *on* (x, y) *and on* (w, z), starting from a state where x, y, z and w are all on table and clear, and each of them denotes a distinct block.

One general class of such proof schema involves showing that every top level goal of the planning problem as well as the precondition of every action are established and protected. Establishment requires that there is an action in the plan that gives that condition, and protection requires that once established, the condition is not deleted by any intervening action.

A crucial point is that the extent of generalization depends on the flexibility of the proof strategy used. Kambhampati and Kedar (1994) discuss a spectrum of generalization strategies associated with a spectrum of proof strategies, while Shavlik (1990) discusses how the number of actions in the plan can also be generalized.

Learning from Failure

When learning from the failure of a search branch, EBL starts by analyzing the plans at the failing nodes and constructing an explanation of failure. The failure explanation is just a subset of constraints in the plan at the current search node, which, in conjunction with domain theory ensures that no successful solution can be reached by further refining this plan. The explanations can range from direct constraint inconsistencies (e.g., ordering cycles), to indirect violation of domain axioms (e.g., the plan requiring both clear(B) and On(A,B) to be satisfied at the same time point). The explanations at the leaf nodes are "regressed" over the decisions in the search tree to higher level nodes to get explanations of (implicit) failures in these higher level nodes. The search control rules can then essentially recommend pruning any search node which satisfies a failure explanation.

The deep affinity between EBL from search failures and the idea of ►nogood learning and dependency-directed backtracking in CSP is explored in Kambhampati (1998). As in dependency directed backtracking, the more succinct the explanation, the higher the chance of learning effective control rules. Note that effectiveness here is defined in terms of the match costs involved in checking whether the rule is applicable, and the search reductions provided when it is applicable. Significant work has been done to identify classes of failure explanation that are expected to lead to ineffective rules (Etzioni, 1993). In contrast to CSP that has a finite depth search tree, one challenge in planning is that often an unpromising search node might not exhibit any direct failure with a succinct explanation, and is abandoned by the search for heuristic reasons (such as the fact that the node crosses a depth limit threshold). Strategies for finding implicit explanations of failure (using domain axioms), as well as getting by with incomplete explanations of failure are discussed in Kambhampati et al. (1996). EBL from failures has also been applied to retrieval (rather than search) failures. In this case, the failure of extending a plan retrieved from the library to solve a new problem is used to learn new indexing schemes that inhibit that case from being retrieved in such situations (Ihrig & Kambhampati, 1997).

Learning Adjustments to Heuristics

Most recent work in planning has been in the context of heuristic search planners, where learning from failures doesn't work as well (since the heuristic search may change directions much before a given search branch ends in an explainable failure). One way of helping such planners is to improve their default heuristic (Yoon et al., 2008). Given a heuristic $h(s)$ that gives the heuristic estimate of state s, the aim in Yoon et al. is to learn an adjustment $\delta(s)$ that is added to $h(s)$ to get a getter estimate of $h^*(s)$ – the true cost of state s. The system has access to actual plan traces (which can be obtained by having the underlying planner solve some problems from scratch). For each state s on the trace, we know the true distance of state s from the goal state, and we can also compute the $h(s)$ value with respect to the default heuristic. This gives the learner a set of training examples which are pairs of states and the adjustments they needed to make to the default heuristic meet the true distance. In order to learn the $\delta(s)$ from this training data, we need to enumerate the features of state s that are relevant to it needing the specific adjustment. This is where EBL come in. Specifically, one way of enumerating the relevant features is to explain why s has the default heuristic value it does. This, in turn, is done by taking the features of the relaxed plan for state s. Since the relaxed plan is a plan that assumes away all negative interactions between the actions, relaxed plan features can be seen as features of the explanation of the label for state s in terms of a *partial domain theory* (one which ignores all the deletes of all actions).

EBL from Incomplete Domain Theories

While most early efforts for speed-up focused on complete and correct theories, several efforts also looked at speed-up learning from incomplete theories. The so-called Lazy EBL approaches (Chien, 1989; Tadepalli, 1989) work by first constructing partial explanations, and subsequently refine the over-general rules learned. Other approaches that use similar ideas outside planning include Flann and Dietterich (1989) and Cohen (1992). As we noted above, the work by Yoon et al. (2008) can also be seen as basing learning (in their case of adjustments to a default heuristic function) w.r.t. a partial domain theory.

EBL to Learn Domain Knowledge

Although most work in EBL for planning has been focused on speedup, there has also been some work aimed at learning domain knowledge (rather than control knowledge). Of particular interest is "operationalizing" a complex, if opaque, domain model by learning from it a simplified domain model that is adequate to efficiently solve an expected distribution of problems. The recent work by Levine and DeJong (2006) is an example of such an effort.

EBL and Knowledge-Level Learning

Although the focus of this article is on EBL as applied to planning, we need to foreground one general issue: whether EBL is capable of knowledge-level learning or not. A popular misconception of EBL is that since it depends on a complete and correct domain theory, no knowledge-level learning is possible, and speedup learning is the only possibility. (The origins of this misconception can be traced back to the very beginning. The two seminal articles on EBL in the very first issue of the Machine Learning journal differed profoundly in their interpretations of EBL. While Mitchell, Keller, and Kedar-Cabelli (1986) assumed that EBL by default works with complete and correct theories (thus precluding any knowledge-level learning), DeJong (2006) provide a more general view of EBL that uses background knowledge – whether or not it is complete – to focus the generalization (and as such can be seen as a knowledge-based feature-selection step for a subsequent inductive learner)). As we noted at the outset however, EBL is not required to depend on complete and correct domain theories, and when it doesn't, knowledge level learning is indeed possible.

Utility Problem and its Non-Exclusive Relation to EBL

As we saw above, much early work in EBL for planning focused on speed-up for the underlying planner. Some of the knowledge learned for speedup – especially control rules and macro-operators – can also adversely affect the search by increasing either the search space size (macros) and/or per-node cost (matching control rules). Clearly, in order for the net effect to be positive, care needs to be exercised as to which control rules and/or macros are stored. This has been called the "utility problem" (Minton, 1990) and significant attention has been paid to develop strategies that either dynamically evaluate the utility of the learned control knowledge (and forget useless rules) (Markovitch & Scott, 1988; Minton, 1990), or select the set of rules that best serve a given distribution of problem instances (Gratch, Chien, & DeJong, 1994).

Despite the prominent attention given to the utility problem, it is important to note the non-exclusive connection between EBL and utility problem We note that *any* strategy that aims to provide/acquire control knowledge will suffer from the utility problem. For example, utility problem also holds for inductive learning techniques that were used to learn control knowledge (cf. Leckie & Zukerman, 1993). In other words, it is not special to EBL but rather to the specific application task. We note that it is both possible to do speedup learning that is less suceptible to the utility problem (e.g., learn adjustments to heuristics, Yoon et al., 2008), and possible to to use EBL for knowledge-level learning (Levine & DeJong, 2006).

Current Status

EBL for planning was very much in vogue in late eighties and early nineties. However, as the speed of the underlying planners increased drastically, the need for learning as a crutch to improve search efficiency reduced. There has however been a recent resurgence of interest, both in further speeding up the planners, and in learning domain models. Starting 2008, there is a new track in the International Planning Competition devoted to learning methods for planning. In the first year, the emphasis was on speedup learning. ObtuseWedge, a system that uses EBL analysis to learn adjustments to the default heuristic, was among the winners of the track. The DARPA integrated learning initiative, and interest in model-lite planning have also brought focus back to EBL for planning – this time with partial domain theories.

Additional Reading

The tutorial (Yoon & Kambhampati, 2007) provides an up-to-date and broader overview of learning techniques applied to planning, and contains significant discussion of EBL techniques. The paper (Zimmerman & Kambhampati, 2003) provides a survey of machine learning techniques used in planning, and includes a more comprehensive listing of research efforts that applied EBL in planning.

Cross References

►Explanation-Based Learning
►Speedup Learning

Recommended Reading

Bhatnagar, N., & Mostow, J. (1994). On-line learning from search failures. *Machine Learning, 15*(1), 69–117.

Borrajo, D., & Veloso, M. M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review, 11*(1–5), 371–405.

Chien, S. A. (1989). Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *IJCAI* (pp. 590–595).

Cohen, W. W. (1992). Abductive explanation-based learning: A solution to the multiple inconsistent explanation problem. *Machine Learning, 8*, 167–219.

DeJong, G., & Mooney, R. J. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*(2), 145–176.

Estlin, T. A., & Mooney, R. J. (1997). Learning to improve both efficiency and quality of planning. In *IJCAI 1997* (pp. 1227–1233).

Etzioni, O. (1993). A structural theory of explanation-based learning. *Artificial Intelligence, 60*(1), 93–139.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

Gratch, J., Chien, S. A., & DeJong, G. (1994). Improving learning performance through rational resource allocation. In *AAAI 1994* (pp. 576–581).

Ihrig, L. H., & Kambhampati, S. (1997). Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Journal of Artificial Intelligence Research, 7*, 161–198.

Kambhampati, S. (1994). A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence, 67*(1), 29–70.

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

Kambhampati, S., Katukam, S., & Qu, Y. (1996). Failure driven dynamic search control for partial order planners: An explanation based approach. *Artificial Intelligence, 88*(1–2), 253–315.

Leckie, C., & Zukerman, I. (1993). An inductive approach to learning search control rules for planning. In *IJCAI 1993* (pp. 1100–1105)

Levine, G., & DeJong, G. (2006). Explanation-based acquisition of planning operators. In *ICAPS 2006* (pp. 152–161).

Markovitch, S., & Scott, P. D. (1988). The role of forgetting in learning. In *ML 1988* (pp. 459–465).

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*(2–3), 363–391.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*(1–3), 63–118.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*(1), 47–80.

Shavlik, J. W. (1990). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning, 5*, 39–40.

Tadepalli, P. (1989). Lazy explanation based learning: A solution to the intractable theory problem. In *IJCAI 1989* (pp. 694–700).

Yoon, S., Fern, A., & Givan, R. (2008). Learning control knowledge for forward search planning. *Journal of Machine Learning Research, 9*, 683–718.

Yoon, S., & Kambhampati, S. (2007). Learning for planning. Tutorial delivered at ICAPS 2007. http://rakaposhi.eas.asu.edu/learn-plan.html

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