Using a Hybrid Evolutionary-A* Approach for Learning Reactive Behaviours

Carlos Cotta and José M. Troya

Dept. of Lenguajes y CC.CC., University of Málaga,
Complejo Tecnológico (3.2.49), Campus de Teatinos,
E-29071, Málaga, Spain
{ccottap, troya}@lcc.uma.es

Abstract. A hybrid approach for learning reactive behaviours is presented in this work. This approach is based on combining evolutionary algorithms (EAs) with the A* algorithm. Such combination is done within the framework of Dynastically Optimal Forma Recombination, and tries to exploit the positive features of EAs and A* (e.g., implicit parallelism, accuracy and use of domain knowledge) while avoiding their potential drawbacks (e.g., premature convergence and combinatorial explosion). The resulting hybrid algorithm is shown to provide better results, both in terms of quality and in terms of generalisation.

1 Introduction

The control of autonomous mobile agents is a complex task to which great efforts are devoted due to its practical applications. In general, such control is achieved by means of both *planning* and *reactive* components [13]. Each of these components has it own particularities, and can be examined in combination (e.g., [1, 2, 7]) or in isolation (e.g., [11, 12]). In line with the latter, this work focuses on the acquisition of reactive behaviours in mobile agents.

Reactive behaviours are driven by a stimulus-to-response mapping, i.e., the agent receives some information about its local environment and decides the action(s) to carry on exclusively on the basis of such information. This kind of behaviour has usually the advantage of not requiring any underlying global model of the world in which the agent is located. The obvious drawback of reactive systems is the fact that they can get stuck into dead-ends, situations in which the correct action does not only depend on the locally available information but also the structure of the world at a higher-level (hence the necessity of long-term planning capabilities). Nevertheless, reactive systems have been shown to provide a very good performance in a wide variety of scenarios and remain a very suitable option when response-time is critical.

There exist several techniques for designing reactive systems. These can be typically classified into reinforcement learning and optimisation techniques. Algorithms such as Holland's bucket brigade, Sutton's temporal difference learning or Watkins's Q-learning lie in the first class. Within the second class, evolutionary algorithms deserve special attention because of their power and flexibility.

Regarding the use of these techniques for this purpose (e.g., [8, 9, 11]), a critical point is the use of as much domain knowledge as possible. Otherwise, the user would be relying on a fortuitous matching between her algorithm and the problem under consideration [14]. Such specialised algorithms are usually termed *hybrid* evolutionary algorithms [6].

This work presents a hybrid evolutionary algorithm for acquiring reactive behaviours. In the proposed algorithm, domain knowledge is included by using a specialised technique (the A* algorithm) as an internal operator. The remainder of the article is organised as follows. First, the agent and the worlds used in the experiments are described (Sect. 2). Next, the classical A* approach for solving the posed problem is shown (Sect. 3). Then, the hybrid algorithm is introduced (Sect. 4). Subsequently, experimental results are presented (Sect. 5). Finally, some conclusions are extracted and future work is outlined (Sect. 6).

2 The Agent and its World

The agent used in this work is located in a two-dimensional toroidal grid-world in which several obstacles are distributed. The purpose of the agent is to reach a certain target point from its initial location within an allowed time. To do so, the agent is capable of making some elementary actions such as moving straight ahead a single grid square, turning 90º to its left, or turning 90º to its right. Obviously, the agent must avoid obstacles while navigating through its world. For this purpose, it is equipped with proximate sensors that can inform of the presence or absence of obstacles in front of the agent, 90º to its left, or 90º to its right (see Fig. 1, left). In addition, these sensors can also detect whether the target point is in any of these three locations or not.

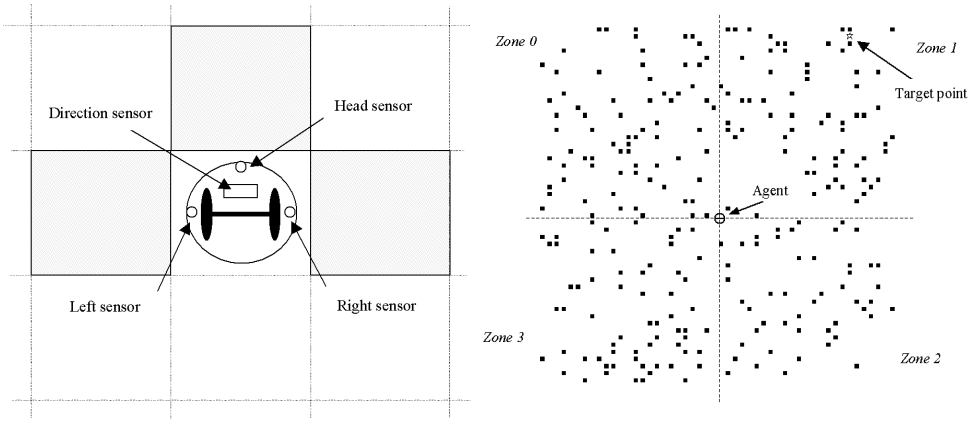

Fig. 1. (Left) Structure of the agent used in experiments. (Right) Example world and regions into which it is divided according to the location of the target point.

The agent is equipped with a direction sensor as well. This sensor allows determining in which of four imaginary regions of the world the target point is located. These regions are illustrated in Fig. 1 (right). It must be noted that these regions are not absolute but relative to the agent's actual orientation. For example, the agent is facing North in Fig. 1 and hence the target point is in zone 1. Now, if the agent turned 90^o to its right, the target would be in zone 0. Notice also that these regions are determined taking into account the toroidal shape of the world. Thus, if the agent were a few positions South from the location shown in the previous example, the target point might happen to be in zone 2.

According to this description, the goal is to design a reactive behaviour allowing the agent reaching its target in as many situations as possible. Such reactive behaviour can be defined in a variety of ways, e.g., using a neural network [15], a fuzzy rule-base [8], a cellular automata [3], etc. This work is in line with the latter approach. To be precise, a lookup-table is sought relating every possible sensorial input with a primitive action. At each time-step, the agent must look up the action that corresponds to the current inputs and carry it out. Since each proximate sensor can provide three different inputs (`OBSTACLE`, `NO-OBSTACLE`, `TARGET`), and the direction sensor can return four values, the resulting table has $3^3 \cdot 4 = 108$ entries. Since three primitive actions – `MOVE-AHEAD`, `TURN-LEFT`, `TURN-RIGHT` – are available, this implies a search space of $3^{108} \approx 3 \cdot 10^{51}$ tables.

3 A Classical Approach: A*

A classical approach for finding the lookup-table mentioned above is the utilisation of the A* algorithm. Based on incrementally constructing solutions in an *intelligent* fashion, this technique constitutes a powerful tool for solving search problems to optimality. Before getting into the application of this technique to the design of reactive behaviours, some notation details must be given.

Let $\mathcal{W}$ be the current world, and let α be the configuration of the agent (position and orientation). Now, let $\mathcal{I}(\mathcal{W}, \alpha)$ be the sensorial input of the agent when configured according to α. Let $\hat{M}$ be a (possibly underspecified) function relating sensorial inputs with actions, and let $\hat{M}_1 \supset \hat{M}_2$ whenever $\hat{M}_2$ provides the same outputs that $\hat{M}_1$ does and $\hat{M}_2$ is defined in at least one case in which $\hat{M}_1$ is not. Finally, let τ be the maximum allowed time for reaching the target and let Ξ be a function such that $\Xi(\mathcal{W}, \alpha, \hat{M}) = (\langle \alpha_1, \cdots, \alpha_k \rangle, \eta)$. This function provides a trace of the agent trajectory across configuration space when behaving according to $\hat{M}$. The value η is an indication of the final status of the agent: `AT-TARGET`, `COLLISION`, `TIMED-OUT` or `UNKNOWN`. This latter value is returned whenever no action is specified in $\hat{M}$ for the current input.

Now, the application of the A* algorithm requires the availability of an optimistic evaluation function ψ such that $\psi(\mathcal{W}, \alpha, \hat{M})$ provides a lower bound on the number of steps necessary for reaching the target when the agent is configured as α and behaves according to any $\hat{M}', \hat{M} \supset \hat{M}'$. It is easy to see that making ψ return the Manhattan distance from the agent's current location to

the target point fulfils this requirement. Having defined this function, the whole process is as follows:

1. Let $P_0^0 = (\alpha_0, \hat{M}_0, 0, t_0)$, where α_0 is the initial configuration of the agent, $\hat{M}_0$ is a fully underspecified function, and $t_0 = \psi(\mathcal{W}, \alpha_0, \hat{M}_0)$. Let $P^* = (\hat{M}_0, \infty)$ be the current best solution. Insert P_0^0 in the node queue.
2. If the node queue is empty, go to 3. Otherwise let $P_j^i = (\alpha, \hat{M}, h, t)$ be the first element in the queue.
   (a) Let $\Xi(\mathcal{W}, \alpha, \hat{M}) = (\langle \alpha_1, \cdots, \alpha_k \rangle, \eta)$.
   (b) if $\eta \neq$ `UNKNOWN` then
      i. if $\eta =$ `COLLISION` or $\eta =$ `TIMED-OUT` then $\bar{P} = (\hat{M}, t' + \tau)$, where $t' = h + k + \psi(\mathcal{W}, \alpha_k, \hat{M})$.
      ii. if $\eta =$ `AT-TARGET` then $\bar{P} = (\hat{M}, h + k)$.
      iii. If $\bar{P}$ is better than P^*, update the latter and purge nodes in the queue.
      iv. Go to 2.
   (c) Create three nodes P_{j+1}^{3i+1}, P_{j+1}^{3i+2}, and P_{j+1}^{3i+3} from P_j^i. Each node is $P_{j+1}^{3i+r} = (\alpha_k, \hat{M}_r, h', t_r)$, where $\hat{M}_r$ is obtained by extending $\hat{M}$ to return the rth possible action when the input is $\mathcal{I}(\mathcal{W}, \alpha_k)$, $h' = h + k$, and $t_r = \psi(\mathcal{W}, \alpha_k, \hat{M}_r)$. Insert these nodes in the queue keeping it ordered according to the sum of the last two components of each node.
   (d) Go to 2.
3. Return P^*

This algorithm will thus return the lookup-table allowing the agent reach the target in minimal time from the given starting point. Since the problem has been posed with the goal of obtaining a generalisable reactive behaviour, the process must be slightly modified. To be precise, a training set is selected and the A* algorithm tries to find the table that minimises the sum of the times required to reach the target in each training case or, if such a solution is not possible, a table that firstly maximises the number of training cases solved and secondly minimises the total time. Notice that no global model of the world (i.e., high-level knowledge about the distribution of obstacles) is required. All information used for finding the optimal solution is locally obtained through simulation.

This algorithm has been evaluated on a set of nine different worlds. These worlds are named as Wxy, where $x \in \{10, 25, 50\}$ indicates the dimension of the world (each world is a $x \times x$ grid), and $y \in \{a, b, c\}$ indicates the density of obstacles (5%, 10% and 20% respectively). For each world, a training set of five cases has been selected. Subsequently, the best solution found has been tested for generality on a test set whose size depends on the dimension of the world (50, 400 and 2000 cases respectively). The results are shown in Table 1.

These results are very indicative of the two main drawbacks of the A* algorithm. On the one hand, it is very sensitive to the size of the task to be solved. As it can be seen, the algorithm expended a high computational effort for solving W25b and ran out of memory in three cases (W25c, W50b, and W50c). Moreover, it did not find any fully satisfactory solution for all training cases in W50b

Table 1. Results of the A* algorithm on nine different worlds. The cost values are measured as the number of single simulation steps carried out.

World	Timeout	Optimal solution	Iterations	Cost	Performance on test set
W10a		13.20	11615	289857	74%
W10b	25	13.80	19664	300391	88%
W10c		14.00	26429	281091	58%
W25a		32.40	75818	10352732	63%
W25b	150	31.60	222788	29083614	48%
W25c		[26.40, 51.00]	>300000	>25000000	36%
W50a		54.60	119864	36861614	66%
W50b	400	[45.60, 479.40]	>150000	>37000000	14%
W50c		[34.80, 1629.80]	>110000	>16000000	2%

and W50c (1 and 4 training cases were left unsolved[1]). On the other hand, the solutions found are not very generalisable. This is a direct consequence of the internal functioning of the A* algorithm. Assume that the final solution is found when evaluating node P_j^i. This node was obtained as successive extensions of $P_{j-1}^{i\backslash 3}, P_{j-2}^{i\backslash 9}, \cdots, P_0^0$. Hence, it contains information regarding the best decisions to be taken only in the situations found during this *optimal* path, i.e., the path from the root node of the implicitly defined search tree to the optimal leaf node. All that may have been learnt in solving other situations is discarded since these situations do not take place in this optimal path.

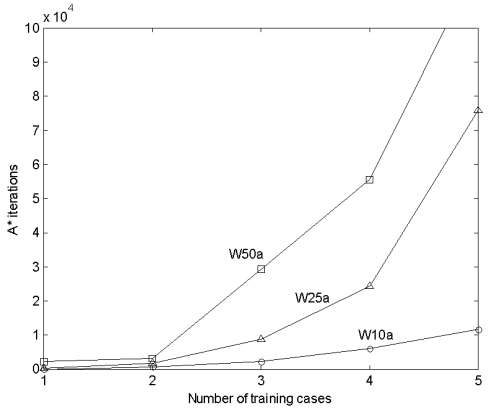

Fig. 2. Growth of the computational cost of the A* algorithm when the number of training cases is increased.

[1] These results were not bad a priori since there might exist no better solution. However, further experimentation with the hybrid EA showed that this was not the case.

This generalisation problem could be solved by considering a larger training set whose optimal solution covered all possible situations. However, the subsequent combinatorial explosion makes this approach unrealistic. This is illustrated in Fig. 2. As it can be seen, the computational cost of the algorithm grows very fast when the size of the training set is increased. For this reason, it is clear that alternative approaches must be found. These will be discussed in next section.

4 The Hybrid EA-A* Approach

Evolutionary algorithms constitute a very suitable alternative to A* for finding the lookup-table. A naïve approach for applying EAs to this problem would firstly consist of defining an encoding function for storing the lookup-table into an individual, e.g., a linear chromosome in which the rows of the table are consecutively arranged. Since this is an orthogonal representation [10] (i.e., all combinations of genes are feasible), the next step would simply involve selecting any of the standard genetic operators that can be found in the literature (e.g., single-point crossover – SPX –, uniform crossover – UX –, etc.).

However, such a simple approach is likely to provide very poor results. Recall that this is highly epistatic problem in which the value of each gene (i.e., a specific action to be carried out when a certain sensorial input is received) does not contribute with a fixed amount to the fitness of an individual. On the contrary, the goodness of the reactive behaviour defined is determined by the interplay between all genes. For this reason, a blind recombination operator that randomly shuffles the genetic material of recombined solutions will provably produce solutions with a phenotype (reactive behaviour) completely unrelated to the parents, even when the latter are genotypically similar. In an extreme situation, it may even reduce to macromutation.

The algorithm would be largely more effective if it were able to extract positive behavioural patterns from existing solutions and transmit them to the offspring. This can be achieved within the framework of Dynastically Optimal Forma Recombination [4] (DOR). This framework comprises a family of recombination operators of the form

$$\mathrm{DOR} : \mathcal{S} \times \mathcal{S} \times \mathcal{S} \rightarrow [0, 1], \tag{1}$$

where $\mathcal{S}$ is the search space and $\mathrm{DOR}(x, y, z)$ is the probability of generating z when recombining x and y. Besides the obvious $\sum_{z \in \mathcal{S}} \mathrm{DOR}(x, y, z) = 1$, the probability distribution induced by these operators verify that

$$\mathrm{DOR}(x, y, z) > 0 \Rightarrow \{[z \in \Gamma(\{x, y\})] \wedge [\forall w \in \Gamma(\{x, y\}) : \phi(w) \geq \phi(z)]\} \tag{2}$$

where ϕ is the fitness function (to be minimised without loss of generality) and $\Gamma(\{x, y\})$ is the dynastic potential [10] of x and y, i.e., the set of solutions that can be built using nothing but the information contained in x and y.

Thus, the solutions created by DOR are the best that can be constructed using the genetic material of the parents. On the one hand, this implies that

DOR is a fully transmitting operator, i.e., no implicit mutation (genetic information not present in any of the parents) is introduced in the offspring. On the other hand, the fitness-oriented functioning of DOR makes valuable portions of solutions be transmitted to offspring only if they contribute to a good resulting behaviour. In other words, DOR is capable of identifying valuable high-order formae, preventing their disruption. This intelligent combination of information has provided very good results on epistatic problems [5].

In order to implement DOR, it is required to use an embedded A*-like mechanism so as to find the best solution in the dynastic potential of the parents. In this case, the algorithm described in Sect. 3 can be used. It is only necessary to modify step 2c by considering that the possible actions to be taken in a given situation $\mathcal{I}$ are just those present in any of the parents for $\mathcal{I}$. Notice that the search carried out by this subordinate A* algorithm is thus restricted to small portions of the search space and hence its computational cost is largely reduced with respect to the original unrestricted version. Moreover, individuals in the population tend to be more similar as the EA converges and, subsequently, the dynastic potential of selected solutions tends to be smaller and DOR is less computationally expensive.

This combination of EAs and A* has an additional advantage. Each individual carries an information that reflects its past evolution (in fact, the evolution of its ancestors). This way, things that were learnt in the past are retained as long as they do not negatively affect the present behaviour. This "accumulated history" effect is also present in a simple EA, but the learning capabilities of the hybrid algorithm are larger. For this reason, solutions obtained with the hybrid EA are expected to be more general than either the EA or the A* algorithm by themselves. This will be studied in next section.

5 Experimental Results

Experiments have been done with a steady-state EA ($popsize = 100$, $p_c = .9$, p_m=1/chromosomeLength) using ranking selection ($\eta^+ = 2.0, \eta^- = 0.0$). This algorithm has been run 40 times for each operator and test world. In order to make a fair comparison between DOR and the other simpler operators, each run is terminated when a fixed number of simulation steps ($\tau \cdot 10^5$ in these experiments, where τ is the timeout value) is reached. Thus, the internal calculations performed by DOR are effectively accounted. As in Sect. 3, a training set of five cases is used in the fitness function.

First of all, Fig. 3 shows how the hybrid EA is much more successful in solving the training cases. As it can be seen, while standard operators only provide an acceptable performance on the smallest instances and with the lowest obstacle density, DOR consistently yields satisfactory results: above a 70% of the runs provide a fully successful solution for the training set (the percentage is 100% for 5 out of 9 test worlds). The exception is world W50c for which none of the operators could find a full solution (it must be noted that such a solution may

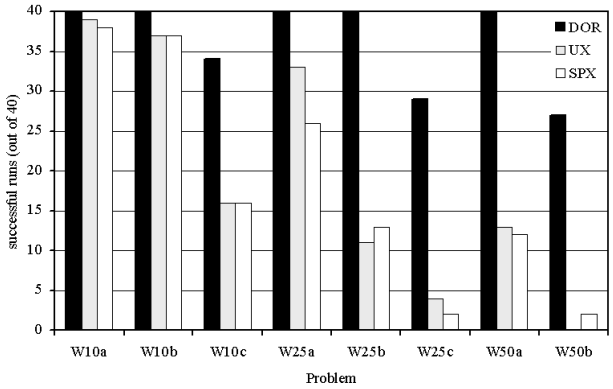

Fig. 3. Number of runs in which each operator provided a fully satisfactory solution for the training set.

not exist). Nevertheless, DOR was capable of solving 3 out of the 5 training cases while SPX could only solve one and UX could not solve any of them.

Table 2 shows a more detailed summary of the results. Notice that DOR is not only more effective in finding satisfactory solutions, but also provides higher-quality results. By comparing the median values[2] provided by DOR with the optimal/best-known solutions (see Table 1), it can be seen that DOR yields near-optimal solutions. Moreover, the lower variance of DOR results with respect to SPX and UX indicates a more stable algorithm.

Table 2. Comparison of different genetic operators on nine different environments. All results correspond to series of forty runs.

		SPX			UX			DOR		
World	Timeout	mean	σ	median	mean	σ	median	mean	σ	median
W10a		17.36	12.06	16.00	15.88	7.47	15.20	13.85	0.87	13.80
W10b	25	22.15	25.96	16.20	18.10	9.26	16.20	14.35	0.74	14.20
W10c		51.49	38.39	59.60	53.85	38.81	59.60	18.75	10.75	14.00
W25a		176.70	210.00	45.60	121.63	193.32	40.00	33.17	2.14	32.40
W25b	150	376.08	285.43	399.30	380.96	272.08	468.40	36.65	6.67	34.00
W25c		520.32	179.07	616.80	502.23	218.40	616.40	99.92	107.70	38.80
W50a		442.39	348.27	438.00	423.87	335.45	438.00	61.17	11.72	58.80
W50b	400	1532.52	554.56	1636.60	457.65	457.65	1636.60	337.32	506.03	103.8
W50c		2021.45	60.77	2031.20	2031.20	0.00	2031.20	1638.78	538.75	1650.60

[2] The median value seems to be a more representative measure of the quality of the results than the mean value since the former is much less sensitive to outliers. Furthermore, it provides an reasonable alternative to averaging the fitness of solutions that solve the whole training set with solutions that do not solve any training case.

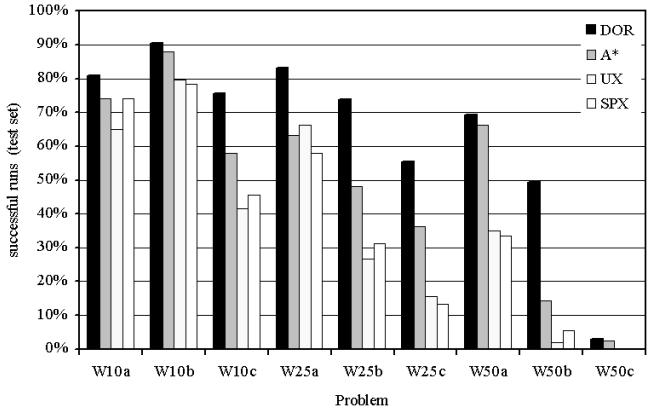

Fig. 4. Percentage of the test set solved for each of the techniques considered.

Finally, the results obtained with the EA are tested for generality. Fig. 4 shows the results. Firstly, notice the poor results of standard EAs. The solutions provided by UX and SPX do not reach 50% success in 6 out of 9 worlds. The A* algorithm performs better than standard EAs, but its performance quickly drops when the density of obstacles is increased. The hybrid EA provide the overall best results, outperforming both A* and standard EAs on all worlds. Moreover, this improvement is larger on instances with higher obstacle densities. It must be noted that the results on W50c are not satisfactory for any algorithm (although the hybrid algorithm remains the best). This is a really hard instance as mentioned before, and may require longer evolution times and/or a larger training set to cope with such a tough environment.

6 Conclusions

This work has presented a hybrid approach for learning reactive rule-bases. By combining EAs with the A* algorithm, a synergetic system has been achieved. This hybrid algorithm has been shown to provide higher-quality results than standard EAs. These results are also better than those of the A* algorithm in terms of their generalisation to previously unseen test cases. Furthermore, the hybrid EA is capable of tackling instances in which the A* algorithm would suffer the effects of the combinatorial explosion.

Future work will try to extend these results to more sophisticated agents. In this sense, notice that most details of the agent are encapsulated within the simulation function Ξ and hence they do not affect the presented algorithm qualitatively. Nevertheless, it is clear that issues regarding simulations of higher computational cost are worth studying. Work is in progress in this area. Additionally, new environments and tasks to be solved will be tackled as well.

Acknowledgement

This work is supported by the Spanish *Comisión Interministerial de Ciencia y Tecnología* (CICYT) under grant TIC99-0754-C03-03.

References

1. K. Ali and A. Goel. Combining navigational planning and reactive control. In *Theories of Action, Planning, and Robot Control. Bridging the Gap: Proceedings of the 1996 AAAI Workshop*, pages 1–9, Menlo Park, CA, 1996. AAAI Press.
2. C.T.C. Arsene and A.M.S. Zalzala. Control of autonomous robots using fuzzy logic controllers tuned by genetic algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 428–435. IEEE NNC - EP Society - IEE, 1999.
3. T.D. Barfoot and D'Eleuterio G.M.T. An evolutionary approach to multiagent heap formation. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 420–427. IEEE NNC - EP Society - IEE, 1999.
4. C. Cotta, E. Alba, and J.M. Troya. Utilising dynastically optimal forma recombination in hybrid genetic algorithms. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 305–314. Springer-Verlag, Berlin, 1998.
5. C. Cotta and J.M. Troya. Tackling epistatic problems using dynastically optimal recombination. In B. Reusch, editor, *Computational Intelligence. Theory and Applications*, volume 1625 of *Lecture Notes in Computer Science*, pages 197–205. Springer-Verlag, Berlin Heidelberg, 1999.
6. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Computer Library, New York, 1991.
7. J.-Y. Donnart and J.-A. Meyer. Learning reactive and planning rules in a motivationally autonomous animat. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):381–195, 1996.
8. F. Hoffmann and G. Pfister. Learning of a fuzzy control rule base using messy genetic algorithms. In F. Herrera and J.L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 279–305. Physica-Verlag, Heidelberg, 1996.
9. J.R. Koza. *Genetic Programming*. MIT Press, Cambridge MA, 1992.
10. N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10:339–384, 1994.
11. A.C. Schultz and J.J. Grefenstette. Using a genetic algorithm to learn behaviours for autonomous vehicles. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, pages 739–749, Hilton Head SC, 1992.
12. C. Thornton. Learning where to go without knowing where that is: the acquisition of a non-reactive mobot behaviour by explicitation. Technical Report CSRP-361, School of Cognitive and Computing Sciences, University of Sussex, 1994.
13. G. Weiss. *Multiagents Systems: a Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge MA, 1999.
14. D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.
15. B. Yamauchi and R. Beer. Integrating reactive, sequential and learning behaviour using dynamical neural networks. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behaviour*, pages 382–391, Cambridge MA, 1994. MIT Press/Bradford Books.