01 Jan 2004

Evolutionary Algorithms, Markov Decision Processes, Adaptive Critic Designs, and Clustering: Commonalities, Hybridization and Performance

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Samuel A. Mulder

Recommended Citation

D. C. Wunsch and S. A. Mulder, "Evolutionary Algorithms, Markov Decision Processes, Adaptive Critic Designs, and Clustering: Commonalities, Hybridization and Performance," *Proceedings of the International Conference on Intelligent Sensing and Information Processing, 2004*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2004.
The definitive version is available at https://doi.org/10.1109/ICISIP.2004.1287704

Evolutionary Algorithms, Markov Decision Processes, Adaptive Critic Designs, and Clustering: Commonalities, Hybridization, and Performance

Donald C. Wunsch II and Samuel Mulder
Applied Computational Intelligence Laboratory
University of Missouri – Rolla
Rolla, MO 65409 USA
www.ece.umr.edu/acil

Abstract

We briefly review and compare the mathematical formulation of Markov Decision Processes (MDP) and Evolutionary Algorithms (EA). In so doing, we observe that the Adaptive Critic Design (ACD) approach to MDP can be viewed as a special form of EA. This leads us to pose pertinent questions about possible expansions of the methodology of ACD. This expansive view of EA is not limited to ACD. We discuss how it is possible to consider the powerful Chained Lin-Kernighan (chained LK) algorithm for the Traveling Salesman Problem (TSP) as a degenerate case of EA. Finally, we review some recent TSP results, using clustering to divide-and-conquer, that provide superior speed and scalability.

Formulation of MDP

An MDP consists of the following:

Epochs: Typically numbered from t = 0, 1, ... N, where N is the horizon, possibly infinite.

States: Indicated by a variable, say, x, which can be continuous or discrete. The discrete state space is typical, even if infinite. Continuous state spaces are equivalent to Partially Observable Markov Decision Processes.

Actions: Represented by a set A = {a1, a2, a3, ...}. The ai can be functions of x and/or t.

Rewards (often punishments or costs): Incrementally given by u(x(t), a(t)), and cumulatively, by the cost-to-go function:

$$J(t) = u(t) + J(t+1). \quad (1)$$

Thus, $J(0) = u(0) + u(1) + \ldots + u(N)$.

Transition Probabilities: $P[x(t+1) \mid x(t), a(t)]$.

The Markov property implies that this is independent of previous x and a values. In the general case, when the probabilities do not reduce to binary, deterministic values, this implies that (1) is suppressing a conditional expectation operator on the right hand side of the equation.

We also need to introduce the notion of a policy $\pi_i = [a_i(x,0), a_i(x,1), \ldots a_i(x,N)]$, defined for all x, t. If π_i is independent of t, we call it a stationary policy.

Adaptive Critic Designs

An ACD uses a critic element to mediate learning. Most implementations to date used two or more (typically three) neural networks, where an Action network generated an output to indicate the action a_i, and a Critic network estimated the cost-to-go function J(t) or, alternatively, its derivative with respect to the state. The training signal for the Critic is of interest. In the perfect case, (1) applies always. Since the Critic's output is only an estimate, $\hat{J}$, we consider the quantity:

$$e = \hat{J}(t) - [u(t) + \hat{J}(t+1)]. \quad (2)$$

Since this would be equal to zero in the ideal case, any deviation is used as an error signal to train the Critic network.

Formulation of EA

An EA approach to a problem consists of:

Data Structure:

A function X: Solutions →Data Structure. This mapping needs to be chosen in such a way as to provide information that will be useful in improving the solution iteratively.

Time is therefore implicitly included, so at any time, the function is denoted x(t).
Initialization: Size of population of candidate solutions, related characteristics.
Variation operators: Number of parents, number of offspring, type of crossover, probability of applying the operator – all can be denoted v(x(t)).
Scoring the offspring and selection pressure, whether proportional to fitness, truncating the most fit, or a combination (called tournament), all denoted by s(v(x(t))).

The whole process can compactly be represented $x(t+1) = s(v(x(t)))$. (3)

Implicit in the above, but worth mentioning explicitly, is the factor of randomness in the functioning of the above process. Although it is possible to specify the above deterministically, this is considered a degenerate case, which eliminates much of the advantage of EA.

Reformulation of an ACD as an Instance of EA

To consider an ACD in the light of EA, we first must resolve some issues of notation. The notion of time in the two formulations, for example, has different meanings. For MDP and ACD, time is part of the problem specification, that is, we are required to find optimal actions over time, for the defined horizon. For EA, time is an iteration variable to search for, hopefully, increasingly fit solution populations. This is easily resolved by replacing t in the EA formulation by an iteration variable i. Also, for EA, x indicates a population of solutions, but it is a state of the external system for MDP and ACD. The appropriate notation for a population of solutions is $\Pi(i) = [\pi 1, \pi 2, \ldots \pi m]$, where m can vary from iteration to iteration. Now, we can focus on the fitness and selection criteria from (3) and the cost function estimate in (2). (3) becomes:

$$\Pi(i+1) = s(v(\Pi(i))). \quad (4)$$

We can thus view the Action network of an ACD as a mapping from the state space to a policy, that is, at any time t, from any state x in X, to any of the π_i. We can denote the time dependence of the policy by making the index of the policy dependent on time, i.e. $\pi_{i(t)}$. Thus:

$NA(t): X \rightarrow \pi_{i(t)}$.

So, the ingredients for an EA - ACD are:
Initial population:
One parent, the initial Action network mapping, $\pi_{i(0)}$.

Fitness function:
$\hat{J}$ (From the Critic network. Should be as small as possible.)

Variation Function:
The variation induced by backpropagation on NA(t) using $\hat{J}$ as an error signal. Training of the Critic indirectly acts as a variation as well by changing $\hat{J}$.
This variation function can be simple, as in (Widrow, 73), (Watkins, 89), (Werbos, 90) or quite complex (Werbos, 90), (Prokhorov and Wunsch, 97).

Selection Pressure:
A binary function: If the neural network has not converged yet, keep the child. If the network has converged, keep the parent. A variety of mechanisms are used to determine whether the network has converged, such as: $\hat{J}$ sufficiently small, weight changes sufficiently small, or training set error equal to validation set error.

This section has reinterpreted an ACD as an EA. In the next section, we will examine what this means for future potential ACD.

Alternative Formulation for ACD

It is straightforward to observe that there is no need for some of the constraints in the above formulation. By removing these, we automatically expand the definition of an ACD to allow a broader family of EA. Basically, anything except the fitness function can change and still serve the intended purpose of an ACD.

Initial Population: N parents.

Fitness function: $\hat{J}$

Variation function:
Virtually anything. Examples are backpropagation on N neural networks in parallel, crossover of weights, or any training algorithm.

Selection Pressure:
Also open to choice. Anything tending to favor lower values of $\hat{J}$ should work.

Memetic EA and the TSP

In general, EA, are mostly useful on problems where no good algorithms are known. EA have been applied to the TSP before, but have been generally unsuccessful as compared to state of the art heuristic algorithms. A good overview of the current state of the art in TSP optimization is given in (Gutin and Punnen, 2002). EA receive barely a passing mention, because they have not been competitive with this type of problem. (Fogel, 1994) and (Larranaga et al, 1999) present a good overview of straightforward implementations of genetic algorithms for the TSP. While they do give decent tours on small problem instances (less than 100 cities), the times required to solve such instances are very large compared to other heuristic approaches.

Until recently, the best results on large-scale TSP instances came from variations on an algorithm proposed by Lin and Kernighan in 1973 (Lin and Kernighan, 1973), (Rego and Glover, 2002). This algorithm takes a randomly selected tour and optimizes it by stages until a local minimum is found. This result is saved and a new random tour selected to begin the process again. Given enough time, the optimal tour can be found for any instance of the problem, although in practice this is limited to small instances for an absolute result due to time constraints. Even with extremely large instances, however, reasonably short tours can typically be found (Johnson and McGeoch, 2002). A popular variant, responsible for the best performance of this class of algorithms, is called chained-LK. This, instead of beginning with a random tour at each new iteration, perturbs the previous tour and optimizes from there. This perturbation generally consists of a process known as a double bridge (Applegate et. al., 2000). This consists of a 4-edge exchange that is never found by the basic LK algorithm and is illustrated in Figure 1.

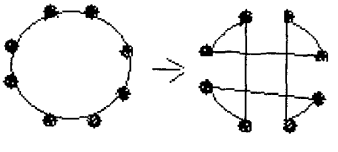

Figure 1. Double Bridge

This exchange has the effect of changing the global shape of the tour and starts the basic LK algorithm in a position that should find a different minimal tour, increasing the chance that the actual minimum tour is found.

A new approach to evolutionary algorithms (Ulder et al, 1990) combines local search algorithms with evolutionary algorithms to create a new hybrid. These algorithms use local search as a learning function to create a memetic algorithm. A memetic algorithm is one in which parents pass on not only their genetic material, but also learned characteristics. By applying a local search operator to members of the population, the search space is explored more efficiently. To see how this works, consider the space of possible tours as a high dimensional search space. If we consider local search as a hill-climbing technique, then the evolutionary approach places new population members in the space and the local search climbs to the nearest hill. This greatly reduces the size of the space to be searched, and insures that good solutions are found at each generation. We can also think of the chained LK algorithm as a special case with one population member and one child created each generation using the double bridge as a mutation operator. It remains to be seen whether increasing the population size brings any advantage in terms of search speed. At the very least, it makes a parallel implementation trivial to code.

Clustering Divide-and-Conquer TSP Algorithm

The algorithm presented in this paper combines ART (Carpenter and Grossberg, 1987, 88) and LK local optimization to divide and conquer instances of the TSP. We begin by reading in the cities, stored in TSP-LIB format (Mostcoto, 2002). The ordering of the cities in memory represents the current tour. This is known as a permutation representation. This permutation representation is chosen because weight-matrix representations become intractable for large values of n, i.e. a 250k-city problem would require around 62.5 GB of memory to store an edge-weight matrix. Since the problem is Euclidean, it is sufficient to store (x,y) coordinates for each city and calculate distances on the fly.

The first stage of the algorithm involves sorting the cities into clusters using the ART algorithm described previously. Our variation of ART uses

the vigilance parameter to set a maximum distance from the current pattern. A vigilance parameter between 0 and 1 is considered and used as a percentage of the global space to determine the vigilance distance. Values were chosen based on the number and size of individual clusters desired, but typical values ranged from 0.80 to 0.97. The learning rate was set to 0.02. The clusters at this point were still in a random order. The individual clusters were then each passed to a version of the LK algorithm also described above. Since the size of the tours was controlled and kept under a thousand cities, we allowed the LK search depth to be infinite as long as the total improvement for a given swap series remained positive.

Now we are faced with the problem of combining a number of subtours back into one complete tour. This may be accomplished by adding the cluster tours into a combined tour, one at a time. Obviously the order in which the tours are added back is important. To minimize the cost added by inter-tour edges, it is important to add tours that are adjacent to the current combined tour. Our method for accomplishing this is to add the tours in order of increasing distance from the origin. It is not clear whether this is best, and this is an area for future research. The other major factor involved with merging the tours is running time. Since this a potentially global operation, care must be exercised in the nature of the algorithm. For example, attempting to find an optimal linking between the two tours could be at least an $O(n_g^2)$ algorithm, which is unacceptable, because the n involved would be the total number of cities, not just the cities in a tour. To avoid the $O(n_g^2)$ global operation, we first find the centroid of the cluster to be added. This is just the average of the x and y coordinates of each city in the cluster, and is easily calculated in $O(n_c)$, where the n involved is the size of an individual cluster. We then find the k nearest cities to that centroid in the combined tour. Clearly, this operation requires $O(n_g)$ time. Next, we consider each of the k cities from the main tour to determine the cost of inserting the cluster tour in place of the following edge. This involves the comparison of k cities to n_c cities to determine the lowest cost matching, yielding a running time of $O(k*n_c)$, where $k << n_g$. Finally, the cluster tour is inserted into the merged tour at the best location discovered. Some illustrative results are provided in Figure 2. The advantage over chained LK becomes apparent for large instances. For example, for the ten million city problem, the Clustering Divide-and-Conquer algorithm runs in 10,529 seconds on a PC, while chained LK takes 43,631 seconds, to achieve virtually identical quality tours (within 1%.) For the twenty-five million city tour, Clustering Divide-and-Conquer takes 13,500 seconds, while chained LK can not solve the problem at all within the memory constraints of the machine. Superior memory management guarantees improved scalability for the Divide-and-Conquer approach.

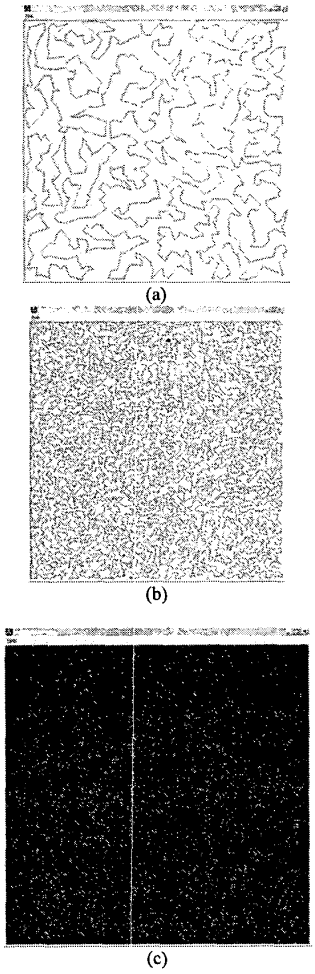

Figure 2. TSP resulting tours for (a) 1k, (b) 10k, (c) 1M cities.

Conclusion

We have examined and compared the fundamental definitions of ACD and EA. In so doing, this allowed us to see an ACD as a special case of an EA. It is important to note that the overlap is not complete, for example, it excludes deterministic decision processes. However, this interpretation does include many of the richest and most interesting problem representations in ACD. This allows us to expand the notion of ACD, whereby we create a representation based on fundamental principles, as opposed to arbitrary hybrids. We also discuss how this expansive view of EA applies to other techniques, such as the chained LK for TSP. Finally, we introduce a clustering divide-and-conquer algorithm that combines ART networks with chained LK, for superior performance on large TSP instances.

References

K. Menger. Das Botenproblem. Ergebnisse Eines Mathematischen Kolloquiums, 2:11-12, 1932.

S. Lin, B. W. Kernighan, (1973). An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research 21, 498-516.

D. Applegate, W. Cook, and A. Rohe, (2000). Chained Lin-Kernighan for large traveling salesman problems. Technical report http://www.keck.caam.rice.edu/reports/ chained lk.ps

T. Cormen, C. Leiserson, R. Rivest (1996). Introduction to Algorithms (pp. 954-960). Cambridge, MA: MIT Press.

R. Agarwala, D.L. Applegate, D. Maglott, G.D. Schuler, A.A. Schaffler (2000). A Fast and Scalable Radiation Hybrid Map Construction and Integration Strategy. http://www.ncbi.nlm.nih.gov/genome/rhmap/

C. A. Bailey, T. W. McLain, R. W. Beard (to appear). Fuel Saving Strategies for Dual Spacecraft Interferometry Missions. Journal of the Astronatical Science.

C.H. Papadimitriou. (1977). The Euclidean Traveling Salesman Problem is NP-Complete. Theoretical Computer Science *4*, 237-244.

M.L. Braun, J.M. Buhmann (2002). The Noisy Euclidean Traveling Salesman Problem and Learning. Advances in Neural Information Processing Systems 14. Cambridge, MA: MIT Press.

E.M. Cochrane, J.M. Cochrane (1999), Exploring competition and co-operation for solving the Euclidean Traveling Salesman Problem by using the Self–Organizing Map. Artificial Neural Networks, 1, 180-185.

S. Aurora (1998). Polynomial time approximation schemes for {Euclidean} traveling salesman and other geometric problems. Journal of the ACM, V.45 N.5, 753-782.

G. Carpenter, S. Grossberg (1988). The art of adaptive pattern recognition by a self-organizing neural network. IEEE Computer, March, 47-88.

P. Moscoto (2002). TSPBIB Home Page. http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html

D. Applegate, R. Bixby, V. Chvatal, W. Cook (2001). Concorde – a code for solving Traveling Salesman Problems. http://www.math.princeton.edu/tsp/concorde.html

N. Vishwanathan, D. Wunsch (2001). A Hybrid Approach to the TSP. IEEE International Joint Conference on Neural Networks, Washington, DC.

C. Rego, F. Glover (2002). Local Search and Metaheuristics. In G. Gutin and A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations (pp. 309-368). Boston: Kluwer Academic Publishers.

D. Johnson, L. McGeoch (2002). Experimental Analysis of Heuristics for the STSP. G. Gutin and A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations (pp. 369-487). Boston: Kluwer Academic Publishers.

G. Gutin and A. Punnen eds. (2002). The Traveling Salesman Problem and It's Variations. Kluwer Academic Publishers, Netherlands.

D.B. Fogel (1988). An Evolutionary Approach to the Traveling Salesman Problem. Biological Cybernetics, 60, (pp 139-144).

P. Larranaga, C. M. H. Kujipers, R. H. Murga, I. Inza and S. Dizdarevic (1999). "Genetic Algorithm for the Traveling Salesman Problem: A Review of Representations and Operations," Artificial Intelligence Review, vol. 13, no. 2, (pp. 129-170).

N.L.J. Ulder, E.H.L. Aarts, H.J. Bandelt, P.J.M. Van Laarhoven, and E. Pesch (1990). Genetic Local Search Algorithms for the Traveling Salesman Problem, in Parallel Problem Solving from Nature. Springer-Verlag, Berlin Heidelberg, (pp. 106-116).

N. Vishwanathan, D. Wunsch (2001). "A Hybrid Approach to the TSP", IEEE International Joint Conference on Neural Networks, Washington, DC.

S. Mulder and D. Wunsch (2002). "Large-Scale Traveling Salesman Problem via Neural Network Divide and Conquer". ICONIP.

S. Mulder and D. Wunsch (2003). "Using Adaptive Resonance Theory and Local Optimization to Divide and Conquer Large Scale Traveling Salesman Problems". IJCNN, Portland, OR.

S. Mulder and D. Wunsch (2003). "Million City Traveling Salesman Problem Solution by Divide and Conquer Clustering with Adaptive Resonance Neural Networks". Neural Networks. Vol. 16, no. 5-6, (pp. 827-832).

Barto, A., Sutton R. and Anderson, C., "Neuronlike elements that can solve difficult learning control problems," *IEEE Trans. on Systems, Man and Cybernatics*, vol. 13, pp. 834-846, 1983.

Bellman, R., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

Fogel, D., "An introduction to simulated evolutionary optimization," *IEEE Trans. on Neural Networks*, Vol. 5, no. 1, pp. 3-14, 1994.

Fogel, D., "Evolution, neural networks, games and intelligence," Fogel, D., *Proceedings of the IEEE*, Vol. 87, no. 9, pp. 1471-1496, 1999.

Prokhorov, D. and Wunsch, D., "Adaptive Critic Designs," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 997-1007, September, 1997.

Sutton, R., "Learning to predict by the method of temporal differences," *Machine Learning*, no. 3, pp. 9-44, 1988.

Venayagamoorthy, G., Harley R. and Wunsch, D., "Comparison of Heuristic Dynamic Programming and Dual Heuristic Programming Adaptive Critics for Neurocontrol of a Turbogenerator", *IEEE Transactions on Neural Networks*, Volume: 13 Issue: 3, Page(s): 764 - 773, 2002.

Watkins, C, *Learning from Delayed Rewards*, Ph.D. dissertation, Cambridge University, Cambridge, England, 1989.

Werbos, P., "A menu of designs for reinforcement learning over time," Werbos, P., in W. T. Miller, R. Sutton and P. Werbos (Eds.), *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990.

White, David A., and Donald A. Sofge, *Handbook of Intelligence Control: Neural, Fuzzy, and Adaptive Approaches*, editors, Multiscience Press, 1992.

Widrow, Bernard, Nerendra Gupta, and Sidhartha Maitra, "Punish /reward: Learning with a critic in adaptive threshold systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 3, no. 5, pp. 455--465, 1973.

Wunsch, D., "The cellular simultaneous recurrent network adaptive critic design for the generalized maze problem has a simple closed-form solution," *Proceedings of International Joint Conference on Neural Networks*, Como Italy, Vol. 3, pp.79-82, 2000.

Carpenter, G., and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115, 1987.

Carpenter, G., and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *IEEE Computer*, vol. 21, no. 3, pp. 77-88, 1988.