

The Cooperative Royal Road: Avoiding *Hitchhiking*

Gabriela Ochoa¹, Evelyne Lutton², and Edmund Burke¹

¹ Automated Scheduling, Optimisation and Planning Group, School of Computer Science & IT, University of Nottingham, Nottingham NG8 1BB, UK

² COMPLEX Team, INRIA Rocquencourt
Domaine de Voluceau BP 105, 78153, Le Chesnay Cedex, France

Abstract. We propose using the so called *Royal Road* functions as test functions for cooperative co-evolutionary algorithms (CCEAs). The Royal Road functions were created in the early 90's with the aim of demonstrating the superiority of genetic algorithms over local search methods. Unexpectedly, the opposite was found to be true. The research deepened our understanding of the phenomenon of *hitchhiking* where unfavorable alleles may become established in the population following an early association with an instance of a highly fit schema. Here, we take advantage of the modular and hierarchical structure of the Royal Road functions to adapt them to a co-evolutionary setting. Using a multiple population approach, we show that a CCEA easily outperforms a standard genetic algorithm on the Royal Road functions, by naturally overcoming the hitchhiking effect. Moreover, we found that the optimal number of sub-populations for the CCEA is not the same as the number of components that the function can be linearly separated into, and propose an explanation for this behavior. We argue that this class of functions may serve in foundational studies of cooperative co-evolution.

1 Introduction

Co-evolutionary Algorithms (CEAs) represent a natural extension to standard evolutionary algorithms for tackling complex problems; they can be generally defined as a class of evolutionary methods in which the fitness of an individual depends on its relationship to other members of the population. Several co-evolutionary approaches have been proposed in the literature; they vary widely, but the most fundamental classification relies on the distinction between cooperation and competition. In cooperative algorithms, individuals are rewarded when they work well with other individuals, and punished otherwise. Whereas in competitive algorithms, individuals are rewarded at the expense of those with which they interact. Most of the work on CEAs has been in competitive models; there has been, however, an increased interest in cooperation to tackle difficult optimization problems by means of problem decomposition [7, 3, 14, 18, 11, 10]. The behavior of CEAs is very complicated and often counter-intuitive. Moreover, our knowledge about the dynamics and ways of improving standard evolutionary algorithms, is not directly transferable to co-evolution [15]. Thus, there is

a need to conduct foundational research on co-evolutionary systems in order to improve their applicability as a problem solving methodology. With this in mind, we propose using the so called Royal Road functions [13, 4] as test functions in cooperative co-evolution. The Royal Road functions were proposed with the aim of isolating some of the features of fitness landscapes thought to be most relevant to the performance of genetic algorithms. Surprisingly, it was found that a random-mutation hill climber significantly outperformed the genetic algorithm on these functions. However, this work led to a greater understanding of the phenomenon of hitchhiking in evolutionary search, whereby some deleterious alleles may become fixed in the population, after an early association with a highly fit schema.

Cooperative co-evolutionary approaches are generally applied to decomposable problems. Thus, we take advantage of the modular and hierarchical structure of the Royal Road test functions to adapt them to the co-evolutionary setting. We argue that these functions may serve theoretical studies of cooperative co-evolution, since the landscape can be varied in a number of ways, and the global optimum and all possible fitness values are known in advance. It would also be possible to study the dynamics of the search process by tracing the origins and history of individual building blocks. Moreover, these functions may be decomposed in several ways (including one or more blocks on each sub-component), which made them useful in studies testing the automated emergence of co-adapted components [18]. This study also makes a comparison between a standard and a cooperative evolutionary algorithm on several instances of the Royal Road functions. The cooperative algorithm explores all the alternative problem decompositions possible with the modular Royal Road functions. Our results show a clear advantage of the cooperative algorithm in this scenario, and we go further to analyze why this is the case. This analysis leads us to revisit the hitchhiking effect and the *building blocks* hypothesis in genetic algorithms.

Next section gives a brief overview on cooperative co-evolution, distinguishing between single and multiple population approaches, and describing some test problems used so far when studying cooperative co-evolution. Thereafter, section 3, introduces the Royal Road functions and describes the hitchhiking phenomenon. Section 4, describes the algorithms and parameter settings used, whilst section 5 presents and analyses our results. Finally, section 6 summarises our findings and suggests directions for future work.

2 Cooperative co-evolution

Previous work on extending evolutionary algorithms to allow cooperative co-evolution can be divided into approaches that have a single population of interbreeding individuals, and those that maintain multiple interacting populations.

Single population approaches: The earliest single-population method that extended the basic evolutionary model to allow the emergence of co-adapted subcomponents was the classifier system [6]; which is a rule-based learning

paradigm that evolves fixed length stimulus-response rules. An interesting generalization of this paradigm for solving complex problems was proposed in [1], where an aggregation of multiple individuals (in a single population) is considered for solving the inverse problem for Iterated Function Systems. In this approach, which has been termed *Parisian Evolution*, an additional fitness measure (a “local” fitness) is used to independently evaluate the sub-components during the search process, while a “global” fitness is used at each generation to measure the progress of the aggregate solution. This scheme is well suited for incorporating additional or incomplete information about the searched solution. However, in order to avoid trivial and degenerate solutions, a special mechanism for maintaining the population diversity should be devised. Successful applications of the Parisian Approach can be found in the image analysis and signal processing literature [11, 2]; and in data retrieval applications [10].

Multiple population approaches: The first authors to apply a multi-species cooperative co-evolutionary approach to tackle a difficult optimization problem were Husbands and Mill [8, 7] who successfully co-evolved job-shop schedules, using a parallel distributed algorithm. A few years later, the work by Potter and De Jong [16] helped to popularise the idea of cooperative co-evolution as an optimisation tool. The authors devised a multiple population framework where a decomposition of the problem into subcomponents should be identified. Each component is, then, assigned to a subpopulation that evolves simultaneously but in isolation to the other sub-populations. The fitness of an individual in a given sub-population is calculated after selecting collaborators from the other sub-populations in order to form a complete solution. Notice that diversity in the ecosystem is, in this framework, naturally achieved through maintaining genetically isolated populations. This framework has been further analysed [22] by considering a relationship between cooperative co-evolution and evolutionary game theory, and thus studying it from a dynamical systems perspective. From the problem-solving point of view, multi-species cooperative co-evolution has been applied to neural networks and concept learning [14, 17, 18]; and to inventory control optimisation [3].

2.1 Abstract test functions

Most foundational empirical studies of cooperative co-evolution have used non linear function optimization problems as benchmark [17, 23]. These problems are well suited for cooperative co-evolution, since a natural decomposition is straightforward: each subpopulation represents a particular variable of the function. In [14], much simpler functions (*oneRidge* and *twoRidges*) are studied. In his dissertation [15], Potter used several test functions including, a simple binary string covering task, continuous nonlinear functions, and Kauffman’s coupled NK landscapes [9]. In a further, more theoretically oriented PhD dissertation, Wiegand [21] used cooperative versions of test functions such as the *OneMax*, *LeadingOnes*, and *Trap* functions.

3 Royal Road functions and the hitchhiking effect

The building-block hypothesis [5] states that the genetic algorithm works well when short, low-order, highly-fit schemata (building blocks) recombine to form even more highly-fit, higher-order schemata. Thus, the genetic algorithm’s search power has been attributed mainly to this ability to produce increasingly fitter partial solutions by combining hierarchies of building-blocks. Despite recent criticism, empirical evidence, and theoretical arguments against the building-blocks hypothesis [19], the study of schemata has been fundamental in our understanding of genetic algorithms. The first empirical counter-evidence against the building-block hypothesis was produced by Holland himself, in collaboration with Mitchell and Forrest [13, 4]. They created the Royal Road functions, which highlight one feature of landscapes: hierarchy of schemata, in order to demonstrate the superiority of genetic algorithms (and hence the usefulness of recombination) over local search methods such as hill-climbing. Unexpectedly, their results demonstrated the opposite: a commonly used hill-climbing scheme (*random-mutation hill-climbing*) significantly outperformed the genetic algorithm on these functions. With this hill-climbing approach, a string is randomly generated and its fitness is evaluated. The string is then mutated (by a bit-flip) at a randomly chosen position, and the new fitness is evaluated. If the new string has an equal or higher fitness, it replaces the old string. This procedure is iterated until the optimum has been found or a maximum number of evaluations is reached. It is ideal for the Royal Road functions, since it traverses its “plateaus” and reaches the successive fitness levels. However, the algorithm (as with any other hill-climber) will have problems with any function with many local minima. The authors [13, 4] also found that although crossover contributes to genetic algorithm performance on the Royal Road functions, there was a detrimental role of “stepping stones” - fit intermediate-order schemata obtained by recombining fit low-order schemata. The explanation suggested for these unexpected results lies in the phenomenon of hitchhiking (or spurious correlation), which they describe as follows [12]: “once an instance of a higher-order schema is discovered, its high fitness allows the schema to spread quickly in the population, with 0s in other positions in the string hitchhiking along with the 1s in the schema’s defined positions. This slows down the discovery of the schema’s defined positions. Hitchhiking can, in general, be a serious bottleneck for the GA.”

3.1 Functions $R1$ and $R2$

To construct a Royal Road function [4], an optimum string is selected and broken up into a number of small building blocks. Then, values are assigned to each low-order schema and each possible intermediate combination of low-order schemata. Those values are, thereafter, used to compute the fitness of a bit string x in terms of the schemata of which it is an instance.

The function $R1$ (Figure 1) is computed as follows: a bit string x gets 8 points added to its fitness for each of the given order-8 schemata ($s_i, i = 1, 2, \dots, 8$)

4 Methods

As the cooperative co-evolutionary algorithm, we used the multiple populations approach (see Figure 3) firstly proposed by Potter and De Jong [16]; and later studied by other authors [22, 15].

```

gen = 0
for each species s do
  Pop_s(gen) = initialized population
  evaluate(Pop_s(gen))
while not terminated do
  gen++
  for each species s do
    Pop_s(gen) <- select(Pop_s(gen - 1))
    recombine(Pop_s(gen))
    evaluate(Pop_s(gen))
    survive(Pop_s(gen))

```

Fig. 3. The structure of a cooperative co-evolutionary algorithm.

In order to adapt the Royal Road functions to the co-evolutionary setting, a solution string is broken into equally sized sub-strings that contain one (or more) of the original function lower-order schemata. Each of these sub-strings represents a problem subcomponent, and is thus assigned to a separate population. A global solution is assembled by concatenating these sub-components. We used the simplest method of evaluating an individual in a given population; which is to couple it with the current best members of the remaining populations, apply the resulting string to the global function, and assign the resulting value as the fitness of the subpopulation. The initial fitness of each subpopulation member is computed by combining it with a random individual from each of the other species. We evaluated the cooperative co-evolutionary algorithm by comparing its performance with that of a standard genetic algorithm on several Royal Road functions (both *R1* and *R2*). In order to maintain resemblance with the originally proposed Royal Road functions [13, 4], we used functions (both *R1* and *R2*) with lower-order schemata (building blocks - BBs) of length 8. With regard to the string length, we considered functions of $L = 64, 128, \text{ and } 256$, that is, functions containing 8, 16 and 32 of these BBs. Several numbers of subpopulations (or species, SP) were considered starting from the minimum of two species, and doubling this number up to the maximum given by the number of BBs in the function. This corresponds to subpopulations having, respectively, a string length equal to half of the total Royal Road function length (half the number of BBs), down to subpopulations having a string length of 8 (i.e. a single BB). To set the size of each sub-population, we select a fixed number of individuals in the whole ecosystem, and thereafter distributed them equally among the sub-populations. For setting this ecosystem population size, we tested a range of values (64, 128, 256, and 512) and selected, for each string length and function, the size producing the best performance (see Table 1). We also selected

the optimal population size, in this range, for the standard genetic algorithm, which turned out to be the smallest size explored (64) for all the functions. The remaining algorithm components were equal for the standard and cooperative genetic algorithm, and were held constant over the experiments. Specifically, we used binary tournament selection, 2-point crossover (rate = 0.8) and bit-flip mutation (rate = $1/L$, L = chromosome length), and 50 replicas for experiments. Further methodological details and the performance measures, are described in the following section.

	<i>R1</i>			<i>R2</i>		
	$L = 64$	$L = 128$	$L = 256$	$L = 64$	$L = 128$	$L = 256$
<i>SGA</i>	64	64	64	64	64	64
<i>CCEA</i>	128	256	256	64	128	256

Table 1. Optimal population sizes (in the set of: 64, 128, 256, and 512), for both the standard genetic algorithm (SGA) and cooperative co-evolutionary algorithm (CCEA). For CCEA the size producing the best performance over all the number of species tested, was considered. L stands for the Royal Road function’s string length.

5 Empirical results and analysis

For comparing the performances, the algorithms were allowed to continue until the optimum string was discovered, and the number of evaluations of this discovery was recorded. Table 2 shows the average number of evaluations ($\times 10^2$) to discover the optimum string, for the *R1* (top) and *R2* (bottom) functions. The standard deviations are shown within brackets. Note that 50 replicas for each experiment were carried out. From Table 2, we see that the CCEA (with any number of species) clearly outperformed the SGA on all the instances studied. On average, the CCEA (with the appropriate number of species) found the optimum a factor of about two times faster on *R1*, and three times faster on *R2*. Notice that, as has been reported before, the SGA performs better on *R1* than on *R2*. This is not the case, however, for the CCEA where the algorithm has a similar best performance on both *R1* and *R2*. Our explanation for the improved performance of the CCEA lies in the phenomenon of hitchhiking, described in section 3. By maintaining separate populations, the CCEA is able to avoid hitchhiking, since each sub-population samples independently each schema region. Thus, more than one desirable schema may appear simultaneously in the ecosystem, and thereafter these sub-components are aggregated when calculating the global function. In [12], the authors identify some features that would improve the performance of GAs as compared to other heuristic search algorithms. These are: (i) *independent samples*, (ii) *sequestering desired schemas*, and (iii) *instantaneous crossover* of desired schemas. It is clear that a cooperative genetic algorithm contains all these features, and entails a better implementation of the building-blocks hypothesis (see section 3).

<i>R1</i>			
Algorithm	$L = 64$ (8 BBs)	$L = 128$ (16 BBs)	$L = 256$ (32 BBs)
<i>SGA</i>	227.8 (90.23)	665.1 (225.99)	2150.9 (805.70)
<i>CCEA</i> ₂	165.4 (66.22)	571.4 (270.62)	2161.2 (768.20)
<i>CCEA</i> ₄	142.2 (71.56)	402.2 (143.18)	1473.9 (577.42)
<i>CCEA</i> ₈	114.1 (43.30)	327.1 (130.71)	1094.8 (464.03)
<i>CCEA</i> ₁₆		365.2 (128.84)	851.5 (319.73)
<i>CCEA</i> ₃₂			1140.9 (352.26)
<i>R2</i>			
Algorithm	$L = 64$ (8 BBs)	$L = 128$ (16 BBs)	$L = 256$ (32 BBs)
<i>SGA</i>	241.3 (119.69)	947.3 (622.71)	3278.6 (1560.43)
<i>CCEA</i> ₂	173.5 (74.76)	640.8 (301.91)	2480.1 (1141.24)
<i>CCEA</i> ₄	115.6 (51.22)	412.2 (214.98)	1432.1 (472.53)
<i>CCEA</i> ₈	127.0 (56.36)	305.7 (92.60)	1094.5 (430.02)
<i>CCEA</i> ₁₆		399.9 (142.02)	876.2 (330.46)
<i>CCEA</i> ₃₂			1389.2 (365.93)

Table 2. Average number of evaluations ($\times 10^2$) and standard deviations to find the optimum on the *R1* (top) and *R2* (bottom) functions. The sub-index in the CCEA corresponds to the number of species.

Another interesting observation (Table 2 and Figure 4) is that the number of species (*SP*) on the CCEA, that produced the best performance was consistently (with the exception of *R1* with 8 BBs - $L = 64$) achieved by *SP* = half of the number of blocks in the function. This corresponds to a sub-population string length of 16 bits, namely two 8-bits BBs. This can be more clearly appreciated in Figure 4, which compares the algorithm’s performance (*SGA* and CCEA with different a number of sub-populations) on both *R1* and *R2*, with 16 and 32 blocks ($L = 128$ and 256). Thus, the optimal number of sub-populations for the CCEA (i.e. the number of problem sub-components) is not the same as the number of pieces that the function can be linearly separated into, which, in principle, may appear to be a counter-intuitive observation. The following set of experiments, offers an explanation for this behavior.

5.1 Dynamic behavior

In order to find an explanation for the observed optimal number of sub-populations in the CCEA (*SP* = half of the number of blocks in the function), we study, in this section, the algorithms’s dynamic behavior. For the analysis we selected the *R1* function with $L = 128$ (16 BBs), and a CCEA with 8 and 16 sub-populations, which were the *SP* values producing the best performance in this scenario. Figure 5 illustrates the performance curves for both a single run (left-hand plot) and averaging 50 runs (right-hand plot). Each point in the curves represents the global objective function value of the aggregated solution. Each run lasted 50×10^4 function evaluations, and the global objective value was sampled every 100 evaluations.

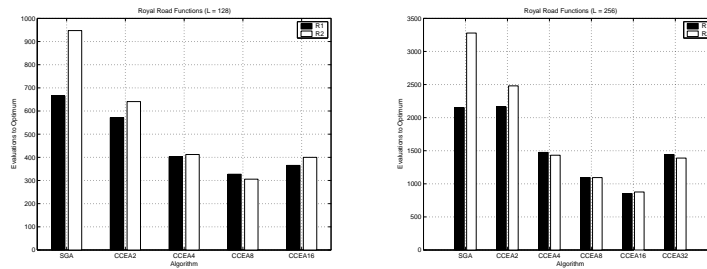


Fig. 4. Comparing the algorithm’s performance on both $R1$ and $R2$ with $L = 128$ (left plot), and $L = 256$ (right plot). The bars measure the average number of evaluations ($\times 10^2$) to find the optimum.

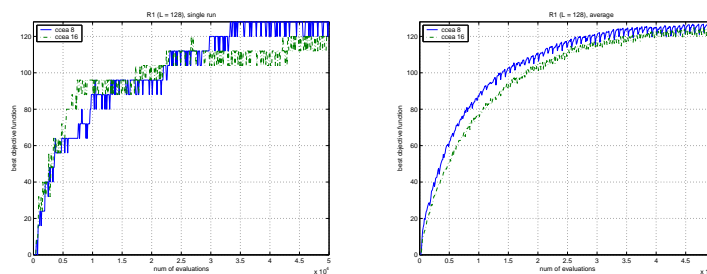


Fig. 5. Comparing the dynamic behavior of CCEA with 8 and 16 sub-populations on the $R1$ function with $L = 128$. The left-hand plot illustrates a single run, whereas the right-hand plot, averages 50 runs.

Notice that, on average, the CCEA with 8 species outperformed the CCEA with 16 species throughout the whole run. The single run dynamics looks more complicated, with both algorithms having similar performance at the initial stages of the search, and $CCEA_{16}$ dominating at some intermediate stages. Towards the final stages of the search, however, it is $CCEA_8$ which is producing and maintaining higher fitness values. Notice that in the process of the run, the fitness levels are discovered and lost several times before getting established, which suggests that the convergence behavior of the multi-population CCEA is slower and more complex than that of a standard genetic algorithm. The curves in Figure 5 show the behavior of the aggregate solution, hiding the information about the fitness contribution of each sub-population. In order to have a closer look at the fitness contribution of each schema or BB to the global objective function, Figure 6 illustrates the contribution of an example schema, where without loss of generality we select schema 9 (s_9). Both single run (top plot)

and average (bottom plot) behavior are illustrated for the CCEA with 8 and 16 species³.

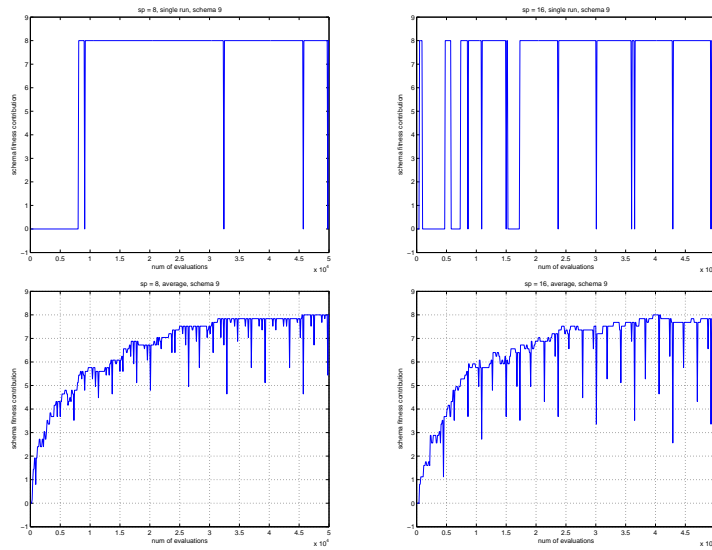


Fig. 6. Comparing the dynamic fitness contribution of a representative schema (s_9) in a CCEA with 8 (left-hand plots) and 16 (right-hand plots) species. The top plots illustrate single runs, whereas the bottom plots show the averages of 50 runs.

From the single run plots of the fitness contribution of schema 9 (figure 6, top plots), it can be seen that the CCEA with 16 species is more unstable at maintaining the BBs of 8 consecutive ones (fitness contribution of 8), in other words the BB appears and disappears more easily throughout the search. This is because when $SP = 16$, an individual in each sub-population is composed of a single BB, so any bit mutation would break it. This is not the case with $SP = 8$, where each individual in a sub-population is composed of two consecutive BBs. In this scenario, a bit mutation may destroy one of the BBs, but keep the other, and the correct concatenation of the two BBs can be easily recovered by a recombination event. This behavior is also reflected by the average plots (Figure 6, bottom plots) where the fitness contribution curve of schema 9 reaches lower levels (deeper drops) when $SP = 16$ (right-hand bottom plot). These plots therefore suggest that having sub-populations composed by individuals containing two BBs instead of a single BB, would benefit the stability and permanence of the appropriately set BBs in the sub-populations, thus supporting the overall better global behavior.

³ The plots for all the other schemata were qualitatively similar, and are not shown due to space limits.

6 Conclusions

Cooperative co-evolution is suitable to decomposable problems; consequently, we have taken advantage of the modular and hierarchical structure of the Royal Road functions to adapt them as test functions for cooperative co-evolutionary algorithms (CCEAs). Our empirical results show that a CCEA clearly outperforms a standard genetic algorithm on the Royal Road functions, confirming our intuition that cooperative co-evolution helps in overcoming the so-called hitchhiking (or spurious correlation) effect, which is known to hinder the performance of evolutionary algorithms. This suggests that CCEAs may be an advantageous technique, even for static optimization, as they entail a better instantiation of the building-blocks hypothesis.

An advantage of the Royal Road functions as test functions in cooperative co-evolution is that they admit several natural decompositions, which makes them useful in studies to test the automated emergence of co-adapted components. Our results show that having two basic sub-components, instead of a single sub-component for sub-populations produced better overall performance, which suggests that caution should be taken when manually proposing a problem decomposition. A potential drawback of the Royal Road functions as test beds for cooperative co-evolution, is similar to that highlighted for standard evolutionary search; namely the independence (or separation) between the building blocks. To overcome this limitation, Watson et al. [20] have proposed the so called Hierarchical If-and-only-if (H-IFF) functions that have a hierarchical decomposable structure where sub-problems are not separable. In consequence, a natural extension of our contribution will be to propose a cooperative version of the H-IFF family of functions. Another interesting extension would be to compare and assess the scenarios where a single-population implementation of cooperative co-evolution (such as the Parisian approach [1]) would be advantageous over a multiple-population one.

References

1. Pierre Collet, Evelyne Lutton, Frederic Raynal, and Marc Schoenauer, *Polar IFS+parisian genetic programming=efficient IFS inverse problem solving*, Genetic Programming and Evolvable Machines **1** (2000), no. 4, 339–361.
2. E. Dunn, G. Olague, and E. Lutton, *Parisian camera placement for vision metrology*, Pattern Recognition Letters **27** (2006), no. 11, 1209–1219.
3. Roger Eriksson and Björn Olsson, *Cooperative coevolution in inventory control optimisation*, Proceedings of the Third International Conference on Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, 1997.
4. Stephanie Forrest and Melanie Mitchell, *Relative building-block fitness and the building block hypothesis*, Foundations of Genetic Algorithms 2 (San Mateo), Morgan Kaufmann, 1993, pp. 109–126.
5. John H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.
6. John H. Holland and J. S. Reitman, *Cognitive systems based on adaptive algorithms*, Pattern-directed inference systems, Academic Press, New York, 1978.

7. P. Husbands, *Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation*, Lecture Notes in Computer Science **865** (1994), 150–166.
8. Philip Husbands and Frank Mill, *Simulated co-evolution as the mechanism for emergent planning and scheduling*, Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufman, 1991, pp. 264–270.
9. Stuart A. Kauffman and Sonke Johnsen, *Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches*, Artificial Life II, Addison-Wesley, 1992, pp. 325–369.
10. Yann Landrin-Schweitzer, Pierre Collet, and Evelyne Lutton, *Introducing lateral thinking in search engines*, Genetic Programming and Evolvable Machines **7** (2006), no. 1, 9–31.
11. Jean Louchet, Maud Guyon, Marie-Jeanne Lesot, and Amine M. Boumaza, *Dynamic flies: a new pattern recognition tool applied to stereo sequence processing*, Pattern Recognition Letters **23** (2002), no. 1-3.
12. Melanie Mitchell, *When will a genetic algorithm outperform hill-climbing?*, Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufman, 1993.
13. Melanie Mitchell, Stephanie Forrest, and John H. Holland, *The royal road for genetic algorithms: Fitness landscapes and GA performance*, Proc. of the First European Conference on Artificial Life, MIT Press, 1992, pp. 245–254.
14. David E. Moriarty and Risto Miikkulainen, *Forming neural networks through efficient and adaptive coevolution*, Evolutionary Computation **5** (1998), no. 4, 373–399.
15. Elena Popovici and Kenneth A. De Jong, *Understanding cooperative co-evolutionary dynamics via simple fitness landscapes*, Genetic and Evolutionary Computation Conference, GECCO 2005, ACM, 2005, pp. 507–514.
16. Mitchell A. Potter and Kenneth De Jong, *A cooperative coevolutionary approach to function optimization*, Parallel Problem Solving from Nature – PPSN III, Springer, 1994, Lecture Notes in Computer Science 866, pp. 249–257.
17. ———, *The coevolution of antibodies for concept learning*, Parallel Problem Solving from Nature – PPSN V, Springer, 1998, Lecture Notes in Computer Science 1498, pp. 530–539.
18. Mitchell A. Potter and Kenneth A. De Jong, *Cooperative coevolution: An architecture for evolving coadapted subcomponents*, Evolutionary Computation **8** (2000), no. 1, 1–29.
19. Colin Reeves and Jonathan Rowe, *Genetic algorithms: Principles and perspectives*, Kluwer, 2002.
20. Richard A. Watson and Jordan B. Pollack, *Hierarchically consistent test problems for genetic algorithms*, 1999 Congress on Evolutionary Computation, 1999, pp. 1406–1413.
21. R. P. Wiegand, *An analysis of cooperative coevolutionary algorithms*, Ph.D. thesis, George Mason University, 2004.
22. R. Paul Wiegand, William Liles, and Kenneth De Jong, *Analyzing cooperative coevolution with evolutionary game theory*, Proceedings of the 2002 Congress on Evolutionary Computation CEC2002, 2002, pp. 1600–1605.
23. R. Paul Wiegand, William C. Liles, and Kenneth A. De Jong, *An empirical analysis of collaboration methods in cooperative coevolutionary algorithms*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann, 2001, pp. 1235–1242.