Working from Blueprints: Evolutionary Learning for Design

Sushil J. Louis

Department of Computer Science
University of Nevada
Reno - 89557
email: sushil@cs.unr.edu

When confronted by a problem, human designers often work forward from similar previously solved problems to solve the current problem. Based on this principle, we propose a new learning system especially suited for design using case-based reasoning principles to augment genetic algorithm search. When confronted with a problem we seed a genetic algorithm's initial population with solutions to similar, previously solved problems and the genetic algorithm then adapts its seeded population toward solving the current problem. Preliminary results on open-shop scheduling and re-scheduling indicate the feasibility of this approach.

Key words: Design, Genetic Algorithms, Case-Based Reasoning

1 Introduction

Genetic algorithms (GAs) are randomized parallel search algorithms that search from a population of points [Holland, 1975]. We typically randomly initialize the starting population so that a genetic algorithm can proceed from an unbiased sample of the search space. However in many application areas we confront sets of similar problems. It makes little sense to start a problem solving search attempt from scratch with a random initial population when previous search attempts may have yielded useful information about the search space. Instead, seeding a genetic algorithm's initial population with solutions to similar previously solved problems can provide information (a search bias) that, hopefully, increases the efficiency of the search. Our approach borrows ideas from case-based reasoning (CBR) in which old problem and solution information, stored as cases in a case-base, help solve a new problem [Riesbeck and Schank, 1989].

The approach outlined in this paper has many application areas in engineering design [Tong and Sriram, 1992] and we use example problems from open shop

Preprint submitted to Elsevier Preprint

scheduling and re-scheduling as a test-bed for the work reported in this paper. Although we only consider genetic algorithms, the basic ideas can be extended to other evolutionary computation methods, and with some modifications, to other heuristic search methods.

The next two sub-sections provide short introductions to case-based reasoning and genetic algorithms. For researchers familiar with the topics these subsections may be skipped without loss of continuity. Section 2 outlines the issues in combining GAs and CBR and discusses previous work in this area. The open-shop scheduling and re-scheduling problems and our methodology and experimental results on the test problems are presented in subsequent sections. The last section presents conclusions and directions for future work.

1.1 Case-Based Reasoning

Case-based reasoning (CBR) is based on the idea that reasoning and explanation can best be done with reference to prior experience, stored in memory as *cases* [Riesbeck and Schank, 1989,Bareiss, 1991]. When confronted with a problem to solve, a case-based reasoner extracts the most similar case in memory and uses information from the retrieved case and any available domain information to tackle the current problem. The strategy is first to collect and index a large and varied collection of examples, and then, when presented with a new situation, to fetch and possibly manipulate or *adapt* the stored example that most closely resembles the new situation.

1.2 Genetic Algorithms

Genetic algorithms are stochastic, parallel search algorithms based on the mechanics of natural selection, the process of evolution [Holland, 1975,Goldberg, 1989]. GAs were designed to search large, non-linear search spaces where expert knowledge is lacking or difficult to encode and where traditional optimization techniques fall short. They are flexible and robust, exhibiting the adaptiveness and graceful degradation of biological systems.

Genetic algorithms, like other search algorithms, balance the need for exploration — to avoid local optima, with exploitation — to converge on the optima. GAs dynamically balance exploration versus exploitation through the recombination (crossover and mutation) and selection operators respectively. When we seed a genetic algorithm's initial population with cases we alter this balance, subsequently affecting the search bias and thus the *kind* of solutions generated. We propose using a kind of associative memory — organizing information based on "similarity" — to help augment genetic algorithm search.

2 Combining GAs and CBR

Genetic algorithms provide an efficient tool for searching large, poorly understood spaces often encountered in function optimization and machine learning [Holland, 1975]. The probabilistic nature of GA search allows GAs to be successfully applied to a variety of NP-hard problems [Goldberg, 1989,Powell et al., 1989,Caldwell and Johnston, 1991]. The population-based, emergent nature of computation in GAs lends them a degree of robustness and flexibility that is often unobtainable using other approaches.

A genetic algorithm explores a subset of the search space during a problem solving attempt and its population serves as an implicit memory guiding the search. Every individual generated during a search defines a point in the search space and the individual's evaluation provides a fitness. This information when stored, organized, and analyzed can be used to explain the solution, that is, tell which parts of the genotype are important, and allow a sensitivity analysis [Louis et al., 1993]. However, this information is usually *discarded* at the end of a GA's run and the resources spent in gaining this information are wasted. If we store this information (explicit memory) and use it in a subsequent problem solving attempt on a related problem we can tune the GA to a particular space and thus increase performance in this space.

One early attempt at reuse can be found in Ackley's work with SIGH [Ackley, 1987]. Ackley periodically restarts a search in an attempt to avoid local optima and increase the quality of solutions. Eshelman's CHC algorithm, a genetic algorithm with elitist selection and cataclysmic mutation, also restarts search when the population diversity drops below a threshold [Eshelman, 1991]. Both approaches only attack a single problem not a related set of problems. Ramsey and Grefenstette come closest to our approach and use previously stored solutions to initialize the genetic algorithm's initial population and thus increase a genetic algorithm's performance in an environment that periodically changes with time [Ramsey and Grefensttete, 1993]. They report encouraging results in such environments (better performance). Our work however, tackles sets of similar problems and addresses the issues of which and how many cases to inject into the population. To our knowledge, the earliest work in combining genetic algorithms and case-based reasoning was done by Louis, McGraw, and Wyckoff who used case-based reasoning principles to explain solutions found by genetic algorithm search [Louis et al., 1993].

Figure 1 shows a conceptual view of a simple version of our system. There are two basic modules, the search module and the CBR module. The search module in this paper consists of a genetic algorithm and the CBR module for this work only contains a few cases relevant to the problem we are going to solve. When confronted with a problem, the CBR module looks in its case base for *similar* problems and their associated solutions. If any similar problems are found their solutions are injected into the initial population (a *small* percentage of the population is initialized this way) of the genetic (or other search) algorithm and the GA searches from this population.

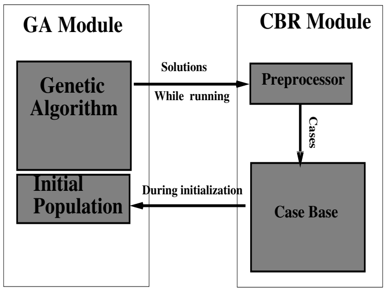


Fig. 1. Conceptual view of our system

The case-base does what it is best at — memory organization; the genetic algorithm handles what it is best at — adaptation. The resulting combination takes advantage of both paradigms; the genetic algorithm component delivers robustness and adaptive learning while the case-based component speeds up the system.

The genetic algorithm also provides a ready-made case generating mechanism as each individual generated during a GA search can be thought of as part of a case. However, even a population size of a 100 run for a 100 generations can generate up to $100 \times 100 = 10,000$ cases leading to problems with storage and indexing. CBR systems usually have difficult in finding enough cases; our problem is the opposite. We need to sift through a large number of cases to find potential seeds for the initial population. CBR systems often include principled mechanisms for generalization and abstraction of cases because it may be impractical to store all prior experience in detail. For our system, generalized individuals correspond to schemas (subsets of the search space) and can be stored as cases. This is described in [Louis et al., 1993]. Since genetic algorithms should be used in poorly-understood domains, one side-effect of this approach is that the generalizations and abstractions may in fact *induce* a useful domain model.

There are at least three points of view to consider when combining genetic algorithms and case-based reasoning.

- From the **case-based reasoning** point of view, genetic algorithms provide a robust mechanism for *adapting* cases in poorly understood domains conducive to a case-based representation of domain knowledge. In addition, the individuals produced by the GA furnish a basis for generating new cases to be stored in the case-base.
- From the point of view of **genetic algorithms**, case-based reasoning provides

a long term memory store in its case-base. A combined GA-CBR system does not require a case-base to start with and can bootstrap itself by learning new cases from the genetic algorithm's attempts at solving a problem. This paper stresses the genetic algorithm point of view and addresses the following questions.

(i) What cases are relevant, or, which previously found solutions do we use to seed a genetic algorithm's population? Selecting appropriate cases is important since it directly relates to speed and efficiency of a search. If the "right" cases are injected, a GA does not have to waste time exploring unpromising subspaces because these cases provide partial, near-optimal, or optimal solutions. In other words, they provide good building blocks for solutions to the current problem. This is crucial if the size of a search space is extremely large since injecting appropriate cases will provide the genetic algorithm with better starting points and thus speed up search and improve performance.

(ii) How many cases should we inject into the initial population? Search algorithms must balance *exploitation* with *exploration* for two reasons: 1) too much exploitation in less promising areas may cause the loss of crucial information on correct solutions and convergence to a local optimum, and 2) speed and efficiency will suffer if the algorithm wastes time in exploring unpromising areas. A randomly initialized population has maximum diversity and thus maximum capacity for exploration [Louis and Rawlins, 1993]. We expect injected individuals (cases) to have a higher fitness than randomly generated individuals. Since a GA focuses search in the areas defined by high fitness individuals, we expect large numbers of high fitness individuals in an initial population to increase exploitation. This increased concentration or exploitation of a particular area can cause the GA to get stuck on a local optimum. Balancing exploitation with exploration now means balancing the number of randomly generated individuals with the number of injected individuals in the initial population.

– From the **machine learning** pointing of view, using cases instead of rules to store information provides an alternative to Holland classifier systems [Holland, 1975,Goldberg, 19 which use simple string rules for long term storage and genetic algorithms as their learning or adaptive mechanism. In our system, the case-base of problems and their solutions supplies the genetic problem solver with a long term memory. The combination takes advantage of both paradigms but we have to be careful and ensure that the combined system does not inherit the deficiencies of both.

Our results on the open shop scheduling (OSSP) and re-scheduling problems (OSRP) establish the feasibility of this approach and shed light on some of the issues above and raise others to explore. In the next section, we describe a simple system to test the feasibility of combining genetic algorithms and case-based reasoning on OSSP and OSRP problems.

3 Methodology and Results

In our experiments, a genetic algorithm finds and saves solutions to an open shop scheduling problem P_{old}, the problem is changed slightly to P_{new}, and *appropriate* solutions to P_{old} are injected into the initial population of the genetic algorithm that is trying to solve the new problem P_{new}. If the cases from P_{old} contain good building blocks or partial solutions, the genetic algorithm can use these building blocks or schemas and quickly approach the solution to P_{new}. The results show that compared to a genetic algorithm that starts from scratch (from a randomly initialized population), the genetic algorithm with injected solutions very quickly finds good solutions to P_{new} and that the quality of solutions after convergence is usually better.

3.1 Open Shop Scheduling and Re-Scheduling

The open-shop scheduling problem is an important practical scheduling problem, but is known to be very hard to solve in a reasonable amount of time. Re-scheduling is also an important aspect of the problem in the real world. It involves modifying a schedule in the process of execution in order to take account of a changing environment. In the general $j \times m$ open-shop scheduling problem, there are j jobs and m machines. Each job contains a set of tasks which must be done on a different machine for a different amount of time, with no a-priori ordering on the tasks within a job. The problem is to minimize the makespan, the total time between the beginning of the first task till the end of the last task. A valid schedule is a schedule of job sequences on each machine such that a machine is not processing two different tasks at the same time, and different tasks of the same job are not simultaneously being processed on different machines. We must take into account both resources and time constraints with the aim of finding a valid schedule with a minimum makespan. The OSSP differs from the job-shop scheduling problem in that there is no *a-priori* ordering on the tasks within a job. This leads to an extremely large search space. For example on a 5×5 OSSP, there are 25! different task orderings which is approximately equal to 1.5×10^{25} different valid schedules.

In open-shop re-scheduling (OSRP), we need to be able to handle changing conditions on the shop floor. For example, jobs may be canceled, machines may fail, or we may run low on inventory. If the work has not yet begun on the current schedule, then a simple way of re-scheduling is to rerun the GA for this scheduling problem in the changed environment. We need quick and efficient methods of re-scheduling to tackle the problems of frequently changing environments. In this paper we handle the case of a machine breaking down and being replaced with a new, faster machine. P_{old} is the problem with the old machine while P_{new} is the problem with the new machine, where the processing times of all tasks on the new machine have been decreased by 10 time units; note that we have made sure that P_{new} is similar to P_{old}. We use individuals from a GA's run on P_{old} as cases for the re-scheduling problem, P_{new}. After using a variety of criteria to pick individuals for cases and to choose how many individuals to inject into the initial population of size 200 run for 300 generations, we obtained good performance under the following conditions (More details on the encoding and genetic operators are available in [Xu and Louis, 1996]).

(i) We inject only 8 cases into the initial population. Injecting between 5% and 10% of the population usually produced good results across the set of population sizes that we tried. Injecting a small percentage of the population establishes a good balance between exploration and exploitation of the search space. The injected individuals, if chosen well, represent good schema and help the GA quickly find promising solutions.

(ii) The individuals chosen as cases were:
  - The best individuals in the first and the last generations (Since we used a kind of elitist selection, the best individual in the last generation was the best ever).
  - Six good individuals in the generations between the first and last. For example, if we ran for 300 generations, then we pick the best individuals in generations which are a multiple of $300/6 = 50$.

We mutated these six individuals picked from intermediate generations and inject all eight individuals into the initial population of the GA for the OSRP. Figure 2 compares the average performance of a genetic algorithm initialized with these cases with a randomly initialized GA on two different 5×5 OSRP problems. We call the the GA that uses cases a **C**ase **I**nitialized **G**enetic **A**lgo**R**ithm or CIGAR, and the **R**andomly **I**nitialized **GA**, RIGA in the rest of this paper. The graphs in this section display the best makespan averaged over 10 runs with different random seeds. We can see that the CIGAR starts with a much lower (better) initial best makespan and that even after 300 generations CIGAR solutions are better than solutions from a randomly initialized GA. Table 1 compares starting and ending makespans for eight different 5×5 OSSP benchmark problems. [1] The last column specifies the machine that was changed (picked randomly). Notice that CIGAR does comparatively well in early generations and provides good schedules more quickly than the randomly initialized GA.

We get similar results on a set of five 7×7 OSRP benchmark problems. Figure 3 displays average best makespans over ten runs with different random

[1] All problems in this paper were obtained from the OR-library on the world wide web at the location: `http://mscmga.ms.ic.ac.uk/info.html`.

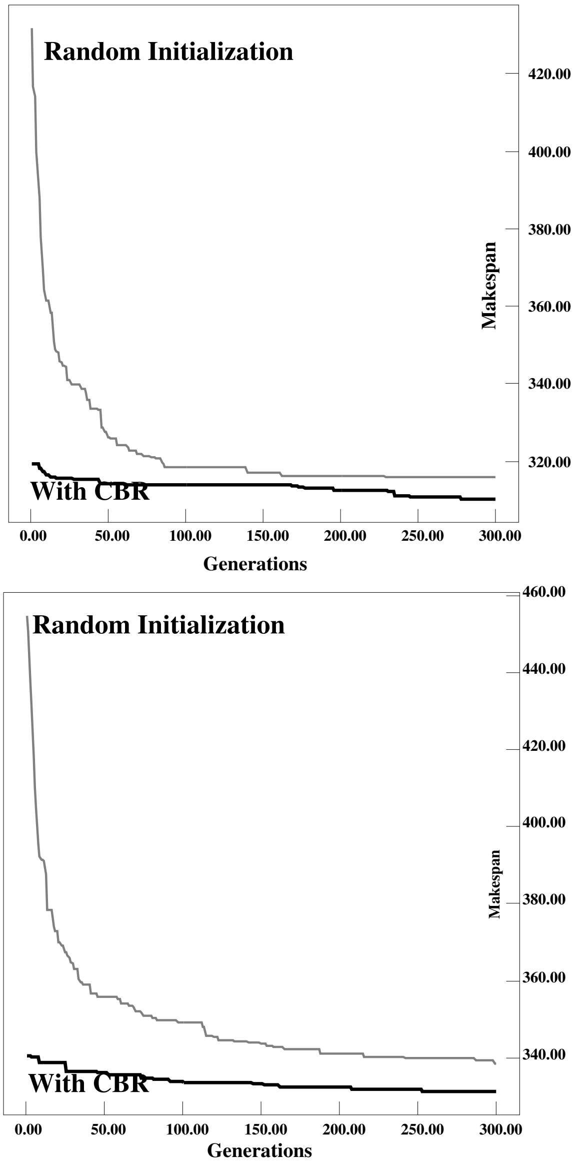


Fig. 2. Comparing performance of a RIGA with a CIGAR for 5x5 OSRP problems.

seeds on two of the problems. We increase the population size to 300 and run for 400 generations on the 7×7 problems. Once again the CIGAR does better, especially in early generations.

4 Discussion

In early experiments, we were not very successful in using case-based reasoning principles. First, when the environment was changed by deleting a whole job, the results were not good enough to show the advantages of using CIGARs. We found that it was hard for the GA with CBR to tackle problems where

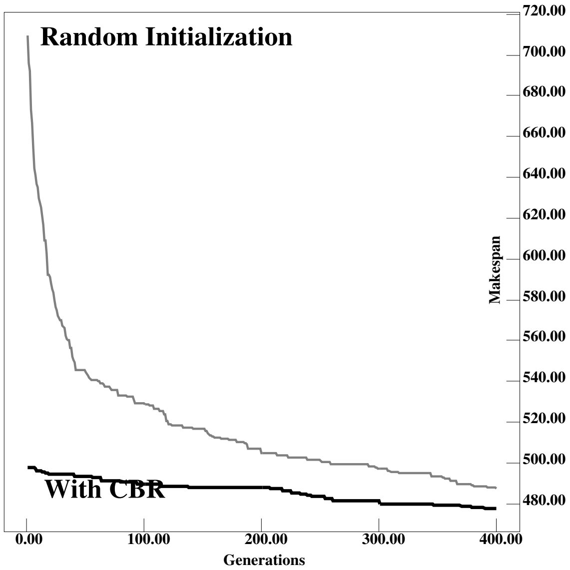
Table 1
Performance comparison of average makespan on 5x5 OSRP problems

Problem No.	CIGAR		RIGA		Machine Change
	Start	End	Start	End	
1	318	302	412	309	Change M2
2	281	271	369	269	Change M5
3	356	338	453	343	Change M4
4	337	326	438	328	Change M2
5	363	343	466	348	Change M1
6	342	328	436	333	Change M2
7	383	370	499	369	Change M5
8	357	346	475	347	Change M5

the environment changes significantly. CBR is based on the idea that stored cases resemble a new situation. Since a job contains a series of different tasks, after deleting a whole job, good schedules for the current problem may be very different from the previous one for our encoding.

We also failed to get good results when we tried to save all, half, or a quarter of the best individuals in the previous run (P_{old}) and insert them into the initial population for P_{new}. Saving and re-using too many good individuals makes the GA quickly converge to local optima. In other words, too many injected individuals tipped the balance towards exploitation of the search space with not enough exploration. We obtain enough exploration when we seed a small percentage of the population, that is, use only the two best strings and mutate the other six good intermediate strings from P_{old}.

We can explain these results if we think of solutions as forming the leaves of a tree. Adapting one solution into another now means backing up to a common parent and then moving down the second solution's branch. Injecting cases closer to a common parent on the tree shortens the path to a similar problem's solution since less backtracking is involved. The individuals generated by a genetic algorithm form such a tree with increasing schema order toward the leaves [Louis et al., 1993]. This first simple model thus indicates that increasing problem dissimilarity implies injecting cases of lower fitness corresponding to when the GA has fixed fewer positions in its population's genotypes (lower order schemas). Lower fitness correlates with time (generation number) and with a position closer to the root of our solution tree. Thus, if we don't have a good measure of problem similarity, we can inject a set of cases with different similarities (saved at different times during the search attempt on P_{old}) and let selection cull those cases that are not useful. This also explains why our



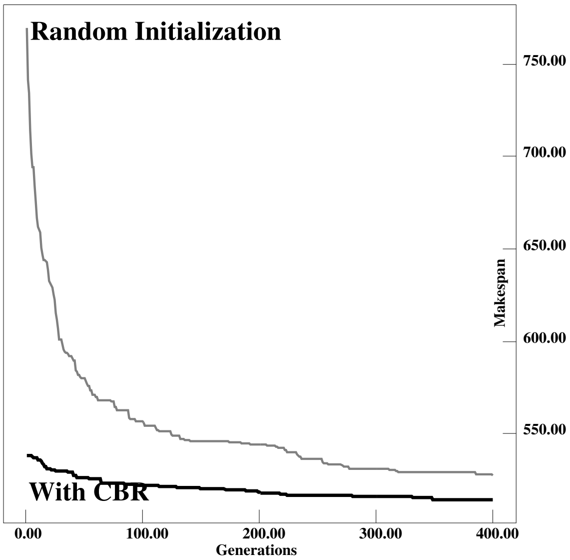


Fig. 3. Comparing performance of a RIGA with a CIGAR for 7x7 OSRP problems

method of choosing individuals to be injected works well. Since, we expect fitness to increase with time, choosing individuals from intermediate generations allows us to inject individuals with varying fitnesses into the the initial population. If P_{new} is very similar to P_{old} injected high fitness individuals (high fitness with respect to P_{old}) corresponding to those saved during later generations will probably augment genetic search. Other injected individuals will quickly die off. On the other hand if P_{new} is not very similar to P_{old} individuals saved from earlier generations will proliferate and help the search while others don't contribute as much.

The genetic algorithm's robustness lets us use a much coarser specification of problem distance. We only need to know that P_{old} is within a certain threshold distance from P_{new}, we do not need a more precise estimate. This threshold is problem dependent but an approximate distance measure is usually easier to obtain than an exact measure. In any case, if none of the injected individuals turn out to be useful, we would only have lost some time in evaluating these individuals before they die out, and would not expect to be worse off than starting with a completely random population.

These results broadly show that there is a tradeoff between performance, both in terms of speed and quality of solutions, and problem similarity. When more interested in quality, we should inject a small number of cases of varying quality into the initial population leading to a better balance between exploration and exploitation of the search space by the genetic algorithm. The CIGAR system always starts off better but the difference in performance usually decreases over time. The combined system should also be able to handle the injection of inappropriate cases — selection will quickly cull these from the population. We believe that injecting a large number of cases of varying similarity may prove counter-productive. The randomly initialized component of the population provides a much need diversity which, if diluted, will probably lead to quick convergence to local optima. More work needs to be done to resolve this issue.

5 Conclusions and Future Work

Our results demonstrate the feasibility of combining genetic algorithms with case-based reasoning principles to augment search. Instead of discarding information gleaned from previous problem solving attempts through search, we save and inject solutions to similar problems into the initial population of a genetic algorithm to increase performance. Our preliminary results indicate the feasibility and usefulness of this approach and show that choosing the right quantity and quality of cases plays a large part in determining performance. We obtained consistently better performance in terms of both speed and quality on the the open-shop re-scheduling problem.

Injecting a large number of high fitness individuals into the population usually does not produce good quality solutions over the long run as there is too much exploitation in the area defined by these injected individuals. Injecting too few, low fitness individuals also does not produce an advantage versus running the GA with a randomly initialized population. The "right" quality and quantity of individuals must be injected to realize performance gains. We find that the fitness of individuals to be injected depends on the distance between the old problem, to which we have solutions, and the current one. Problem size and the structure of the search space also play an important part in this issue. The results indicate an inverse relationship between problem distance and the fitness of injected individuals. Finally, although we can inject a larger number of individuals for larger problems, injecting a number of individuals between 5 to 15 percent of the population size provides good results over our problem sets. In the absence of information, these results suggest injection of a small percentage (depending on how quickly we want results) of cases of different fitnesses to cover the various possibilities in problem distance, solution distance, and search space structure.

We are currently working on implementing a pilot system that will gradually decrease the time taken to solve problems within a domain using genetic search and an expanding case base. As the system adds to the case base while attempting to solve problems the probability of finding similar solutions in the case base will increase and lead to a decrease in the time taken to solve a new problem. We will be forced to confront problems in indexing, storage, and retrieval of cases; important issues in designing CBR systems.

Although the approach has promise, much work remains to be done. We need to consider the effect of different selection schemes, recombination operators, and niching operators, for genetic search, as well different search algorithms. Deriving more refined estimates on the quality and quantity of individuals to inject will broaden applicability. Individuals need not be injected solely into the initial population. We can keep track of the performance of injected individuals and their progeny and use this information to design and inject individuals in intermediate generations. Finally, the tradeoffs between speed and solution quality needs to be explored in more detail.

Acknowledgement

The author gratefully acknowledges support provided by the National Science Foundation under Grant IRI-9624130.

References

[Ackley, 1987] Ackley, D. A. (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers.

[Bareiss, 1991] Bareiss, R., editor (1991). *Proceedings of the Case-Based Reasoning Workshop*. Morgan Kauffman, Inc.

[Caldwell and Johnston, 1991] Caldwell, C. and Johnston, V. S. (1991). Tracking a criminal suspect through "Face-Space" with a genetic algorithm. In *Proceedings of Fourth International Conference on Genetic Algorithms*, pages 416–421. Morgan Kauffman.

[Eshelman, 1991] Eshelman, L. J. (1991). The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms-1*, pages 265–283. Morgan Kauffman.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

[Holland, 1975] Holland, J. (1975). *Adaptation In Natural and Artificial Systems.* The University of Michigan Press, Ann Arbour.

[Louis et al., 1993] Louis, S. J., McGraw, G., and Wyckoff, R. (1993). Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:21–37.

[Louis and Rawlins, 1993] Louis, S. J. and Rawlins, G. J. E. (1993). Syntactic analysis of convergence in genetic algorithms. In Whitley, L. D., editor, *Foundations of Genetic Algorithms - 2*, pages 141–152. Morgan Kauffman, San Mateo, CA.

[Powell et al., 1989] Powell, D. J., Tong, S. S., and Skolnik, M. M. (1989). Engeneous domain independent machine learning for design optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 151–159. Morgan Kauffman.

[Ramsey and Grefensttete, 1993] Ramsey, C. and Grefensttete, J. (1993). Case-based initialization of genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California. Morgan Kauffman.

[Riesbeck and Schank, 1989] Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Cambridge, MA.

[Tong and Sriram, 1992] Tong, C. and Sriram, D. (1992). Introduction. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design*, pages 1–53. Academic Press, Inc.

[Xu and Louis, 1996] Xu, Z. and Louis, S. J. (1996). Genetic algorithms for open shop scheduling and re-scheduling. In *Proceedings of the ISCA 11th International Conference on Computers and Their Applications.*, pages 99–102, Raleigh, NC, USA. International Society for Computers and Their Applications.