ΠP periodica polytechnica

Electrical Engineering
55/1-2 (2011) 21–29
doi: 10.3311/pp.ee.2011-1-2.03
web: http://www.pp.bme.hu/ee
© Periodica Polytechnica 2011

RESEARCH ARTICLE

An open learning system

Borbála Katalin Benkő / *Dávid* Lányi

Received 2011-03-01

Abstract

In this paper we discuss an approach for achieving high adaptivity in complex dynamic environments through learning. Unlike in traditional approaches, where adaptive systems need to be strongly equipped with knowledge about their environment and possible adaptation strategies, we propose a direction where the level of original preconceptions is kept low. In our model, both the knowledge and the strategies emerge dynamically and flexibly, during the system's normal operation. The proposed model also includes a mechanism that systematically replaces old preconceptions with more accurate ones (e.g. to observe new features). The proposed principles were evaluated in a theoretical world via simulation; where the adaptive system was challenged to win an amoeba game against opponents with different strategies, and changing game rules (3-7 long series required to win).

Keywords

collective-adaptive systems · machine learning · knowledge sharing · dynamic requirements

Borbála Katalin Benkő
Dávid Lányi
Department of Telecommuncations, BME, H-1117 Budapest, Magyar tud. krt. 2, Hungary

1 Introduction

Adaptive systems possess the ability of adjusting themselves to the changes of the environment by fine-tuning their operational parameters or altering their behavioral strategy according to the observed situation. Typically, adaptive systems use a feedback loop to achieve this: observe the actual state of the world, determine the best action to do, and carry out the action [5]. Compared to traditional control systems, an immense difference is that in adaptive case there is no preferred setting to find-the world, so the adaptation target changes from time to time, and the system must follow it, accordingly.

A traditional approach for achieving adaptivity is to maintain a *world model*, detect when *changes* occur, and explicitly start an adjustment process within the feedback loop. Powerful tools like reasoning, semantics and ontology guarantee that the adaptation is effective and convergent [2, 7, 8]. However, the success of this approach largely relies on the accuracy of the system's *explicit knowledge*: on the completeness of the world model and on the efficiency of built-in adjustment mechanisms. As long as environmental changes are in line with this knowledge, the system is guaranteed to adapt efficiently; but it is theoretically impossible to react to changes that were not handled by the world model, or to select adjustment strategies that were not encoded previously. In other words, preconceptions that make the adaptation fast and effective in some cases form an impenetrable obstacle in others. The adaptation potential of the system is highly restricted by these hard preconceptions [3].

Our first research question was if the burdens of this explicit knowledge can be avoided, in order to unbind the learning process, and let the system openly find its way for adaptation (instead of blindly following what scientist told it to do). Openness is a goal on several layers: an open and flexible world model which is free of hard preconceptions (e.g. the initial preconceptions may be refined or removed based on the perceived evidence); open adaptation strategy building where the system develops behavioral directions itself from simple building blocks instead of picking one of the injected preconceptional strategies; and, finally, social-communicational openness, to share knowledge with others and make use of the presence of multitude of learners. An extremity of such an open system is one without any pre-injected model or strategy, an idea which tackles with dimensions yet unknown in dynamic adaptation.

The topic described above obviously includes learning: we proposed a system that not only adapts but also learns how to adapt. However, in case of an adaptive system, it is usually not easy or even not possible to gather feedback about the goodness of each action – usually, the system senses failure and reward situations but anything else is in a grey cloud. Hence, the learning model must handle the lack of continuous feedback, which leaves only a few options for methodology choice.

Most of today's learning based adaptive systems differentiate between *learning tim*e and *operation time*. At learning time, excessive simulations are used for finding an optimal parameter and strategy setting; while at operation time, this 'optimal' setting is applied without further modification. The learning phase often includes soft computing methodologies such as genetic algorithms [6]. The main problem of this approach is that the learning phase uses simulation (models the response of the world), and if this model is inaccurate or the world simply changes meanwhile, then the learned results will not be effective in the real environment. Another bottleneck is that the learning phase is separated from the system's normal operation.

Our aim was to create/use a learning model which provides seamless adaptation, and adapts to the real, actual problem case instead of a simulated problem case. Learning must take place during – and throughout – the system's normal operation, ready to react to changes at any time. As a consequence, the real response of the real world is available after each action, so the system needs much less to guess or simulate.

In this paper we discuss further details of the knowledge-poor open adaptation model, originally published in [9]. The model is called knowledge-poor because the level of explicit, pre-injected modeling is kept very low. In this paper we investigate the model from three new aspects.

- Optimization: we propose a novel mechanism that helps in keeping the knowledge optimized, both in means of memory space and in means of descriptive power and convergence. This mechanism also enables the agent to develop new observation viewpoints which contribute to a better – less redundant and smaller – world model and more speedy convergence.
- Adaptation characteristics to different kind of changes: how much the system is able to adapt to small changes, or, to completely new requirements.
- The social perspective: we investigate further aspects of the knowledge sharing mechanism, specifically, the benefits and drawbacks of the knowledge injection in light of the optimized feature set.

The paper is organized as follows. Section 2 specifies the problem space. Section 3 presents the basic model of the knowledge-poor open adaptation. Section 4 introduces the optimization mechanism that removes redundancy and failed branches, gaining advantages in both calculation complexity and in the speed of convergence. Section 5 tackles shortly with the collective dimension of the learning. Section 6 describes further general aspects and consequences. Section 7 summarizes the results.

2 Problem statement

The problem space considered here is a world with actors who observe their environment and make actions from time to time. Certain states of the world mean reward to the actor who reaches it, while other states result in penalty. We made the following restrictions.

- **Limited information.** Actors use a rough, simplified world model (instead of a fully observable complete one). At startup, they have no preconceptions about rules, requirements or strategies available. Furthermore, the observation ability of actors is limited as well: they may not gather information about all possible aspects of the complete world. – typically they are able to observer their direct neighborhood along some built-in or developed perspective.
- **Time series model.** We focus on scenarios which can be described in means of discrete time-steps (from our actor's point of view). In each logical time step, either our actor or the other actor(s) make an action. (For the sake of generality, a pseudo-actor may be used for modeling those changes in the world that were not caused by the real actors.) In order to keep things simple, the actor "knows" when it is its turn.
- **Dynamic goals.** The environment is not only dynamic because of the effect of other actors; but also in means of the goal to reach (general requirements or rules). In our model, the rules may dynamically change from time to time. Please note that this is a realistic requirement in many real-life scenarios, e.g. when the meta-goal is to make the human user happy, and the human's intentions or habits change from time to time.
- **No information injection.** Actors get known with the actual requirements only implicitly, through the received reward or penalty. Feedback (reward or penalty) may be provided for the actor after it performs an act, and this reward does not merely refer to the least action but to the state the step resulted. Typically, feedback is only received rarely, so, actors need to decide about their steps without knowing the effect of the previous ones.
- **Collectiveness.** The problem space is not limited to the level of single actors. We allow single actors to communicate with others, and share/accept knowledge.

While numerous concrete problem instances can be imagined satisfying the above specification; we choose a showcase in which the basic abilities of the system can be demonstrated and efficiently evaluated. This is actually a generalization of a simple two-player, deterministic, fully observable, zero-sum board game, often known as connect-5, gomoku, or amoeba.

2.1 Inspiration: The Connect-5 game

Connect-5 is a simple board game, an extension of tic-tac-toe for bigger sized boards and longer combinations. There are two players in the game, one with mark X and the other one with mark O. The game starts with a board of tabular arranged empty square cells. Players make steps intermittently; each one places their mark onto an empty cell. A player wins the game, if five of his signs are placed consecutively in one row, column, or in a diagonal line on the board. When a player wins, the other one looses. Tie is reached, if the board has no more empty cells, but no one has won.

More formally, the *state* of the game is represented by the board itself (cells and their contents). The transition between states is the *action* of an actor, and the game is basically a time series of game states. Only one actor is allowed to perform action in each particular state. The action is mandatory, so if there is an empty cell on the board, the upcoming actor must act. After each action, the new state is evaluated by the environment, and if winner or tie state is reached, actors receive *feedback*.

Actors are able to observe a rough model of the actual environment at any time; and are also able to receive feed-back about winning/losing/tie state. They also may accumulate a local knowledge base from these observations. The goal of the actor is to find a strategy (actions) that leads to a winning state, while the environment is dynamically modified by the opponent from time to time.

2.2 Problem instance: a Generalized Connect-5 World

While the basic Connect-5 world incorporates several important properties of our problem space, we decided to generalize it in order to include further aspects of dynamism and collectiveness.

- **Collectiveness.** Instead of a single actor-opponent pair, the world consists of a society of players, engaged in multitudes of parallel games. The members of the society are still autonomous actors – with individual experience, strategy and decision ability –, but they also possess the ability of communicating with each other. Actors may share their expertise with other actors, or learn from others' shared knowledge. Please note that it is not guaranteed that the members of the society face the same problem instance – e.g. same opponent style or the same rules – nor do we say that the knowledge of any individual agent is guaranteed to be of help for others. However, the pure ability of sharing one's dynamically built knowledge is an important attribute for a collective system, also from the theoretical point of view. Pair-wise knowledge sharing may also – but not necessarily – lead to the emergence of society level "common knowledge".
- **Dynamic opponent style and strength.** The strategy, goodness and consistency of the opponent may change over time, resulting in dynamically changing environ mental requirements from the actor's point of view. Extremities may be a random opponent (just picking random steps), and an analytically optimal opponent (who chooses the mathematically optimal strategy for the actual game state and rule set).
- **Changing game rules.** We also allow the game rules to be changed dynamically during the actor's life cycle. For example, the competitive aspect may be removed, so, the player gets rewarded for their own 5-long series regard less of the other player's moves. Another way of changing the rules is to modify the length of the required series: e.g. 4 or 10 items long series may mean victory.

We believe that this generalized Connect-5 world fits with the problem space specification well. The states of the world form a time series. Actors are able to observe the world's state and perform actions. They may receive feedback from the environment in certain states. The world changes because of the actor's action or because of factors that are outside of the actor's control (e.g. opponent's action). Besides the above, we also made the assumption that the actor has no pre-injected knowledge of the rules of the world or about the goal to reach – it has to reach (positive feedback) game states by 'learning by doing'. This may seem to be a selfish requirement under static conditions (where explicit world modeling could result in optimal behavior from the startup), but our goal here is to ensure the openness of the system and its dynamic adaptation ability for immensely new requirements set by the dynamically changing environment (e.g. changing game rules).

The state space in the showcase example – supposing a limited board size – is finite. However, we do not think this makes the showcase too week or inappropriate, as the observation ability of an agent is finite by definition too (so any board size larger than the player's vision would work as infinite), and the mathematical model we use for learning can be easily extended for non-binary (multi-value or even continuous interval) cases too.

In summary, the agent's job is to learn the dynamically changing rules of the world through a feedback mechanism in order to select actions that lead to success; plus, to do this on-the-fly, without having had any pre-injected knowledge or preliminary training session; and possibly in a collective manner (by knowledge sharing).

3 The basic learning model

This section describes the basic learning and adaptation model used within the open autonomous agents. The model combines known basic models (Markov Decision Process and Temporal Difference Learning) with specific extensions.

3.1 Starting point: Markov Decision Process

The inspiration of our model comes from *reinforcement learning* (RL), a general approach that tackles with problems very similar to our problem statement. RL is not one specific mechanism but a dynamically improving domain of machine learning models. A common approach [1, 12] focuses on finding actions in an actual world state, in order to achieve a goal desired in that context. It is assumed, that an agent completing this task is able to sense the environment to some extent, is able to perform actions which influence that state, and is able to receive feedback from the environment about its success.

A widely used mathematical model describing reinforcement learning problems is the *Markov Decision Process* (MDP). It is defined as a five-tuple (S, A, R, P, γ), where S is the set of world states, A is a set of actions, P is a state transition probability function $P : S \times A \times S \mapsto [0, 1]$, (where $P(s', a, s)$ tells the probability of reaching state s' after performing action a in state s), R is the reward function $R \mapsto \mathcal{R}$, and γ is a discount factor from interval (0, 1]. It's important to note that the observation capacity of the agent is limited; hence, the state observed may significantly differ from the real world state. At this stage, S refers to the real world state (later, it will be replaced by the agent's perception).

RL models are often equipped with value functions and policies to help making automatic decisions. We follow the terminology and notation of [4]. There is a *value function* defined for states which is able to better describe the real utility of a state than the reward function itself (which would only tell whether state is terminal). The value function is associated with a *policy* π, which is a mapping from states to actions, $\pi : S \mapsto A$. A value function for a given policy, $V^\pi : S \mapsto \mathbb{R}$ is defined as the expected discounted sum of rewards received when starting (in t=0) from state s, and following policy π:

$$V^\pi(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| s_0 = s, \pi\right]$$

It can be shown [10] that this value function satisfies Bellman's equation, and may be expressed in the following way:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P(s', a^\pi, s) V^\pi(s')$$

If the reward function R and transition probability function P are both known, this formula can be analytically solved as a linear system. However, in most RL settings – including our case – the probability function P is not known; the agent only has access to a subset of state transitions (own experience) and to the feedback coming from the reward function.

To limit the size of $|S|$ – to keep computations on an easy-to-handle level –, often, estimations or approximations are used instead of exact models or values.

We also introduced a feature extraction step between the raw perceived state and the state principle used within the model. Feature extraction helps highlighting important properties of a state, by preprocessing the raw observation be-fore learning. The exact realization of this feature extraction step is an important attribute of the agent, as the extracted information deeply influences the learning process. In the basic model, the feature extraction mechanism is wired-in (and this is all the knowledge – even though being implicit, we call it knowledge – the agent gets at startup). In general versions of the model, features may be introduced or removed on the fly.

In means of terminology, a feature based linear approximator [10] for the value function is defined as:

$$V^\pi(s) \approx w^T \phi(s)$$

where $\phi \in \mathbb{R}^k$ is a feature vector based abstraction belonging to the state s, and $w \in \mathbb{R}^k$ is a parameter vector.

While the usage of feature vectors was originally suggested in order to keep the computational complexity under control and to be able to deal with large or even infinite state spaces, we use it for two other purposes, respectively: (a) to facilitate convergence with the selection and usage of relevant and meaningful features, and (b) to bring openness into the model through the possibility of dynamically adding and removing features – hence refreshing the implicit world mo-del of the agent.

With these approximations we lose the applicability of Bellman's equation, but there are other efficient ways for finding the solution, such as the LSTD.

3.2 Least-Squares Temporal Difference Method

The Least-Squares Temporal Difference (LSTD) algorithm provides way for finding a parameter vector w that approximately satisfies Bellman's equation. Without the full deduction of the method discussed in [4, 11] and recalled in [10] we denote the main formulae, and review it from the aspect of our setting. LSTD attempts to find a fixed point of the approximation

$$w = \tilde{f}(w) = \operatorname{argmin}_{u \in \mathbb{R}^k} \|\Phi u - (\tilde{R} + \gamma \Phi' w)\|^2$$

in which Φ and Φ' are matrices containing m samples of observed state transitions from s to s' in their rows, represented with $\Phi(s)^T$ and $\Phi(s')^T$ in each row; $\tilde{R}$ is a vector containing the obtained reward r_i for each of the m transitions. Because the term to be minimized contains Euclidian norms only, the optimal fixed point can be analytically determined by solving a linear system $\tilde{A}^{-1}\tilde{b}$, where

$$\tilde{A} = \sum_{i=1}^{m} \phi(s_i)\left(\phi(s_i) - \gamma\phi(s_i')\right)^T ; \quad \tilde{b} = \sum_{i=1}^{m} \phi(s_i) r_i$$

In other words, the only knowledge required by the agent for selecting the desirable next state is only a vector (b) and a matrix (A).

- **Vector b** gives a picture about the perceived goodness of each state, based on the total (positive or negative) reward experienced there.

- **Matrix A** describes the experienced state transition pairs. Transitions model the effect of the agent's action along with the effect of the opponent's action, in one unit. The discount factor helps in distinguishing between states immediately preceding an end state and states that are far away. Please note that this abstraction does not include any preconception about the number or nature of opponents, so the model is also applicable for $n > 2$ players.

From our point of view, the most important property of vector b and matrix A is that they can be constructed iteratively; each new experience means a minor addition to them.

The agent may use two different approaches for translating the knowledge (matrix A, vector b) into an action:

- Calculate the expected effect of each possible action, and select the most desirable one based on the value of the result state. (If the number of actions is too large – or infinite –, a sampled subset of the possible actions may be used, with hierarchical refinement.)
- Analytically identify the most desirable result state and then search for an action that leads to it. This approach is more problematic because (a) it is not guaranteed that the state space is connected enough so an arbitrary end state can be reached from the current state, and (b) transitions are not deterministic because of the effect of the opponent, so we may end up in a very different end state than desired.

We used the approach (1) in the model.

3.3 The model

Our open adaptation model uses an extended version of LSTD. The world model of the agent is directly incorporated in the matrix A and vector b, and indirectly in the feature extraction mechanism. The decision making uses LSTD's maximum likelihood decision maker. In each turn, the agent evaluates two things: the provisory effect of its own action (the value of the reached state) and the provisory effect of all the actions of other agents.

The extensions we made on LSTD are the followings:

- Dynamic features. The feature extraction mechanism is not static, new features may be added or removed at runtime. (When the feature set changes, A and b get modified accordingly – rows/columns are added or removed.)
- Knowledge optimiztaion. We created a systematic optimization algorithm which removes redundancy and failed branches from the agent's knowledge. This algorithm reduces the size of A and b, as well as the feature extraction logic, automatically.
- Collective layer. We added a collective layer, enabling agents to exchange and combine their knowledge (A, b).

4 Knowledge optimization

The number and goodness of features is a key factor in the convergence speed of the agent's knowledge. *Too many features* cause the calculations to be computationally costly. *Irrelevant features* increase the computational cost and may hinder the convergence by pointing out false directions for the agent to follow. *Redundant features* increase the computational cost and may hinder the convergence too (e.g. if they are not strongly connected on the lowest observation level). *Constant zero features*, which describe aspects that never occur in practice, also waste the computational energy. It may also happen that a feature is useful for the model and helps at decision points, but the totality of the features together is not optimal (e.g. too much redundancy).

We propose an automatic knowledge optimization mechanism that helps transforming the set of observable features and the knowledge itself into a format that is better in means of convergence and, at the same time, is computationally cheaper to handle. The optimization mechanism is based on three submechanisms:

- Feature extraction: a mechanism to re-organize features so that they get more useful for learning. By usefulness we mean that they depict more informative properties (e.g. combination of simple features); or, the feature set as a whole is less redundant (each feature refers to a different aspect of the world instead of co-referring to the same aspect many times).
- Feature removal: features that are of no help are removed.
- Feature generation: a mechanism to create completely new features. The difference between feature extraction and feature generation is that the former works from the current feature set while the latter is independent of that.

Another important element of the optimization mechanism is the triggering: when to start the optimization, in order to make sense and not harm.

4.1 Feature extraction

Feature extraction is rather a purpose than a tool, it is a common name for several, often really unsimilar methods that all serve the same goal: to make the set of features better. Feature extraction typically analyzes the original features by taking a representative sample from the data and analyzing the feature matrix of this sample. In the feature matrix, each row refers to one line of data, and each column refers to one feature (e.g. $m(i, j)$ means the j^{th} feature of the i^{th} data sample).

Traditional feature extraction techniques cannot be applied to our model directly, because we do not collect data samples.

On the other hand, we do collect a more sophisticated extract from the world states: the state transition matrix A. We decided to use A as the basis of feature extraction.

Traditional feature extraction techniques include autocorrelation analysis, Singular value Decomposition, Principal Com-

Algorithm 1 Feature extraction. The optimization algorithm first identifies the correlation groups, and then replaces the original features in the group with a single combined one. This means, that both the feature calculation logic (which determines what are the feature values for a given worlds state), both vector b and matrix A need to be updated.

```
1 Find correlation groups in A. Greedy algorithm:
 1.1 For each row i in A:
  1.1.1 If i is already assigned to a correlation group, then continue with the next i.
  1.1.2 Create a new correlation group Gi. Put i into Gi.
  1.1.3 For all rows j > i:
   If j is already assigned to a group, then continue with the next j.
   If (for each row r in Gi:   corr(A[r],A[j]) > CORRELATION_LIMIT), then put j into Gi.
  1.1.4 If the cardinality of Gi  is below 2 then throw Gi away.
2 For each correlation group, update the feature calculation logic.
 2.1 For each rowId in the group: remove the corresponding feature from
  the feature calculator logic.
 2.2 Add a new feature which is the combination of the original (removed) features.
  We used the average function as the combination operator.
  f[new] = g(f[rowId1], f[rowId2],.. f[rowIdn])
3 For each correlation group, update the b vector.
 3.1 For each rowId in the group: remove the corresponding value from b.
 3.2 Add a new value which is the combination of the original (removed) values.
  We used the average function as the combination operator.
  b[new] = g(b[rowId1], b[rowId2],.. b[rowIdn])
4 For each correlation group, update the A matrix.
 4.1 For each rowId in the group: remove the corresponding row from A.
 4.2 Add a new row which is the combination of the original (removed) rows.
  We used the average function as the combination operator.
  A[new] = g(A[rowId1], A[rowId2],.. A[rowIdn])
 4.3 For each rowed in the group: remove the column from A.
 4.4 Add a new column which is the combination of the removed columns. For each row i:
  A[i,new] = g(A[i,rowId1], A[i,rowId2],.. A[i,rowIdn])
```

ponent Analysis and other techniques that are good, but have certain computational cost.

We decided to use a simpler, cheaper, and more intuitive methodology: the filtering of correlation groups.

The algorithm is based on the observation that in a well trained agent, the rows of the matrix A seem often to be very similar to each other. We used correlation to describe this similarity:

$$\mathrm{corr}(A[i],A[j]) = \frac{\mathrm{cov}(A[i],A[j])}{\sigma_{A[i]}\sigma_{A[j]}} = \frac{E[(A[i]-\mu_{A[i]})(A[j]-\mu_{A[j]})]}{\sigma_{A[i]}\sigma_{A[j]}}$$

where $A[i]$ means the i^{th} row of A, cov is the covariance operator, E is the expected value operator, μ denotes the expected value and σ denotes the standard deviation.

Correlation group in matrix A is a set of rows whose pairwise correlation is very high.

The optimization algorithm has $O(n^3)$ computational cost and $O(n^2)$ space cost where n is the number of original features. Please note that the matrix A contains n^2 values so it is impossible to process its contents in less than $O(n^2)$ times. The most significant factor in the computational complexity is the 1^{st} step.

The feature extraction influences the feature count, hence, and the size of A and b. Each correlation group adds a new feature and removes as many old features as the cardinality of group. If FC denotes the feature count, $|x|$ means the cardinality of a group, then the feature count can be calculated as follows:

$$FC_{new} = FC_{old} + |G| - \sum |G_i|$$

4.2 Feature removal

Please note that both the horizontal and the vertical usefulness needs to be taken into account. Otherwise, we would mistakenly remove features that occur in reward states, so are on the end side of an action. For these features, the row corresponding to them will remain unfilled forever, but the corresponding column is in active use.

4.3 Feature generation

Our work on systematic feature generation is not detailed in this paper. We used template generation and a shifting/rotation/translation/reflection. Some experiment also included the usage of an extended Dynamic Time Warping (DTW) model.

4.4 Triggering

The knowledge optimization needs a stable starting point, in other words, a stable knowledge to optimize. If the optimization is executed on a half-developed knowledge, it may not detect

Algorithm 2 Feature removal. We propose a simple feature removal algorithm which removes those features that are never used.

```
1 For each row r in A: determine if it's unnecessary
 1.1 Count sum of values in the rth column (sum_vertical) and in the rth row (sum_horizontal).
 1.2 If sum_vertical < USEFULNESS_LIMIT and sum_horizontal < USEFULNESS_LIMIT
  then mark r as unnecessary.
2 For each unnecessary row r: remove the corresponding row and column.
```

important redundancies or may detect false correlation groups (due to the sparsity or unbalanced contents of A).

We propose to use a shifting window based self-evaluation function in order to trigger optimization cycles. When the agent feels that within the last few rounds its performance was stable, it starts an optimization cycle.

5 Collective learning

Adaptivity is often required in distributed situations, where autonomous building blocks of the system need to find their optimum locally, without central help or control. Collective adaptive systems make use of the connectivity of individual blocks – through communication – in order to help converging faster or globally better. In typical collective-adaptive system, there is a common world model, shared by the members of the society. The presence of an explicit world model facilitates the collective behavior, as it provides a common and well-defined understanding of the world and of possible strategies. However, in our approach, there is no clearly defined common world model to share – each agent builds and optimizes its knowledge autonomously.

We defined a collective self-evaluation and expertise sharing mechanism for the society of knowledge-poor adaptive systems [9]. Knowledge sharing is realized as multitude of pair-wise shares, in a self-organizing manner, without central control and without stashing common knowledge centrally. The knowledge sharing mechanism includes the following:

- Knowledge import model: a mechanism to integrate external knowledge into the one's own knowledge base.
- Self-evaluation mechanism: a metric for the agent to evaluate the goodness of its knowledge in the actual environment.
- Sharing and acceptance mechanism: a mechanism that initiates and controls knowledge sharing/acceptance. Participants of the share – donor and acceptor – are autonomous elements, so it is their free decision what, when and with whom to share or accept.

For more details please refer to [9].

6 Discussion

This section discusses consequences, generalization directions and limitations of the pervious models.

6.1 Descriptive properties

The descriptive properties of the model were partially covered in Section 3: the model is suitable for problems where the state of the environment changes from time to time and the actor is able to perform actions picked from a finite – or, when approximation is acceptable, then even infinite – set of possible actions. Opponents and environmental rules are not directly modeled within the agent's knowledge, so the model is generally applicable for multi-actor situations. The learning process is knowledge-poor, so, except for the initial features, the agent does not need pre-injected knowledge. Learning happens naturally, during the agent's normal activity; which is in contrast with today's popular adaptive system approaches, where the learning phase precedes the phase of normal operation.

6.2 Interesting phenomena

A very important factor influencing the learnability of a problem is how well the actual problem case is presented, hence, the **behavior of the opponent**. Opponents may significantly influence the convergence of the learning process, especially for up-experienced agents. When playing against a dummy (e.g. random) opponent, the agent easily spends significant amount of time in irrelevant sections of the problem space – as none of the players knows how to become successful. In case of a strong opponent the agent learns fast what to avoid and, probably, also what to do to win (see section 6). It is an interesting question whether the strongest opponent is the best, or it is more optimal to learn against consequent but imperfect players. The advantage of an imperfect opponent is that it leaves space for the agent to learn how to correct errors and how to make use of the other's mistakes. We believe, and tests indicate, that the variety of opponent styles leads to the best kind of know-ledge for static and dynamic cases, respectively.

The agent's own strategy may also hinder the emergence of good knowledge. Too fast convergence in the knowledge may be dangerous because it develops over-specialized strategies that work well against the current opponent but may not help if the environment changes. To avoid overspecialization, the agent may choose **prevention strategies**, such as picking second-best directions. Such a strategy leads to a better coverage of the problem space, which may be suboptimal in the current game, but could help against future opponents with yet unknown strategies.

The collective layer of the learning may become sensitive to unbalances in the self-interpretation of the agent. An extreme case is when the agent's self-confidence is so low that its original knowledge gets zero weight at knowledge import, meaning that the imported knowledge replaces the receptor's own knowledge (**suppressive import**). In this case the receptor becomes the donor's equal copy or **clone**. This may be desirable if the imported knowledge is guaranteed to be of high value while the knowledge of the receptor is clearly non-performing. However, suppressive import may easily lead to a drastic drop in the **population's diversity** which may become dangerous when the environment changes again.

7 Evaluation

Models were evaluated through simulation. This section includes the most important results about the standalone learning, adaptation ability, and the collective dimension.

The *standalone model* was evaluated along two lines. First we wanted to see, how efficient is the on-line learning ability in the connect-5 world, with no initial experience, trained against opponents with different strengths. In this setting, we also measured, how the self-evaluation mechanism performs compared to an objective evaluator. After this we examined how trained agents react to drastic environmental changes.

7.1 Adaptation with no initial knowledge

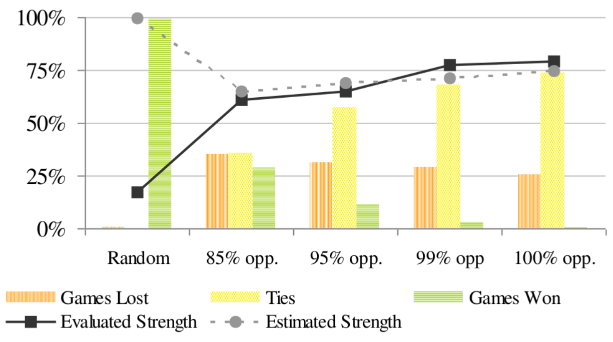

Fig. 1. Learning characteristics and self-evaluation vs. opponent style.

The experiment consisted of an adaptation phase and an objective evaluation phase.

1 First, the untrained agent plays 350 games against a fixed-algorithm opponent, and uses its learning mechanism to adapt. Win/lost/tie statistics were also collected here. We evaluated five identical agents, each playing with a different opponent, namely: one random player; and the four opponents with mathematically optimal strategies but with some – 15%, 10%, 1% and 0% – chance of making an error (failing to choose the perfect action).

2 In the evaluation phase, learning was switched off in order to get an unbiased picture about the knowledge of each agent. Agents were allowed to use their existing knowledge, and were evaluated by playing 100 games against the "perfect" (0% failure rate) opponent, as an absolute measure. Their preliminary self-evaluation (based on the training phase) was compared to the actual measured strength. (Strength is defined as the percentage of non-lost games.)

Columns in Fig. 1 visualize the outcome of the adaptation phase, while the curves refer to the self-evaluated and objectively measured strength. Training results show that the number of games won by the agent falls as opponents get stronger. Surprisingly, the number of lost games does not increase with stronger opponents; instead, games tend to end more often with a tie. Evaluation results indicate that the real gameplay strength is higher for agents trained against stronger opponents. Please note, that although the agent trained with the random player holds the lowest strength, it could also fray out a tie in 17 percent of the games against the strongest opponent. The difference between the self-estimated strength and the actual strength is unexpectedly small, expect for the divergent (random) training environemnt.

7.2 Adaptation potential to different changes

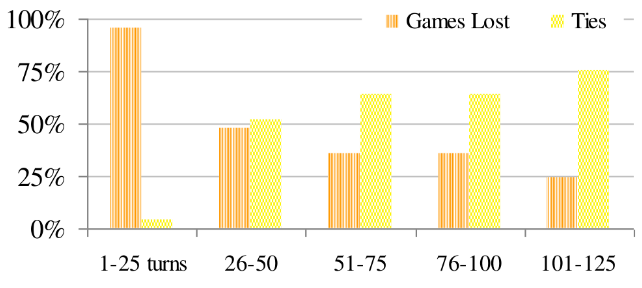

Fig. 2. Adaptation to changing rules (Connect-4 to Connect-5).

The second experiment examines the level of adaptivity to world changes. We used a trained agent, which had a training session of 50 connect-4 games (a game with the same rules as connect-5, except that the combination of four is enough for the victory). Then, the agent had to play connect-5 against the strongest opponent, without any notification or adjustment regarding the rule change. Fig. 2 shows that in the first 25 games the agent had serious problems using the experience gathered earlier, resulting in a defeat rate of almost 96 percent. Although, after 50 games it could defend with 50 percent accuracy, and after 125 games, it reached almost the same strength level, as in the previous on-line learning test. Tests with other rule change schemes (C-5 to C-4, no competitiveness) brought similar results.

7.3 Knowledge optimization

Fig. 3 shows the difference between the default and the optimized knowledge. First, an agent with the default knowledge was trained against a perfect opponent in 40 rounds (K1 agent, 54 features). Then, its knowledge was declared to be stable, and, correlation groups and new complex features were identified. We trained an independent agent using the previously identified optimized features against the perfect opponent, again, in 40 rounds (K2 agent, 28 features). Finally, both K1 and K2 got evaluated with an offline opponent of 0.5 failure rate. It's clearly visible that the optimized features lead to faster convergence, win more games than K2 and, furthermore, managed to

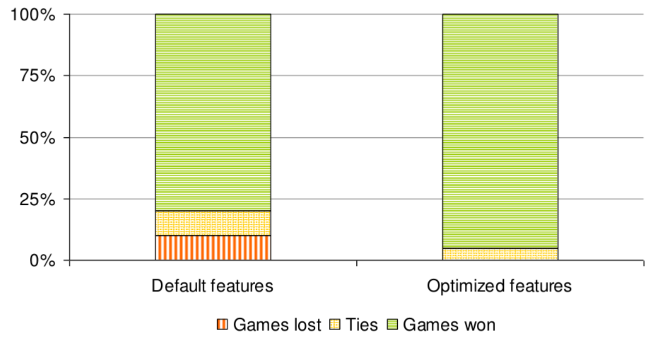

Fig. 3. Evaluation of the optimized knowledge.

avoid lost situations completely.

7.4 Collective learning

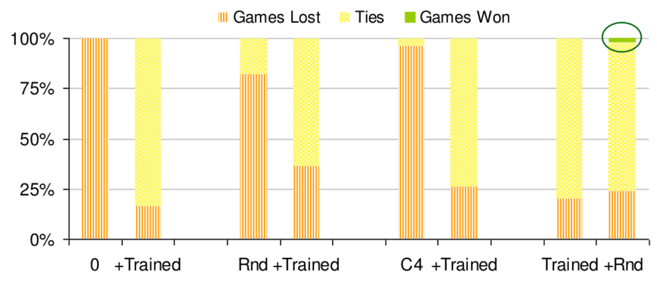

Fig. 4. Collective effects (evaluated against the 100% opponent)

The collective dimension was evaluated along various aspects, here we present one. Differently trained agents got knowledge injections, and then, got evaluated off-line. Listed combinations are: empty (untrained) acceptor + best trained donor (trained with the strongest opponent), randomly trained acceptor + best trained donor, Connect-4 trained receptor + best trained donor, and best trained acceptor + random donor. Fig. 4 shows that knowledge injection had positive effects in all cases. This is not surpising in the first three cases when the injected knowledge was clrealy more accurate than the agent's own. In the last case, a good agent received "worse" knowledge, and still, this helped it to gain winning which was out of question beforehand (however, general strength dropped slightly). This effect can be explained with the nonlinearity of the problem space.

8 Summary

We described an approach for an open learning, collective-adaptive system where learning and adaptation emerge from simple steps during the system's normal operation, even amongst drastically changing environmental conditions. Dynamic features and the lack of mandatory – and too explicit – semantics bring real openness to the model. The model is also equipped with a cooperative layer where learners can share knowledge and help each other to converge faster. The model was evaluated via simulation for a large number of scenarios, and the practical results confirmed its theoretic advantages.

Our model extends the state of the art in the followings: (a) applies temporal difference (LSTD) learning for the problem of adaptive systems where requirements change dynamically over time (b) introduces the possibility of an on-the-fly, automatic knowledge optimization through feature extraction and feature removals (c) brings TD learning into a collective dimension.

References

1 **Baxter J, Weaver A**, *TDLeaf(lambda): Combining temporal difference learning with game-tree search*, Proceedings of the 9th Australian Conference on Neural Networks (1998), 39-43.

2 **Bencomo N et al.**, *Genie: supporting the model driven development of reflective, component-based adaptive systems*, Proc. of the 30th International Conference on Software Engineering (2008), 811-814.

3 **Benkő B K, Brgulja N, Höfig E, Kusber R**, *Adaptive services in a distributed environment*, Proc. of 8th International Workshop on Applications and Services in Wireless Networks, posted on 2008, DOI 10.1109/ASWN.2008.8, (to appear in print).

4 **Bradtke S J, Barto A G**, *Linear least-squares algorithms for temporal difference learning*, Machine Learning 1996/22, posted on 1996, 33-57, DOI 10.1007/BF00114723, (to appear in print).

5 **Brun Y, Di Marzo Serugendo G, Gacek C, Giese H, Kienle H, Litoiu M, Müller H, Pezze M, Shaw M**, *Engineering self-adaptive systems through feedback loops*, Springer, Heidelberg, 2009, LNCS 5525/2009.

6 **De Jong K**, *Evolutionary computation: a unified approach*, In Proc. of the 2008 GECCO conference companion on Genetic and evolutionary computation, posted on 2008, 2245-2258, DOI 10.1145/1570256.1570404, (to appear in print).

7 **Goldsby H J**, *Goal-based modeling of dynamically adaptive system requirements*, Proc. of 15th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS), posted on 2008, DOI 10.1109/ECBS.2008.22, (to appear in print).

8 **Harel D, Marelly R**, *Come, let's play: scenario-based programming using LSCs and the play-engine.*, Springer, Heidelberg, 2005.

9 **Lányi D, Benkő B K**, *A Nontraditional Approach for a Highly Interactive Collective-Adaptive System*, In Proc. of EMERGING 2010. (in press).

10 **Kolter J Z, Ng A Y**, *Regularization and feature selection in least-squares temporal difference learning*, Proc. of the 26th Annual International Conference on Machine Learning (ICML 09) **382** (2009), 521-528, DOI 10.1145/1553374.1553442.

11 **Sutton R S**, *Learning to predict by the methods of temporal differences*, Machine Learning 1988/3, posted on 1988, 9-44, DOI 10.1007/BF00115009, (to appear in print).

12 **Sutton R S, Barto A G**, *Reinforcement learning: an introduction*, The MIT Press, 1998.