HAL
open science

Auto-organisation modulaire d'une architecture intelligente

Bruno Scherrer

To cite this version:

Bruno Scherrer. Auto-organisation modulaire d'une architecture intelligente. Valgo numéro 01-02, La revue en ligne de l'Association des Connexionnistes en THèse, Association des Connexionnistes en THèse, Oct 2001, Montélimar, France, 8 p. inria-00099399

HAL Id: inria-00099399

https://inria.hal.science/inria-00099399

Submitted on 26 Sep 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Auto-organisation modulaire d'une architecture intelligente

Bruno SCHERRER
scherrer@loria.fr

LORIA-INRIA
Campus Scientifique BP 239
F54506 Vandoeuvre-les-Nancy
FRANCE

Résumé

Ce papier présente notre démarche sur l'étude de l'auto-organisation modulaire d'un système de décision complètement générique. Dans un premier temps, nous décrivons l'approche de l'apprentissage par renforcement. Nous montrons de quelle façon le cadre formel des processus décisionnels de Markov (PDM) permet de définir précisément la notion de spécialisation modulaire. Ensuite, nous dérivons une abstraction des principes généraux d'auto-organisation de nombreux algorithmes connexionnistes de classification. Nous adaptons ces principes au problème de l'émergence de modules fonctionnels dans un système s'appuyant sur les PDM : un agent amené à résoudre une série de tâches va, au cours du temps, voir différents modules le constituant se spécialiser et former un tout cohérent et efficace. Nous expliquons et justifions notre démarche et dressons des objectifs à court terme.

Introduction

Un système de décision tel que le cerveau biologique est organisé en aires/modules spécialisés et complémentaires. Nos recherches tendent à s'inspirer de cette idée d'organisation modulaire pour élaborer une architecture intelligente et autonome de type agent robotique ; nous nous inscrivons en ce sens dans le domaine de l'intelligence artificielle (IA). D'un point de vue opérationnel la distribution des représentations et des fonctions a des avantages immédiats en terme de vitesse de traitement (parallélisme intrinsèque) et de robustesse (indépendance relative des modules). Cependant elle pose deux problèmes cruciaux : l'organisation de ces modules au cours du temps (au cours de l'ontogénèse pour l'être vivant) et l'intégration/coopération cohérente des réponses des différents modules lors des traitements décisionnels.

Beaucoup de travaux en IA abordent le comportement autonome sous l'angle modulaire, par exemple [19][6][1][8]. C'est cependant le concepteur (humain) qui décide de la façon dont les modules sont définis et interagissent les uns avec les autres. Rares sont ceux qui s'intéressent à l'émergence de ces modules. En dehors de tâches réactives (associations stimulus-réponse)[7][9][4], il n'y a à notre connaissance pas de travaux concluants sur l'auto-organisation de modules résolvant des tâches complexes.

Sur quoi se baser pour espérer une organisation automatique ? Dans [17], les auteurs montrent que chez le rat, des stimuli visuels peuvent être "branchés" dans une aire auditive. Lorsqu'on procède à cette manipulation, on observe que l'aire habituellement utilisée pour l'ouïe peut être utilisée pour la vision. Cette expérience suggère que le mécanisme d'apprentissage est plus ou

moins le même pour la plupart des aires fonctionnelles. De plus, ce serait essentiellement l'information présentée qui structurerait et génèrerait une aire. Nous nous appuierons dans tout ce qui suit sur l'idée que c'est l'information et en particulier les tâches à résoudre qui permettent de faire apparaître des modules fonctionnels.

Nous présentons dans ce paragraphe le plan du reste du papier. Le modèle que nous utilisons pour décrire les modules fonctionnels est un peu plus complexe qu'une simple association/fonction : Il s'appuie sur la théorie de l'apprentissage par renforcement. Nous commencerons donc par en décrire brièvement le modèle de base, les processus décisionnels de Markov (PDM). Nous introduirons la notion de spécialisation modulaire qui montrera comment une tâche influence la structure d'un module. A la lumière d'un certain nombre d'algorithmes connexionnistes de classification, nous dresserons ensuite une abstraction de ce qu'on peut communément appeler l'auto-organisation. Nous l'appliquerons à un système composé d'un ensemble de modules s'appuyant sur l'apprentissage par renforcement.

1 Processus décisionnels de Markov

Nous commençons par présenter dans cette section le formalisme de l'apprentissage par renforcement. Celui-ci permet de modéliser de façon complètement générique le problème de la décision que nous considérons comme incontournable dans un système intelligent. Le fonctionnement des modules de notre architecture s'appuiera sur ce formalisme.

1.1 Formalisme

Un processus décisionnel de Markov[15](PDM) est un quadruplet $< S,A,T,R >$ qui décrit un système décisionnel. S est l'ensemble des états du système. A est l'ensemble des actions/décisions que peut prendre le système. T, souvent désigné comme la matrice de transition du système, est une fonction de $S \times A \times S$ dans $[0,1]$ qui donne la probabilité que le système passe d'un état à un autre lorsqu'une action est effectuée. C'est dans T que se trouve la majeure partie de l'information sur le système. A chaque instant le système est dans un état, choisit une action et passe dans un nouvel état. Finalement R est une fonction qui à chaque état associe une valeur, appelée récompense. La récompense permet de signifier quels états sont à rechercher (récompense positive) ou à éviter (récompense négative). Comme brève illustration, ce genre de formalisme permet de décrire et de résoudre des problèmes de navigation robotique dans des environnements complexes et bruités[10], de théorie des jeux ou d'allocation de ressources[18].

Lorsqu'un PDM est défini, la théorie de la programmation dynamique montre que le comportement optimal (celui qui maximise la somme des récompenses à long terme) est d'une forme très simple : choisir une action dans chaque état sans maintenir aucune mémoire. Son calcul est directement lié à une fonction dite fonction de valeur optimale (FV). Celle-ci donne la somme maximale des récompense attendues dans le futur lorsqu'on part d'un état ; le comportement optimal consiste alors à passer par les états qui ont la plus grande valeur de FV. La FV vérifie une équation non-linéaire simple qui se calcule aisément [1] à l'aide de l'algorithme *Value Iteration*[15] par exemple.

A titre illustratif, prenons le problème très simple d'un agent devant trouver son chemin dans un environnement complexe. L'environnement (figure 1) est discrétisé en zones régulières.

1. Quand S est discret, fini et de taille raisonnable.

L'agent peut décider de bouger dans les 8 directions cardinales mais le résultat de ses commandes est légèrement bruité. Deux zones de l'environnement, une en haut à droite, et une en bas à gauche, sont successivement désignés comme buts à atteindre. Les murs doivent être évités. La fonction récompense vaut -1 dans les murs, +1 dans la zone but, et 0 partout ailleurs. Les figures 2 et 3 montrent les fonctions de valeur calculées pour chacun des deux buts.

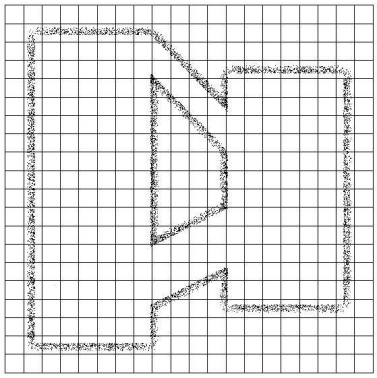

FIG. 1 – *L'environnement*

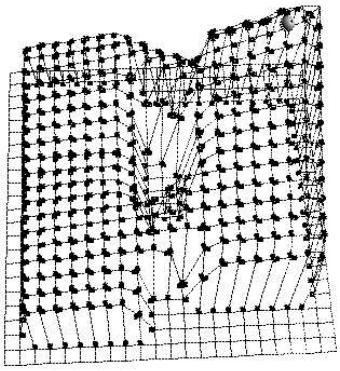

FIG. 2 – *FV pour but 1*

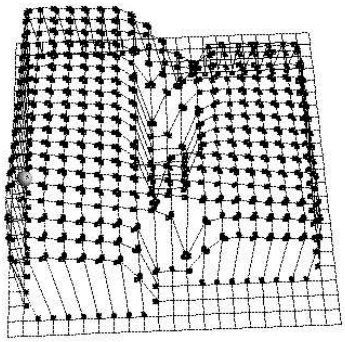

FIG. 3 – *FV pour but 2*

1.2 Spécialisation

Bien souvent, il est fastidieux voire impossible de décrire l'ensemble des états et des transitions pour un système complexe. Quand bien même on arrive à trouver une représentation assez compacte (à l'aide de réseaux bayésiens[3] par exemple), le calcul de la politique optimale peut être plutôt complexe. Nous souhaitons défendre ici l'idée qu'une grande part de l'intelligence réside dans le choix de la représentation (voir [16] pour une discussion sur le sujet). Ainsi nous concentrons nos efforts sur la façon dont nous représentons les états du système et nous utilisons les algorithmes de résolution standards des PDM.

Plus précisément, le procédé que nous proposons d'exploiter consiste à ajouter une contrainte pratique : nous utilisons des PDM dont le nombre d'états est contraint (borné ou prédéfini). S'il y a un nombre très grand (voire infini) d'états réels du système, nous agrégeons ces états en macro-états et nous considérons le PDM qui approxime le véritable PDM en raisonnant sur ces macro-états. Lorsque l'espace d'état initial est continu, cela revient à effectuer une discrétisation.

De récents travaux[13][14] se sont intéressés à la question de construire une discrétisation adaptée à des problèmes de décision dans des espaces continus. Bien que ces travaux concernent à l'origine le contrôle déterministe dans un espace continu, ils s'appliquent quasi-directement aux PDM. En effet cet ensemble de problèmes de contrôle ont en commun de se ramener au calcul d'une fonction de valeur optimale telle que nous l'avons décrite plus tôt.

Au fond l'approche ressemble énormément à ce qu'on appelle communément l'algorithmique paresseuse. On part d'une discrétisation grossière et régulière. On fait le calcul de la fonction de valeur approchée avec ce MDP grossier et on détecte (selon différents critères) les endroits où il serait bon de raffiner la précision. On divise les zones d'intérêts en sous-zones plus précises et on relance le calcul de la fonction de valeur avec le nouveau MDP ainsi défini. De fil en aiguille, on obtient une discrétisation à résolution variable qui est adaptée au calcul d'une fonction de valeur.

Plusieurs critères sont proposés et comparés dans [13]. Dans un article plus récent[14] les auteurs dérivent un critère qui permet assurément d'améliorer le calcul du comportement optimal pour le système. Si l'on fait un compromis entre la simplicité du critère et l'efficacité de l'approximation, la non-linéarité de la fonction de valeur apparaît comme un critère qui donne de très bons résultats. En d'autres termes, cela revient à mettre des ressources là où il se passe le plus de choses (là ou la fonction de valeur est la plus irrégulière). Une illustration de la définition adaptée des états est présentée ci-après (il s'agit du problème typique "Car on the Hill" ; voir [18] pour une description) : on voit la fonction de valeur (figure 4) et la discrétisation adaptée (figure 5).

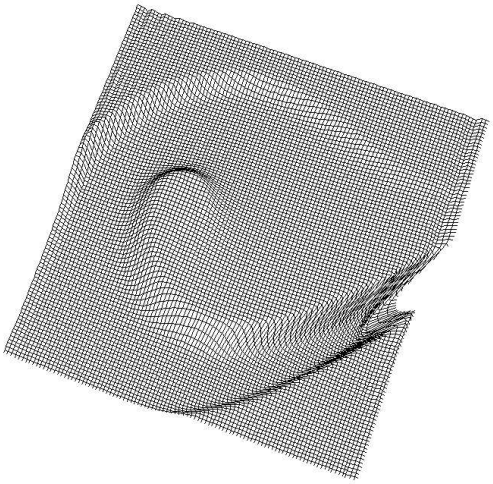

FIG. 4 – *Fonction de valeur*

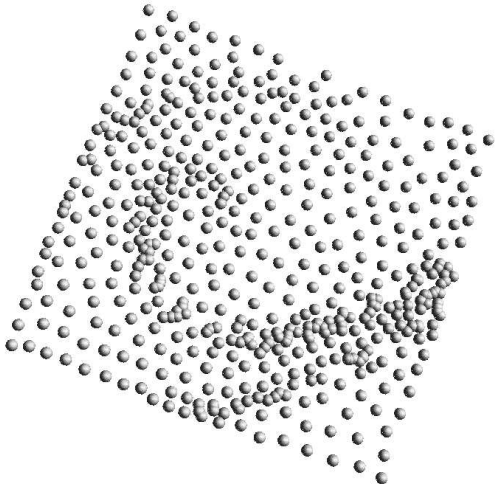

FIG. 5 – *Discrétisation adaptée*

En bref, supposons qu'on ait un PDM dont on ne peut décrire tous les états. On commencera par prendre un ensemble d'états agrégés de façon aléatoire. On évaluera la fonction de transition pour cet ensemble d'état. Nous pourrons alors, étant donné une tâche (c'est-à-dire une fonction récompense) calculer la fonction de valeur approchée ; en utilisant le critère de non-linéarité on pourra à la fois déterminer les zones de l'espace où la fonction de valeur mérite d'être discrétisée de façon plus fine mais aussi là où elle peut être plus grossière. En d'autres termes nous faisons respectivement de la spécialisation et de la généralisation. Ainsi, on pourra maintenir le nombre d'états du PDM dans un certain intervalle conformément à la contrainte que nous avons énoncée plus haut. A chaque fois que l'on rajoutera ou enlèvera des états du PDM, nous devrons réévaluer la matrice de transition T du PDM. La mise à jour des états d'un PDM impliquera par exemple d'explorer le monde là où il y a eu des changements de représentation.

Remarquons que le procédé décrit ci-dessus est directement dépendant de la fonction récompense, et donc d'une tâche particulière. Si nous avons un seul monde et disons 2 tâches différentes, il est fort probable que nous arriverons à 2 discrétisations relativement différentes. Ainsi, on peut voir l'adaptation de l'espace d'état comme un processus de spécialisation sur une tâche donnée. Notons de plus que ce procédé de spécialisation est itératif et donc progressif. Nous réutiliserons cette notion de spécialisation et son caractère progressif dans la section 3.

2 Processus d'auto-organisation

Les systèmes qui nous intéressent, ancrés dans un monde réel, reçoivent une quantité et une diversité phénoménale d'informations. C'est l'objet des statistiques et de la théorie des distributions que de répresenter un phénomène complexe et possiblement aléatoire sous une forme plus concise et donc exploitable. Face à une telle quantité d'événements, qui suivent en général une distribution particulière, il existe des méthodes dites de classification qui permettent de réduire la complexité du phénomène à un petit ensemble de classes représentatives. Il est commun de considérer que l'obtention de ce genre de classification procède d'une auto-organisation et c'est cette notion que nous allons illustrer et généraliser.

2.1 L'algorithme des centre moyens

Soit un corpus de données appartenant toutes à un même espace euclidien. L'algorithme des centres moyens (ou K-means [12]) permet de partitionner ces données en K classes (K fixé a priori). Chaque classe est représentée par un point dans l'espace appelé communément prototype. Le nom de cet algorithme vient du fait que, pendant son déroulement, chaque prototype est le centre de gravité des données qu'il regroupe dans la classe correspondante.

Nous décrivons à présent de façon plus formelle une version *en-ligne* de cet algorithme. On a un corpus de N points et K classes a priori.

- Initialisation : les prototypes de chacune des classes sont initialisés aléatoirement dans l'espace. On associe à chaque prototype j une masse m_j nulle.
- Itérations :
  - Pour les N points i du corpus
    - On calcule la distance au carré de i par rapport à chacun des prototypes. Le prototype le plus proche j_0 est désigné comme vainqueur : $j_0 = \text{argmin}_{1 \leq j \leq K} \|i - j\|^2$
    - La masse du prototype vainqueur est augmentée : $m_{j_0} \leftarrow m_{j_0} + 1$
    - Le prototype vainqueur est rapproché du point i : $\overrightarrow{ij_0} \leftarrow (1 - \frac{1}{m_{j_0}})\overrightarrow{ij_0}$

La figure 6 illustre une classification effectuée sur 3 gaussiennes à l'aide de cet algorithme.

2.2 Variantes

Il existe un certain nombre d'algorithmes connexionnistes qui peuvent être vus comme des variantes de cet algorithme. La plupart des algorithmes dont nous parlons ici sont présentés et comparés dans [2]. Dans les cartes auto-organisatrices de Kohonen, les prototypes sont associés aux points d'une grille (généralement 2D) ; cette association est telle que les points voisins sur la grille correspondent à des prototypes voisins dans l'espace d'entrée. On parle alors de cartographie. Une version incrémentale (qui rajoute dynamiquement des rangées de prototypes) appelée Growing Grid se spécialise là où la distribution est la moins bien représentée et donne des résultats qualitativement meilleurs. Le Gaz Neural construit automatiquement une topologie adaptée aux données. Il existe également une version dynamique du Gaz Neural (Growing Neural Gas) dont la dernière version[5] est capable de s'adapter à des distributions qui varient dans le temps. Nous renvoyons le lecteur aux références sus-citées pour une information plus détaillée. Le but est simplement ici de signaler que, dans le même esprit que l'algorithme des K-means, ces algorithmes vont organiser un ensemble de prototype sur une distribution complexe. La figure 7 illustre sur l'exemple des gaussiennes la classification effectuée par les cartes auto-organisatrices de Kohonen.

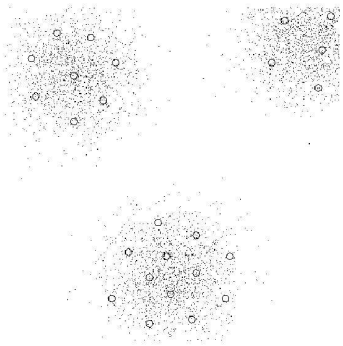

FIG. 6 – *Centres moyens*

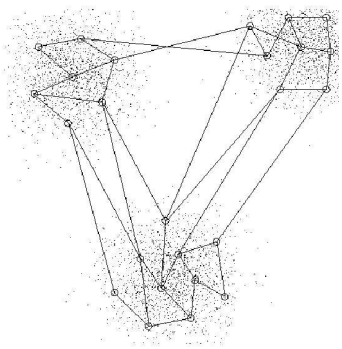

FIG. 7 – *Kohonen*

2.3 Abstraction de l'auto-organisation

Nous avons évoqué un certain nombre d'algorithmes de classification et l'une des qualités qui leur est reconnue est d'engendrer une organisation de manière automatique (on dit encore non supervisée). Il y a auto-organisation dans la mesure où il y a un certain nombre de modules/prototypes/constituants du système qui n'ont *a priori* aucune signification et qui finissent par en prendre une. L'ensemble est au départ non-organisé et le devient progressivement. Nous formulons à présent une abstraction de ces algorithmes, abstraction qui nous semble synthétiser ce qui engendre l'auto-organisation.

Soit un système composé d'un certain nombre de constituants. Nous postulons que nous obtiendrons une auto-organisation si nous exécutons l'algorithme itératif abstrait suivant :

- Initiation : Les constituants sont initialisés de façon aléatoire.
- Itérations :
  - Echantillon : On présente une entrée au système ;
  - Compétition : Tous les constituants proposent une réponse à l'entrée et on choisit la plus adaptée selon un crtitère objectif;
  - Apprentissage : le constituant vainqueur (et eventuellement son voisinage topologique s'il y a lieu) apprend ou se spécialise.

Au cours du temps l'apprentissage doit être de moins en moins intense. D'une manière analogue à l'algorithme des K-means, chacun des constituants doit se spécialiser de moins en moins. Il est nécessaire d'avoir un critère objectif qui permette de mesurer comparativement les réponses des constituants. Selon le type de constituant, l'apprentissage peut revêtir diverses formes : ce peut être une simple mise à jour de poids, mais pourquoi pas un procédé plus complexe telle qu'une rétropropagation au cas où les constituants sont des perceptrons.

3 Application de l'auto-organisation aux PDM

3.1 Description

Replaçons-nous à présent dans le cadre des PDM et montrons comment nous adaptons l'idée d'auto-organisation à des modules fonctionnels. Supposons que nous ayons un système qui ait à

résoudre une série de tâches dans un monde S. Supposons de plus que ce système est composé d'un certain nombre n de modules a priori. Nous définissons chacun de ces modules comme étant un PDM sur l'espace S. S est supposé grand. Chaque module raisonne sur un nombre raisonnable de macro-états de S. C'est simplement la définition des macro-états qui différencie les modules les uns par rapport aux autres.

Au départ, chacun de ces modules aura un certain nombre de macro-états aléatoires. A chaque fois qu'on proposera une tâche au système (une fonction récompense sur S), chacun des modules essaiera de la résoudre (en utilisant sa représentation du monde et en appliquant par exemple l'algorithme Value Iteration). Les n modules seront alors en compétition. Il est particulièrement intéressant de remarquer que chaque module va calculer, à partir de son approximation du monde, une fonction de valeur. A un instant donné, le système est dans un état et peut donc directement comparer l'espérance de récompense que lui propose chacun des modules (la fonction de valeur en l'état courant). Il sera alors immédiat de déterminer celui qui propose la meilleure solution : ce module sera désigné comme vainqueur de la compétition. Le module vainqueur subira alors une spécialisation pour la tâche qu'il viendra de résoudre comme expliqué dans la section 1.2.

3.2 Arguments

Nous donnons dans cette partie des arguments qui justifient notre approche modulaire par rapport à l'approche qui consiste à utiliser un PDM centralisé. Nous voyons 3 avantages spécifiques à notre approche :

1. Le temps de réponse d'un tel système modulaire devrait être plus rapide que celui d'un système centralisé. En effet, le temps nécessaire pour chaque itération est proportionnel à la taille de l'espace d'état au carré[11]. Pour répondre à toutes les tâches qui lui sont demandées, un système centralisé devrait avoir une représentation détaillée de toutes les zones requises pour l'ensemble des tâches et donc plus d'états qu'un simple module spécialisé.
2. L'apprentissage d'une tâche pourra influencer la réalisation d'autres tâches. Un module qui se sera spécialisé sur une tâche donnée, pourra faire bénéficier de cet apprentissage si une nouvelle tâche partage avec elle de nombreuses caractéristiques.
3. Enfin le dernier argument que nous donnons est typique des système connexionnistes. Si un module arrête de fonctionner, les autres (en particulier le deuxième module le plus adapté) devraient prendre le relais.

4 Conclusion et perspectives

Nous avons présenté notre démarche et les références (apprentissage par renforcement et auto-organisation) sur lesquelles nous nous appuyons pour proposer une architecture intelligente modulaire et autonome. Afin de valider nos idées, nous prévoyons de les mettre en œuvre dans un monde simulé, puis dans le monde réel sur une plate-forme robotique [2]. Nous devrions présenter les premiers résultats de ces travaux au moment des présentations orales.

Nous envisageons d'ores et déja un certain nombre de pistes qui pourraient permettre d'étendre notre travail. Premièrement une extension du modèle PDM, les PDM partiellement observés (PDMPO) abordent explicitement le fait que les perceptions instantanées d'un robot ne contiennent

2. Robot Koala de l'équipe CORTEX

pas toute l'information nécessaire pour prendre de bonnes décisions (le contexte de travail, par exemple, peut être dicriminant). Nous projetons donc d'étendre le procédé de spécialisation aux PDMPO. Ensuite, nous souhaiterions voir comment les différents modules pourraient coopérer (plutôt que d'être en permanente compétition). Pour ce faire, nous envisageons d'une part de permettre aux différents modules de partager leurs estimations de fonction de valeur. D'autre part, nous pourrions introduire plus de hiérarchie entre les modules : certains modules pourraient identifier les tâches qu'ils résolvent bien et proposer ces tâches à d'autres modules comme actions élémentaires.

Références

1. P. Blanchet. Modular growing network architectures for td learning. In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan B. Pollack, and Stewart W. Wilson, editors, *From animals to animats 4*, pages 343–352, Cambridge, MA, 1996. MIT Press.
2. L. Bougrain and F. Alexandre. Unsupervised connectionist algorithms for clustering an environmental data set: A comparison. *Neurocomputing*, 1999.
3. Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.
4. B. Digney. Emergent hierarchical control structures: Learning reactive /hierarchical relationships in reinforcement environments.
5. B. Fritzke. A self-organizing network that can follow non-stationary distributions. In *ICANN'97: International Conference on Artificial Neural Networks*, pages 613–618. Springer, 1997.
6. Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of Markov decision processes using macro-actions. pages 220–229.
7. R. Jacobs, M. Jordan, and A. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Technical report.
8. Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *International Conference on Machine Learning*, pages 167–173, 1993.
9. J. Lange, H. Voigt, and D. Wolf. Growing artificial neural networks based on correlation measures, taske decomposition and local attention neurons, 1994.
10. P. Laroche. *Processus Décisionnels de Markov appliqués à la planification sous incertitudes*. PhD thesis, Université Henri Poincaré - Nancy 1, 2000.
11. M. L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, 1996. Also Technical Report CS-96-09.
12. J. MacQueen. Some methods of classification and analysis of multivariate observations, 1967.
13. R. Munos and A. Moore. Variable resolution discretization in optimal control, 1999.
14. R. Munos and A. Moore. Rates of convergence for variable resolution schemes in optimal control. In *International Conference on Machine Learning*, 2000.
15. M. Puterman. Markov decision processes, 1994.
16. Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, 1991.
17. A. Roe, S. Pallas, J. Hahm, and M. Sur. A map of visual space induced in primary auditory-cortex, 1990.
18. R.S. Sutton and A.G. Barto. *Reinforcement Learning, An introduction*. BradFord Book. The MIT Press, 1998.
19. 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.