The University of Kansas

Information and Telecommunication Technology Center

Technical Report

Combining Genetic Algorithms and Case-Based Reasoning for Genetic Learning of a Case Library: A Conceptual Framework

Leen-Kiat Soh

ITTC-FY2001-TR-19860-01

October 2000

Project Sponsor:
U.S. Air Force
Award Number: F30602-99-2-0502

Combining Genetic Algorithms and Case-Based Reasoning for Genetic Learning of a Case Library: A Conceptual Framework

Leen-Kiat Soh

Information and Telecommunications Technology Center (ITTC)
University of Kansas, Lawrence, KS 66044

Abstract

In this paper, we present a conceptual framework that uses genetic algorithms and case-based reasoning to create and maintain a case library and learn a genetic hierarchy of the cases. Our research objective is to equip an agent with a case-based reflective negotiation model so each agent is able to negotiate with its neighbors for resource and task allocations. The *brain* of an agent is a case-based reasoner. Due to the lack of actual cases, in order to design and test our agents in case evaluation and retrieval, adaptation, and storage, we propose to use a genetic algorithm to generate useful cases. In this paper, we present our work thus far on designing a viable genetic learning algorithm that not only measures the fitness of a case by itself but also the fitness of a case as a member of a library. In this paper, we discuss the fitness of measure and the synergetic cooperation between case-based reasoning and genetic algorithms for genetic learning. During the course of the research, we have introduced several new genetic learning concepts such as meta-genetic code, evolutionary adjustment, refinement, incompatibility, and breakthrough population migration, injection, and granularization, and deterministic mating.

1 INTRODUCTION

Our responsibility in the DARPA Autonomous Negotiating Teams (ANTS) research project is to equip agents with a reasoner. This reasoner should determine when to take actions and what those actions are, when to negotiate with a neighbor to perform a task, and what to do with computer resource allocation (such as computing time and disk space). The reasoner should also keep track of its own tasks, and monitor the real-time environment, and housekeep structures under its control. Our approach is to equip an agent with a case-based reflective negotiation model to help with the determination of task feasibility, selection of a feasible neighbor, the determination of the negotiation issues at the outset of a negotiation, and the guidance of the ensued negotiation behavior. Our domain of application is target tracking and identification. When a target is sensed in an environment covered by a group of cooperating sensors, the goal is to provide coverage as completely, efficiently, and as strategically as possible. An agent controls each sensor, and can interact with other agents of other sensors through communication channels. The approach is to enable the decision making be bottom-up and distributed to increase the response speed to a target and the reliability of the network of sensors.

One important research that we have encountered is the collection of cases for the case library (or case base). As pointed out in (Kolodner, 1993, pp. 547-555), there are three general approaches to collecting cases: (1) adaptation from existing databases, (2) outcome of an automated or interactive problem-solving system, and (3) knowledge acquisition from domain human experts. All three of these approaches are not viable for our domain. The actual database of an event (a target being spotted and tracked) is highly complicated and involved, with layers of command decisions and actions-reactions from various viewpoints. In addition, these events were not recorded in a way that resembles a multiagent, negotiation-based environment. Thus, the adaptation would require much research effort. To date, there is not yet a system that is able to solve the target tracking and identification problem in a multiagent infrastructure—that is the objective of the DARPA ANTS project—we cannot rely on the system to provide us with cases. Finally, knowledge acquisition has always been a bottleneck in knowledge engineering (Nwana *et al.*, 1991, Boose, 1993) and even more so in our domain since it is difficult for commanders to articulate traditionally centralized decision making process in terms of distributed decision making, and domain human experts are not readily available.

To obtain cases for the development and testing our agents case-based reasoning capability, we have thus turned to genetic algorithms (e.g., Goldberg, 1989). Genetic algorithms do not require as much domain knowledge in order to operate. We will be able to generate different cases based on a set of primitive case descriptors. Genetic algorithms are also dynamic such that we can modify, maintain, and generate new generations of cases without having to discard existing cases as our system progresses to handle more complicated scenarios. Our objective is to use genetic algorithms to learn useful cases for the development and testing, and refinement of our system.

During the course of the research, we introduce several new genetic learning concepts: meta-genetic code, fitness reduction and evolutionary incompatibility, population migration and evolutionary adjustment, population granularization and evolutionary refinement, and population injection and evolutionary breakthrough.

In this paper, we will first discuss some related work to using genetic algorithms in case-based reasoning. Then, in the third section, we present our methodology and algorithm for creating a case base using genetic algorithms. We will talk about the mapping between the phenotypes and the genotypes, the crossover, mutation, and migration. We will also address a variety of issues in the evaluation of the fitness of a case. Then, in the fourth section, we will discuss our domain and application for our work. Finally, we conclude.

2 BACKGROUND

Most literature in case-based reasoning puts more emphasis on case evaluation, retrieval, adaptation and storage as compared to case collection or the building of the case library (Aha, 1991, Kolodner, 1993, Aamodt and Plaza, 1994, Watson and Marir, 1994, Lopez de Mantaras, R. and E. Plaza, 1997). In research works where case collection were discussed, the emphasis was on case representation and indexing, with a raw form of each case available via adaptation from a database, from a working system, or from human expert knowledge. Our domain, therefore, offers a unique challenge to our case-based design, in that cases have to be generated, automatically.

One related work in automated case generation is that of (Flinter and Keane, 1995). The authors proposed a system called TAL that employed case-based reasoning techniques for chess playing. They concentrated on the automatic generation of suitable case knowledge using a chunking technique on a corpus of grandmaster games. However, in the work, the authors had a collection of game situations from which cases were generated. In our domain, such game situations are not feasible.

Maher and her group (Maher, 1994, Maher and Gomez de Silva Garza, 1996a, 1996b) have, on the other hand, pioneered the work in using genetic algorithms for case adaptation in the domain of structural design of tall buildings. The driving issue was that in order to adapt past experiences to new situations, case-based reasoning algorithms generally rely on domain knowledge and heuristics. The reliance required all adaptation scenarios be foreseen and recognized, which was infeasible due to its size and incompleteness. Therefore, they have explored the use of a knowledge-lean method based on genetic algorithms for the subtask of case adaptation and have applied their systems successfully. Maher's work demonstrates the viability of genetic algorithms in enhancing case-based reasoning.

In another related work, Purvis and Athalye (1997) also used genetic algorithms to improve case adaptation.

3 GENETIC LEARNING AND CASE GENERATION

Our focus is on case-based negotiation. Upon sensing a target, an ANTS agent decides to whether track the target itself or negotiate the task to a neighbor. When a negotiation is chosen, then the agent uses case-based reasoning to obtain the most suitable neighbor, to determine the initial negotiation parameters, and to guide the ensuing negotiation steps. Thus, a case includes four important areas: (1) profile and status, (2) negotiation parameters, (3) abstractions of the actual negotiation, and (4) outcome.

The profile and status include the following:

(a) The profile of the agent: the number of tasks currently being managed by the agent, how many negotiation sub-agents are currently active, whether the communication channel is busy, current usage of computer resource, etc.

(b) The profile of the sensor that the agent is in charge of: the capability of the sensor, which sensing sector is active, what and which targets are currently being tracked by the sensor, the battery power of the sensor, etc.

(c) The profile of the target in question: the direction and speed, etc.

(d) The profile of the neighbor for negotiation: a combined set of a subset of an agent's profile and a subset of a sensor's profile, a score of 'friendliness' and 'usefulness' of the neighbor from the viewpoint of the agent, etc.

(e) The status of the environment: high alert situation, battlefield situation, etc.

The negotiation parameters include the negotiation issues such as the urgency and priority of the negotiation, time limit on the negotiation, the negotiated parameter (time, computer resources, etc.), etc.

The abstractions of the negotiation include the number of interactive steps, the actual elapsed time, the amount of information passed, the negotiation behavior, utility of the negotiation (the difference between the initial offer/response and the last offer/response before termination), etc.

The outcome of the negotiation includes whether the negotiation was a success or a failure, and, if the negotiation failed, the reasons.

Also, an agent who initiates a negotiation has a different negotiation attitude from one who responds to a negotiation request. Thus, in our case library, we have two sets of cases: initiating and responding.

Note also our case evaluation and the best case selection algorithm is a multi-tiered evaluation stage: First, we compare the new case with the old along the profile and status values. Then we select *N* closest cases that pass a similarity threshold. For these *N* cases, we evaluate the abstractions of the negotiation. The selector favors negotiations that are short, with minimal information passing, with minimal number of steps, with minimal number of changes in the negotiation behavior, and with maximal utility. After this selection, if there is only one best case, then the selection is complete; if there is more than one, then we compare the outcome of the negotiation. We favor a success over a failure, and a failure with almost-there final negotiation status over a failure without fixable causes. The next subsection immediately discusses the phenotype-genotype mapping that we perform for converting our case descriptors into strings of 0s and 1s.

Then, we talk about the survival of a case vs. the archival of a case; that is, when should a case be promoted to the case library? Note that in traditional genetic algorithms, the search space is explored to find a solution to a problem. However, in our domain, we aim at obtaining many different cases to build our case library, so, as cases are being generated, we have to decide which deserves to be archived.

The following seven subsections next discuss the issues related to the measurement of fitness for each case. Whether a case survives to reproduce is determined by the case's fitness. Our research effort is now focusing on the issues of case incompleteness, noise tolerance, feature irrelevance, case forgetting, case distribution, and case evolution and maintenance.

One key feature of our approach to genetic learning for creating the case library is that it also considers the post-creation maintenance and enrichment of the case library. Note that after the development and testing period, the genetic learning is temporarily halted, as we will by then have obtained enough cases to cover most real cases (or cases generated by our system) as the system becomes operational. However, the hierarchy is still kept and will help in updating case fitness, extinguishing some cases, re-populating along some evolution lines, etc. So the genetic learning is always present, active during the creation of the case library and dormant as the system stabilizes.

3.1 Phenotype-Genotype Mapping

As known in the area of genetic algorithms, the phenotype describes a living organism for biological systems or a case, as in our research. The phenotype is the taxonomy of a case, profiles, status, negotiation parameters, negotiation abstractions, and the outcome. To be able to perform machine evolution efficiently, the phenotype has to be re-represented using the genotype. The genotype, usually, is a string of 0s and 1s, encoding the information that produces the phenotype.

In our case, binary descriptors can be directly translated into 0s and 1s.

For multi-valued descriptors, we perform feature extension. For example, a sensor has three sectors, and only one can be active. The feature ACTIVE_SECTOR thus has four possible values: 0 (no sector is active), 1, 2, or 3. To code this information in binary, we extend the feature ACTIVE_SECTOR to ACTIVE_SECTOR0, ACTIVE_SECTOR1, ACTIVE_SECTOR2, and ACTIVE_SECTOR3, where ACTIVE_SECTOR0 indicates whether there is an active sector. So, if a sensor's second sector is active, then the following string completely encodes the activity: 1010.
For variable-length features, we perform feature padding. For example, the outcome of a negotiation can be a success or a failure. If the outcome is a success, the case does not have any more related information to add. But if the outcome is a failure, then the case needs to document whether the

negotiation failed because (1) the time allocated for the negotiation ran out, (2) irreconcilable differences between the two negotiating parties, (3) the initiator decides to abort the negotiation due to some event, (4) the responder decides to abort the negotiation due to some event, (5) a communication channel failure, (6) one of the sensors became malfunction, (7) no response from the negotiating partner. We perform feature extension to convert the failure-related REASON to binary REASON_N fields. For a case that has a successful negotiation, we also have such fields, which will remain always as a string of 0s. Note that during the evolution, these strings will not change their values. This limitation on reproduction, mutation, and crossover during genetic learning can be ensured by always checking the SUCCESS value beforehand.

3.2 Survival vs. Archival

At each generation, we perform a fitness measure of each case. Only cases that are fit survive and can be used for the creation of the next generation. Among those cases that survive, we also perform an archival measurement to see whether a case should be promoted to the case library.

To do so, we evaluate the new *recruit* against the cases that are currently in the case library, following the comparison weights and features used in our case evaluation and the best case selection methods. If the new recruit is deemed unique, then it is promoted.

After a case has been promoted, its fitness in the evolution is decreased. The decrement is based on the uniqueness of the case, previously calculated when we evaluate the new recruit. The more unique a case is in the case library, the more the fitness of the case is reduced, and vice versa. This is to ensure that unique cases can remain unique and representative, thus increasing their utility in the case-based reasoning process.

As will be discussed in subsections 3.5, 3.6, and 3.7, the fitness of a case can be strengthened based on the case-utilization statistics. For example, if a case *A* is found to have been retrieved as the best case at a high frequency and the average similarity between the new case and that case *A* is relatively low, that means the case *A* is being stretched out to cover other not-so-similar cases. The fitness of this case is then increased, increasing its chance of reproduction in the genetic evolution.

3.3 Incomplete Cases

To deal with cases with incomplete information, we introduce the *meta-genetic code*. Each case thus has two codes, the meta-genetic and the original genetic code. The meta-genetic code specifies which feature or descriptor value is missing, 0 for absent, and 1 for present. Each code undergoes its own reproduction, mutation, and crossover. Note that the main evolution hierarchy is still based on the original genetic code. The meta-genetic code is generated for a new case by combining the new case's parents' meta-genetic codes. Hence, the meta-genetic code does not affect the growth directly. However, the meta-genetic code does influence the uniqueness of a case in a case library, which affects the fitness of the case in the evolution hierarchy.

The innovation of meta-genetic coding allows the population to evolve along with the case library. It enhances the practicality of the cases generated (since information incompleteness is present in our domain) and indirectly ties the fitness of a case based on partially its incompleteness.

As the case library becomes more complete as the system becomes operational in the future, cases with more complete information will replace cases with incomplete information. If a case is replaced in the case library in this manner, then, in the genetic hierarchy, its fitness stays the same but its meta-genetic code is replaced.

3.3 Uncertain Cases

Similarly, in our domain, we have to deal with uncertain information. Sensors attach uncertainty in what they detect. Thus, in addition to the absolute meta-genetic code, we are also exploring the possibility of a string of uncertain meta-genetic code. Each feature or descriptor value will be modified by a binary code CERTAINTY with low (0) and high (1) as the values. The usage of this uncertain meta-genetic code will be similar to that for incomplete information, discussed in the above section.

3.4 Noisy Cases

To inject noise into evolution hierarchy and to the case library to increase the robustness and coverage of the library, we randomly toggle *q* bits of a genetic code of a case, after it has been promoted to the case library. The noise injection will only be carried out periodically during the lifecycle of the case library. Note that we do not corrupt the evolution directly since, in a way, the evolution is noisy at its own right. We do corrupt the case library, which indirectly affects the evaluation of the uniqueness of a new recruit for the next generation.

3.5 Forgetting Cases and Evolutionary Incompatibility

As the case library becomes more and more complete and the system stabilizes, we compute the case-utilization statistics of the cases in the library. Cases that are used often will have high utility; cases that are used in high-priority, high-impact tasks will have high utility.
If a case has a very low utility, then it will be discarded from the case library and be forgotten. Correspondingly, the case's fitness measure will be lowered, with a degree associated to how much its utility measure is below par.

We call such a *fitness reduction* a situation of *evolutionary incompatibility*. A genetic code becomes less fit due to its incompatibility to its environment, in this case, the applicability of the case in a very specific domain of multiagent, negotiation-based target sensing and tracking.

This fitness reduction concept has a greater implication. We believe that as the system stabilizes, the genetic evolution will learn to specialize itself such that the fitter portion of the population in the hierarchy of generations consists of members with high utility in the case library. In this way, we can build different hierarchies for different tasks and applications of the same domain.

3.6 Case Distribution

To deal with representative cases to reduce storage space and to avoid degradation in case evaluation and retrieval (the speed deteriorates as number of cases in the case library increases), we must pay attention to the case distribution. In addition to the aforementioned concern, we consider the adaptation. If a case is being retrieved very frequently and non-trivial adaptation has been performed for each retrieval, that means, either (1) the case in the library has to be revised if the matched cases have concentrated more or less at a single focus, or (2) the coverage of the case is being strained and we need more similar yet different cases to support the coverage. If the former occurs, we call the situation a case-shifting or a *population migration* due to *evolutionary adjustment*. If the latter occurs, we call the situation a *population granularization* due to *evolutionary refinement*.

3.6.1 Population Migration and Evolutionary Adjustment

A population migration occurs when some of its features need to be modified (or a case needs to be shifted along some dimensions). When such modification is necessary, we will have by then a group of closely similar case examples. We first compute the representative, imaginary core case of the group. We then replace the case in question with this core case. As discussed in subsection 3.5, the fitness of the replaced case will be reduced in the hierarchy.

Now, to perform the evolutionary adjustment, we first attempt to match the core case to a case in the hierarchy. If such a matching is present, we upgrade the fitness of the found case in the hierarchy, and the adjustment is complete. However, if such a matching cannot be identified, then we search for a pair of suitable genetic codes to perform a deterministic mating. The pair of codes must be able to permute to arrive at the exactly code string of the core case. If found, we add the core case to the hierarchy and give it a high fitness measure. If the *parent* search fails, we find the nearest case in the hierarchy tree and mutate it towards the core case, and perform a parent search, and repeat the mutation and search until success.

This evolutionary adjustment is a very useful concept in narrowing the search in the overall genetic learning phase. It is as if we have observed the outcome of a mating (that happened some generations ago) and that outcome has proven to be very useful in our domain, and thus we are trying to perform archaeology to locate the missing links between last known, closest genetic code and the current ones. This enriches the evolution hierarchy.

3.6.2 Population Granularization and Evolutionary Refinement

A population granularization occurs when it has become costly in adapting cases from a retrieved case *A* when the retrieved case *A* is frequently retrieved. Note that before a population granularization is decided, one must weigh the additional cost of evaluating and retrieving more cases as a result of the granularization and the saved cost of constantly adapting cases from the case in question. If the latter outweighs the former, then one can proceed with the granularization. Usually, when this is observed, that means the case *A* is being stretched out to cover too much space in the case base.

We perform granularization conservatively. Out of a group of new cases (that surround the original case), we pick the farthest new case from the original case as the case to add to the case library. We reflect that addition in the genetic hierarchy via evolutionary refinement.

The evolutionary refinement attempts to refine the resolution of a lineage by adding a new branch. First, we increase the fitness of the original case, based on the number of members in the group of new cases that surround it. This gives it a stronger chance in its descendants being fit and getting promoted to the case library. Second, we perform a similar parent search and deterministic mating for the farthest new case. The only difference is that we narrow the search to the descendants and permutations of the original case, since we are trying to refine that particular section of the hierarchy.

3.7 Population Injection and Evolutionary Breakthrough

In our approach to genetic learning, no cases can become extinct. They can, however, become unfit and can stop taking part in the evolution temporarily until a new case or new blood is injected into the hierarchy.

A population injection occurs when a new case, which has a very low similarity to other existing cases in the case library, is found and has a high utility. Such a new case is added to the case library, and will be introduced into genetic hierarchy according to what we term as an evolutionary breakthrough.

This new case is thus added to the hierarchy and subsequent reproduction, mutation, and crossover followed. The special characteristic of the breakthrough is that this case will mate with every final descendant of every branch of the hierarchy, making its presence felt. All children will then be evaluated and those deemed fit will be recruited to the case library.

This evolutionary breakthrough is an aggressive search since it sees this new blood as a refreshing addition to the hierarchy and attempts to populate its branch as explosively as possible to bring it up to par with other established branches.

3.8 Fitness Propagation

After the fitness of a case is modified, the effect must be propagated down to its descendants. Here, we do not propagate the change up to its ancestors.

If the fitness of a case has been decreased, that means the case has become less fit in the current environment. Since all generated descendants cannot be de-generated, we simply propagate the effect down to reduce the fitness of each descendant. If a descendant has previously been promoted to the case library, we double-check the reduction with the case library. If the case has very low utility, then we demote the case. If the case has very high utility, then we stop the propagation along that branch at that point.

If the fitness of a case has been increased, that means the case has become more fit in the current environment. Then, there are two lines of actions to take. First, we update the change down to all its descendants. For all descendants that have become fit, we send them for a possible recruit in the case library. Second, depending on the increase, we also perform a certain amount of growing at each affected node to compensate missed opportunities when the branch was dominated by more fit cases.

3.8 Fitness Measure

As we have mentioned in previous discussions, every new child (or case) is evaluated to obtain a measure of fitness. We then perform further generation on fit-enough children and also send fit-enough children to the case library for possible promotion. A promotion is based on the uniqueness of a case in the case library. Then, the fitness of a case in the hierarchy can change due to evolutionary incompatibilities, adjustments, refinements, and breakthroughs.

First, the fitness of the parents or parent is reflected in the child's fitness measure.

Second, our agent is modeled after a case-based reflective negotiation model. Thus, our objective is to have an agent that is self-aware, knows about its neighbors sufficiently, and can perform negotiation efficiently. To be highly self-aware, an agent should know about itself, i.e., the profile of itself and the sensor under its control. To be situational-aware, an agent should also know about its neighbors, i.e., the profiles of its neighbors. A good self-awareness and neighborhood awareness means that the agent can make better and more informed decisions. Finally, in order to perform negotiation efficiently, we look to the negotiation parameters that guide a negotiation. As mentioned before, these parameters are derived from the profile and status of the agent. To be efficient, first, these parameters should be able to be derived quickly from the available knowledge, and should be determined as certainly as possible. Thus, we are now modeling the derivation. Some values of some features (or descriptors) complicate the derivation. For example, if the profile of an agent does not suit the requirement of a task, that means the negotiation parameters will not be as certain as otherwise. Meanwhile, some values of some features (or descriptors) simplify the derivation. For example, if a task is extremely urgent, then that means the initiator of the negotiation is extremely yielding. Our derivation is rule-based, as we have built an interface to CLIPS. Thus, the difficulty measure of a derivation can be as simple as the number of rules fired to arrive at a set of negotiation parameters.

Third, as discussed in Section 3, an agent favors negotiations that are short, with minimal information passing, with minimal number of steps, with minimal number of changes in the negotiation behavior, and with maximal utility. Thus, a case is more useful it has some of the above characteristics.

Fourth, a case is more useful if it results in a success instead of a failure. A case is also useful, in a failure, if there is a good explanation for its failure.

Hence, the measure of fitness of a child (or a case) is a weighted combination of (1) parental fitness, (2) self-awareness, (3) neighborhood awareness, (4) derivation difficulty, (5) quality of the negotiation abstractions, and (6) outcome of the negotiation.

There is a significantly important idea here. When we generate cases, how can we say whether the negotiation is a success or a failure since there is not any negotiations taking place? Remember that our genetic approach is a learning approach. It updates the genetic hierarchy based on the *field observations* when the agents start to operate and negotiate with each other. The case library will undergo refinement and modification, which will affect the genetic hierarchy. An initial genetic case will be either replaced (due to evolutionary incompatibility, section 3.5), or migrated (due to evolutionary adjustment), or supported with additional, more accurate cases (due to evolutionary refinement). Thus, the correct code of a useful case will ultimately be learned and be represented in both the genetic hierarchy and the case library.

4 DISCUSSION

We employ the genetic learning approach not only to create the initial case library but also to help maintain it. It is an integral part of the whole system. It is also useful for monitoring the growth of the case library and the hierarchy.

During the creation of the case library, the genetic hierarchy serves as a bootstrapping basis. It is bound to provide some seemingly fit cases to the case library, and some cases are bound to be accepted into the library, albeit with highly unlikely genetic coding. For example, two highly unyielding agents should not come to an agreement very quickly. However, in our initial generation, we do not impose a rule to prevent such an offspring. However, if the offspring ever makes it to the case library, it will eventually be discarded since its utility will be extremely low and the original offspring will suffer a fitness reduction.

One critical issue is when to perform genetic learning. Since our application requires real-time response and reasoning, when a new case presents itself, we do not have the luxury to perform the aforementioned steps to adopt the case into the genetic hierarchy since the computation is non-trivial. Thus, we are now looking into separating the genetic learning from the case-based reasoning. First, all new cases will be added to the case library. After a period of operating without referring to the genetic hierarchy, a housekeeping module is invoked to perform the genetic update. Or, whenever the agent is idle and the case library has seen significant changes, housekeeping is performed. The housekeeping module will then inspect the case library and sequentially update each case to the hierarchy.

Another issue is that, in our multi-agent environment, each agent maintains its own case library and thus its own genetic hierarchy. In the beginning, each agent will be given the exactly same hierarchy. But as the system operates, each agent evolves and specializes. Therefore, we expect to see interesting differences among the hierarchies at the end of some time of running the system.
To date, we have implemented the case-based reasoning module of an agent and built the infrastructure of the multi-agent system that facilitates profiling, communication, thinking, negotiation, and task management. We are currently working on populating the case library with cases generated via genetic learning.

5 CONCLUSIONS

We have proposed and described a conceptual framework for a genetic learning approach to create and maintain a case library for our domain of multi-agent, negotiation-based task and resource allocation problem. This learning approach combines case-based reasoning and genetic algorithms to analyze the fitness of a case and the evolution of a genetic hierarchy.

Our focus is on combining the strengths of case-based reasoning and genetic algorithms to create a case library that is easy to maintain and populate. In the beginning, the genetic hierarchy will provide the main population of the library since there is a lack of cases. This initial case library will help us develop and test the case evaluation, retrieval, adaptation, and storage processes of our design. As the system becomes operational, the simulated scenarios or actual events encountered will in turn become the main force behind the evolution. It will provide opportunities for re-organization, re-evaluation of the genetic hierarchy, thereby creating new possibilities that will re-stock the case library.

During the course of the above research, we have identified several new genetic learning concepts such as meta-genetic code, fitness reduction, evolutionary adjustment, refinement, incompatibility, and breakthrough population migration, injection, and granularization, and deterministic mating.

References

Aamodt, A. and E. Plaza (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches, *AI Communications*, **7**(1):39-59.

Aha, D. W. (1991). Case-Based Learning Algorithms, *Proceedings of the 1991 DARPA Case-Based Reasoning Workshop*, 147-158.

Boose, J. H. (1993). A Survey of Knowledge Acquisition Techniques and Tools, in *Readings in Knowledge Acquisition and Learning*, Buchanan, B. G. and D. C. Wilkins (eds.), San Mateo, CA: Morgan Kaufmann, 39-56.

Flinter, S. and M. T. Keane (1995). On the Automatic Generation of Case Libraries by Chunking Chess Games, *Proceedings of the 1st Int. Conf. on Case-Based Reasoning*, Sesimbra, Portugal, Oct 23-26, 421-430.

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

Kolodner, J. (1993). *Case-Based Reasoning*, San Mateo, CA: Morgan Kaufmann.

Lopez de Mantaras, R. and E. Plaza (1997). Case-Based Reasoning: An Overview, *AI Communications,* **10**:21-29.

Maher, M. L. (1994). Creative Design Using a Genetic Algorithm, *Computing in Civil Engineering*, ASCE, 2014-2021.

Maher, M.L. and Gomez de Silva Garza, A. (1996a) Design Case Adaptation Using Genetic Algorithms. In J.Vanegas (ed.). *Computing in Civil Engineering,* ASCE.

Maher, M.L. and Gomez de Silva Garza, A. (1996b) The Adaptation of Structural System Designs Using Genetic Algorithms, *Proceedings of the Int. Conf. on Information Technology in Civil and Structural Engineering Design--Taking Stock and Future Directions*, Glasgow, Scotland. August 1996.

Nwana, H. S., R. C. Paton, T. J. M. Bench-Capon, and M. J. R. Shave (1991). Facilitating the Development of Knowledge-Based Systems: A Critical Review of Acquisition Tools and Techniques, *AI Communications*, **4**(2/3):60-73.

Purvis, L. and S. Athalye (1997). Towards Improving Case Adaptability with a Genetic Algorithm, *Proceedings of the 2nd Int. Conf. on Case-Based Reasoning (ICCBR'97).*

Watson, I. and F. Marir (1994). Case-Based Reasoning; A Review, *Knowledge Engineering Review*, **9**(4):327-354.