Modular self-organization for a long-living autonomous agent

Bruno SCHERRER
scherrer@loria.fr
LORIA, BP 239
54506 Vandceuvre-les-Nancy
France

Abstract

The aim of this paper is to provide a sound framework for addressing a difficult problem: the automatic construction of an autonomous agent's modular architecture. We briefly present two apparently uncorrelated frameworks: Autonomous planning through Markov Decision Processes and Kernel Clustering. Our fundamental idea is that the former addresses autonomy whereas the latter allows to tackle self-organizing issues. Relying on both frameworks, we show that modular self-organization can be formalized as a clustering problem in the space of MDPs. We derive a modular self-organizing algorithm in which an autonomous agent learns to efficiently spread *n* planning problems over m initially blank modules with *rn < n.*

Introduction

This paper addresses the problem of building a long-living autonomous agent; by *long-living,* we mean that this agent has a large number of relatively complex and varying tasks to perform. Biology suggests some ideas about the way animals deal with a variety of tasks: brains are made of specialized and complementary areas/modules; skills arc spread over modules. On the one hand, distributing functions and representations has immediate advantages: parallel processing implies reaction speed-up; a relative independence between modules gives more robustness. Both properties might clearly increase the agent's efficiency. On the other hand, the fact of distributing a system raises a fundamental issue: how does the organization process of the modules happen during the life-time ?

There has been much research about the design of modular intelligent architectures (e.g. [Theocharous *et al,* 2000J [Hauskrechtef a/., 1998] [Kaelbling, 1993]). It is nevertheless very often the (human) designer who decides the way modules are connected to each other and how they behave with respect to the others. Few works study the construction of these modules. To our knowledge, there are no effective works about modular self-organisation except for reactive tasks (stimulus-response associations) (e.g. [Jacobs *et al.,* 1991] [Digney, 1996]).

This paper proposes an algorithm by which the organization of an agent in functional modules is automatically computed. The most significant aspect of our work is that the number *m* of modules is fewer than the number *n* of tasks to be performed. Therefore, the approach we propose involves a high-level clustering process, in which the *n* tasks need to be "properly" spread over the m modules.

Section 1 introduces what we consider as the theoretical foundation for modelling a mono-task autonomous agent: Markov Decision Processes. Section 2 presents the Kernel Clustering approach: we consider this approach as a theoretical basis for addressing self-organization. Finally, Section 3 combines both domains in order to propose a modular self-organizing algorithm.

1 Modelling A Mono-Task Autonomous Agent

Markov Decision processes [Puterman, 1994] provide the theoretical foundations of challenging problems such as planning under uncertainty and reinforcement learning [Sutton and Barto, 1998]. They stand for a fundamental model for sequential decision making and they have been applied to many real worls problems [Sutton, 1997]. This section describes this formalism and presents a general scheme for approaching difficult problems (that is problems in large domains).

A Markov Decision Process (MDP) is a controlled stochastic process satisfying the Markov property with rewards (numerical values) assigned to state-control pairs. Formally, an MDP is a four-tuple (S, *A,* T, *R)* where *S* is the *state space, A* is the *action space, T* is the *transition function* and R. is the *reward function. T* is the state-transition probability distribution conditioned by the control:

$$T(s,a,s') \stackrel{def}{=} \Pr(s_{t+1} = s' | s_t = s, a_t = a) \qquad (0$$

$R(s,a) \in \mathbb{R}$ is the instantaneous reward for taking action $a \in A$ in state *S.*

The usual *MDP problem* consists in finding an *optimal policy,* that is a mapping $\pi : S \rightarrow A$ from states to actions, that maximises the following performance criterion, also called value function of policy *TT:*

$$V^{\pi}(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t . R(s_t, \pi(s_t)) | s_0 = s\right] \qquad (2)$$

It is shown that there exists a unique optimal value function V*; once V^* is computed, an optimal policy can immediately be derived (e.g. see [Puterman, 1994]).

In brief, solving an MDP problem amounts to computing the optimal value function. Well-known algorithms for doing so are *Value Iteration* and *Policy Iteration* (see [Puterman, 1994]). Their temporal complexity dramatically grows with the number of states [Littman *et al,* 1995], so they can only be applied in small domains.

In large domains, it is impossible to solve an MDP exactly, so one usually adresses a complexity/quality compromis through an approximation scheme. Ideally, an approximation scheme for MDPs should consist of a set of *tractable* algorithms for

- computing an approximate optimal value function
- evaluating (an upper bound of) the approximation error
- improving the quality of approximation (by reducing the approximation error) while constraining the complexity.

The first two points are the fundamental theoretical bases for sound approximation. The third one is often interpreted as a learning process and corresponds to what most Machine Learning researchers study. For convenience, we respectively call these three procedures *Appproximate(), Error ()* and *Learn().* Then, the practical use of an approximate scheme can be sketched by algorithm 1. One successively applies the

Algorithm 1 A general approximation scheme for a large state MDP

```
Input: a large state space MDP $\mathcal{M}$ and an initial approximation $\widehat{\mathcal{M}}$.
Output: a good approximate value function $\widehat{V}^*$.

while $Error(\widehat{\mathcal{M}}, \mathcal{M})$ goes on diminishing do
    $\widehat{\mathcal{M}} \leftarrow Learn(\widehat{\mathcal{M}}, \mathcal{M})$
end while
$\widehat{V}^* \leftarrow Approximate(\widehat{\mathcal{M}})$
```

Lcarn() procedure in order to minimize the approximation error; when this is done, one can compute a good approximate optimal value function.

2 Kernel Clustering

Before addressing the problem of modular self-organization, we need to present the Kernel Clustering paradigm. In [Diday, 1973], the author introduces the Kernel Clustering approach as an abstract generalization of vector quantization. Indeed, the author argues that, in general, a clustering problem is based on three elements:

- $(x_i)_{i\in I}$: a set of data points taken from a data space A"
- $\{L_1, .., L_m\}$: a set of kernels taken from a kernel space $\mathcal{L}$
- $d : X \times \mathcal{L} \rightarrow \mathbb{R}^+$: A distance measure between any data point and any kernel. The smaller the distance *d(x,* L), the more *L* is *representative* of the point *x.*

Given a set of kernels $\{L_1, .., L_m\}$, a data point x is naturally associated to its most *representative* kernel *L{x),* i.e. the one that is the closest according to distance *d:*

$$L(x) = \text{argmin}_{L\in\{L_1, \ldots, L_m\}} d(x, L) \quad (3)$$

Conversely, a set of kernels $\{L_1, .., L_m\}$ naturally induces a partition of the data set $(x_i)_{i\in I}$ into m classes $\{C_1, \ldots, C_m\}$, each class corresponding to a kernel:

$$\forall j \in (1, .., m), C_j = \{(x_i)_{i\in I}; L(x_i) = L_j\} \quad (4)$$

Given a data space, a data set, a kernel space and a distance d(), the goal of the Kernel Clustering problem is to find the set of kernels $\{L_1^*, .., L_m^*\}$ that minimizes the distortion *D* for the data set $(x_i)_{i\in I}$, which is defined as follows:

$$D = \sum_{i\in I} d(x_i, L(x_i)) = \sum_{j=1}^{m} \sum_{x\in C_j} d(x, L_j) \quad (5)$$

A general procedure for suboptimally solving this problem is known as the Dynamic Cluster algorithm [Diday, 19731, which we present in an online version in algorithm 2. It is

Algorithm 2 The on-line dynamic cluster algorithm

```
Input: A data set $(x_i)_{i\in I}$
Output: A set of kernels $\{L_1, .., L_m\}$ that optimizes the
  clustering (i.e. that minimizes the distortion)
  Initialization:
  Let $\{L_1, .., L_m\}$ be any set of kernels
  Iterations:
  while the distortion goes on diminishing do
      Randomly pick a data point $x$ from the data set
      Find the kernel the most representative kernel of $x$:
          $L \leftarrow L(x) = \text{argmin}_{L'\in\{L_1, \ldots, L_m\}} d(x, L')$
      Update $L$ so that $d(x, L)$ diminishes
  end while
```

a very intuitive process: for each piece of data x, one finds its most representative kernel *L,* and one updates *L* so that it gets even more representative of *x.* Little by little, one might expect that such a procedure will minimize the global distortion and eventually give a good clustering.

3 Modular Self-Organization

This final section shows how the Kernel Clustering paradigm can be used to formalize a modular self-organization problem in the MDP framework, the algorithmic solution of which will be given by the on-line Dynamic Cluster procedure (algorithm 2).

If one carefully compares the general learning scheme we have described in order to address a large state space MDP (algorithm 1) and the on-line Dynamic Cluster procedure (algorithm 2), one can see that the former is a specific case of the latter. More precisely, algorithm 1 solves a simple Kernel Clustering problem where

- the data space is the space of all possible MDPs and the data set is a unique task corresponding to an MDP *M*

- the kernel space is the space of all possible approximations and there is one and only one kernel: $\widehat{\mathcal{M}}$
- the distance d is the *Error()* function.

It is then straightforward to extend this simple clustering problem to a more general one (with n tasks/data points and m approximate models/kernels). Given a set of m approximate models $\{\widehat{\mathcal{M}}_1, .., \widehat{\mathcal{M}}_m\}$, an MDP $\mathcal{M}$ is naturally associated to the approximate model $\widehat{\mathcal{M}}(\mathcal{M})$ that makes the smallest error:

$$\widehat{\mathcal{M}}(\mathcal{M}) = \text{argmin}_{\widehat{\mathcal{M}}' \in \{\widehat{\mathcal{M}}_1, .., \widehat{\mathcal{M}}_m\}} Error(\widehat{\mathcal{M}}', \mathcal{M}) \quad (6)$$

As before, a set of approximate models $\{\widehat{\mathcal{M}}_1, .., \widehat{\mathcal{M}}_m\}$ naturally induces a partition of any set of n MDPs $(\mathcal{M}_i)_{1 \leq i \leq n}$ into m classes $\{C_1, .., C_m\}$, each class corresponding to an approximate model:

$$\forall j \in (1, .., m), C_j = \{(M_i)_{1 \leq i \leq n}; \widehat{\mathcal{M}}(M_i) = \widehat{\mathcal{M}}_j\} \quad (7)$$

The transpositon of the on-line Dynamic Cluster Algorithm into the MDP framework (algorithm 3) therefore allows to find a set of m approximate models that globally minimize the approximation error for n MDPs:

$$\sum_{i=1}^{n} Error(\widehat{\mathcal{M}}(\mathcal{M}_i), \mathcal{M}_i) = \sum_{j=1}^{m} \sum_{\mathcal{M} \in C_j} Error(\widehat{\mathcal{M}}(\mathcal{M}), \mathcal{M}) \quad (8)$$

In other words, it optimizes the tuning of m modules in order to efficiently solve n tasks, or, as we might say, it self-organizes the m modules in order to improve the resolution of the n tasks.

Algorithm 3 Modular Self-Organization

```
Input: A set of MDPs $(\mathcal{M}_i)_{i \in I}$
Output: A set of approximate models $(\widehat{\mathcal{M}}_1, .., \widehat{\mathcal{M}}_m)$ that
  globally minimizes the approximation error
  Initialization:
  Let $(\widehat{\mathcal{M}}_1, .., \widehat{\mathcal{M}}_m)$ be any set of approximate models
  Iterations:
  while the global approximation error goes on diminishing do
    Randomly pick a task $\mathcal{M}$ from the set of MDPs
    Find the best module for solving $\mathcal{M}$:
      $\widehat{\mathcal{M}} \leftarrow \text{argmin}_{\widehat{\mathcal{M}}' \in \{\widehat{\mathcal{M}}_1, .., \widehat{\mathcal{M}}_m\}} Error(\widehat{\mathcal{M}}', \mathcal{M})$
    $\widehat{\mathcal{M}} \leftarrow Learn(\widehat{\mathcal{M}}, \mathcal{M})$
  end while
```

Conclusion

In this paper, we have described a general scheme for addressing large state space Markov Decision Processes. We have then showed how such an approach could be extended to an interesting problem: modular self-organization. Indeed, we have formalized modular self-organization as a clustering problem in the space of MDPs. A natural algorithmic solution to this clustering problem (algorithm 3) uses an on-line version of the Dynamic Cluster algorithm (algorithm 2). Due to lack of space, we could not show any experimental evaluation; interested readers will find some in [Scherrer, 2003bl and [Scherrer, 2003a].

References

[Diday, 1973J E. Diday. The dynamic clusters method and optimization in non hierarchical-clustering. In SpringerVerlag, editor, *5th Conference on optimization technique, Lecture Notes in Computer Science 3,* pages 241-258,1973.

[Digney, 1996] B. Digney. Emergent hierarchical control structures: Learning reactive hierarchical relationships in reinforcement environments, 1996.

[Hauskrecht et al., 1998] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov Decision Processes using macro-actions. In *Uncertainty in Artificial Intelligence,* pages 220-229, 1998.

[Jacobs *et al,* 1991] R. Jacobs, M. Jordan, and A. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science,* 15:219-250, 1991.

[Kaelbling, 1993] L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *International Conference on Machine Learning,* pages 167-173, 1993.

[Littmanefa/., 1995] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95),* pages 394-402, Montreal, Quebec, Canada, 1995.

[Puterman, 1994] M. Puterman. *Markov Decision Processes.* Wiley, New York, 1994.

[Scherrer, 2003a] B. Scherrer. *Apprentissage de representation et auto-organisation modulaire pour un agent autonome.* PhD thesis, Universite Henri Poincare - Nancy 1, January 2003.

[Scherrer, 2003b] Bruno Scherrer. Modular Self-Organization for a long-living autonomous agent. Technical report, INR1A, April 2003.

[Sutton and Barto, 1998] R.S. Sutton and A.G. Barto. *Reinforcement Learning, An introduction.* BradFord Book. The MIT Press, 1998.

[Sutton, 1997] Richard S. Sutton. On the significance of markov decision processes. In *ICANN,* pages 273-282, 1997.

lTheocharous et al.,2000] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning and planning with hierarchical stochastic models for robot navigation. In *ICML 2000 Workshop on Machine Learning of Spatial Knowledge,* Stanford University, July 2000.