Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, 2001

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

Leen-Kiat Soh
Information and Telecomm. Technology Center
University of Kansas
2335 Irving Hill Road
Lawrence, KS 66045
lksoh@ittc.ukans.edu

Costas Tsatsoulis
Department of Electrical Engr. and Computer Science
University of Kansas
2335 Irving Hill Road
Lawrence, KS 66045
tsatsoul@ittc.ukans.edu

Abstract

In this paper, we present a conceptual framework that combines genetic algorithms and case-based reasoning (CBR) to first learn a genetic hierarchy of cases and then maintain and refine the casebase as the system runs. We propose to use genetic algorithms to generate useful cases since there is not any actual cases to bootstrap our CBR module. We use these evolved cases to develop and test the various stages of the CBR module such as evaluation and retrieval, adaptation, and learning for refining the module. We propose a fitness measure of a case that is based on not only the combination of its attribute values, but also on it being a member of the casebase, which involves its utility in the CBR module. To promote the synergy between CBR and genetic algorithms for genetic learning, we propose and describe several concepts such as meta-genetic code, evolutionary adjustment, refinement, incompatibility, and breakthrough, population migration, granularization, and injection, and deterministic mating.

1 INTRODUCTION

One important research issue in building a case-based reasoning system is the collection of cases for the casebase. 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. These approaches are not practical for our problem domain. Our problem domain is distributed resource allocation and constraint satisfaction. Particularly, the application goal is multiagent target tracking in a real-time and dynamic environment. An agent negotiates with its neighboring agents for resource sharing and task collaboration. 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 multi-agent, 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 in a bottom-up, multi-agent architecture, and we may not rely on an existing system for 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 military commanders (in charge of multisensor target tracking) to articulate traditionally centralized decision making process in terms of distributed decision making, and, as a result domain human experts are not readily available. Therefore, we are not able to collect cases via traditional mechanisms.

We turn to genetic algorithms (e.g., Goldberg, 1989) for generating good cases for our casebase. Genetic algorithms do not require as much domain knowledge in order to operate, compared to CBR that work directly with the system. 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.

The goal is to use genetic algorithms to generate a hierarchy of cases first and use CBR to maintain the casebase, which in turn refine the hierarchy. At the boostrap stage, fit-enough cases are sent to the CBR module for a promotion, and high-utility cases are eventually stored in the casebase. Then, as the system runs, new cases are encountered and added to the casebase. The activity triggers different evolutionary operations and population movements both in the casebase and ultimately in the genetic hierarchy. In this manner, the hierarchy is refined: new cases are evolved and promoted, and old cases are forgotten. In a way, the genetic hierarchy is the nursery for offspring; the casebase becomes the work environment where offspring are tested for their usefulness. In addition, the CBR is able to encounter new cases itself and inject those into the nursery. This synergy of CBR and genetic algorithms involves 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 first describe our problem domain furher. Then we discuss some related work on genetic algorithms in case-based reasoning. Then, in the fourth section, we present our concepts for genetic learnig nand case generation. We address a variety of issues from phenotype-genotype mapping to fitness measure. Then we discuss some issues stemming from the conceptual framework. Finally we conclude.

2 PROBLEM DOMAIN

Our problem domain is real-time resource allocation and constraint satisfaction. In the environment, there are sensors and targets. The objective is to track as many targets accurately as possible. However, resources are limited; the communication channels, the computing power, the CPU allocation, and other facilities are limited and have to be shared. In addition, sensors are constrained in coverage. They have to cooperate in a time-critical fashion in order to measure the target from different locations to obtain accurate triangulation readings for the target's position and velocity. The domain also calls for a strict bottom-up, distributed decision making hierarchy for robustness and reactiveness. That is, there is not any master agents that schedule tasks and manage resources for subordinate agents. Each agent has to negotiate with its neighbors to convince them to give up CPU resource, share communication bandwidth, turn on their sensing sectors, and other tasks. Each agent is intelligent, situation aware, autonomous and reflective. Each is capable of non-trivial deliberation and is also able to react quickly to unexpected events such as an incoming target, or a CPU allocation shortage. One requirement is that due to the time-criticality, negotiations have to be performed quickly and efficiently.

In our approach, we use an argumentative negotiation model in which the initiating agent tries to convince the responding agent to perform a requested task by supplying arguments during negotiation. In this model, each agent is motivated to optimize its local resource but to also maximize the global goal of target tracking. Hence, a negotiating agent needs to know when to abort and give up on a negotiation, when to agree to a request, what (and in what order) arguments to send over, how much CPU resources to use, etc. The negotiating agent encapsulates these decision points in a negotiation strategy. Instead of deriving the negotiation strategy from scratch for each negotiation, we use CBR. The CBR module collects the current status data from the monitoring modules of its agent, retrieves the most similar case from the casebase, adapts the solution (i.e., the negotiation strategy) to the current situation, and conducts the subsequent negotiation accordingly.

3 BACKGROUND

Most literature in case-based reasoning puts more emphasis on case evaluation, retrieval, adaptation and storage as opposed to case collection or the building of the casebase (Aha, 1991, Kolodner, 1993, Aamodt and Plaza, 1994, Watson and Marir, 1994, Lopez de Mantaras and 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 and Gomez de Silva Garza, 1996) 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.

4 GENETIC LEARNING AND CASE GENERATION

In general, a case constists of three sections: the situation or problem space, the solution space, and the outcome. In what follows, we describe these sections for our particular multiagent case-based negotiation application.

The situation space includes the profile and status of the current situation that an agent experiences. It includes the profile of the agent such as the number of tasks currently being managed by the agent, the number of currently active negotiation threads, whether the communciation channel is busy, the current usage of ocmputer resource, etc. It also includes the profile of the sensor that the agent manages such as the specifications (range, gain, sampling size, orientation, and position), the active sensing sector, the battery power, etc. It also includes the velocity and position of the target. It also describes the negotiation neighbor such as the past history of the negotiations between the two agents, the helpfulness of the neighbor, the usefulness of the neighbor, etc. Finally, it may also include the status of the environment such as high alert situation, battlefield situation, etc.

The solution space defines the negotiation strategy. A negotiation strategy determines the number of interaction steps, the time allocated, and the CPU allocated. For an initiating case, the solution space also includes a ranking of information pieces to be used as arguments by the initiating agent, dictacting how the arguments are to be sent. For a responding case, the solution space instead includes a persuasion function and its associated parameters and a persuasion threshold used by the agent to judge the evidence support of the received arguments.

The outcome section documents the outcome of the negotiation. If the outcome was a failure, then the reason for the failure is also recorded. It may also include the number of actual interactive steps, the actual elapsed time, the amount of information passed, the utility of the negotiation (the difference between the initial offer/response and the final agreement, or the difference between the initial offer/response and the last offer/response before termination), etc.

Before moving further, we briefly describe three features of our case-based negotiation model here:

First, each agent has two sub-casebases in its casebase—one for initiating a negotiation and one for responding to a negotiation.

Second, our proposed case evaluation and the best case selection algorithm is multi-tiered. First, we compare the new case with the old along situation space. Then we select *N* most similar cases that pass a similarity threshold. For these *N* cases, we evaluate the outcome space. The selector favors negotiations that are short, with minimal information passing, minimal number of steps, minimal number of changes in the negotiation behavior, and maximal utility. After this selection, if there is only one best case, then the selection and retrieval is complete. Otherwise, we compare the cases along 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 repairs.

Third, in traditional genetic algorithms, the search space is explored to find a fit solution to a problem, but, in our domain, we desire to obtain many different cases to build our case base. Our approach also considers the post-creation maintenance and refinement of the casebase. After the development and testing period, the genetic learning is temporarily halted, as we will by then have obtained enough cases to cover adequately some real situations encountered by the system as it becomes operational. However, the hierarchy is still kept and will help in updating case fitness, forgetting some cases, re-populating along some evolution lines, etc. So the genetic learning is always present, active during the creation of the casebase and dormant as the system stabilizes.

4.1 PHENOTYPE-GENOTYPE MAPPING

In our problem domain, the phenotype is the taxonomy of a case, profiles, status, negotiation parameters, and the negotiation outcome. To perform machine evolution efficiently, we re-represent the phenotype with the genotype, which is usually a string of 0s and 1s, encoding the information that produces the phenotype.

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

For multi-valued attributes, we perform feature extension. For example, a sensor has three sectors, and only one can be active. The feature *active_sector* 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 attributes, 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.

4.2 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 casebase, the more the fitness of the case is reduced. 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 4.6, 4.7, and 4.8, the fitness of a case can be strengthened based on the usage statistics of the case. 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 the 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.

4.3 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 attribute 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 eventually affects the fitness of the case in the evolution hierarchy.

The use of meta-genetic coding allows the population to evolve along with the casebase. 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 casebase 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 casebase in this manner, then, in the genetic hierarchy, its fitness stays the same but its meta-genetic code is replaced.

4.4 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 also explore 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.

4.5 NOISY CASES

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

4.6 FORGETTING CASES AND EVOLUTIONARY INCOMPATIBILITY

As the casebase becomes more complete and the system stabilizes, we will be able to compute the usage statistics of the cases in the casebase. 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 (close to zero) utility, then it will be discarded from the casebase 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.

This fitness reduction concept has another 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 casebase. In this way, we can build different hierarchies for different tasks and applications of the same domain.

In addition, case forgetting in this framework is less risky. When a case is forgotten, it is removed from the case base but still remains in the genetic hierarchy with a low fitness measure. It may still be of use as new cases are introduced and it may evolve and mate to adapt to the new environment and its offspring may then be promoted to the casebase. This allows the CBR module to manage dynamically its casebase according to the environment the system observes efficiently and without losing the knowledge that it has once learned and may become useful again in the future.

4.7 CASE DISTRIBUTION

To obtain representative cases to (1) reduce storage space, (2) avoid degradation in case evaluation and retrieval (e.g., the speed deteriorates as the number of cases in the casebase increases), and (3) conduct accurate strategy adaptations, we must pay attention to the case distribution. If a case has been retrieved very frequently and non-trivial and arduous adaptation has been performed for each retrieval, then that means, either (1) the case in the casebase 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*.

4.7.1 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.

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 combine 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 this *dual-parent* search fails, we find the nearest case in the hierarchy tree and mutate it towards the core case, and perform a *uni-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 offspring of a mating or a mutation that happened some generations ago, and that offspring 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.

4.7.2 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 casebase.

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 casebase. 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.

4.8 POPULATION INJECTION AND EVOLUTIONARY BREAKTHROUGH

In our approach to genetic learning, no cases may become extinct. They may, however, become unfit and 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 casebase, is encountered and has a high utility. Such a new case is added to the casebase, and will be introduced into the 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 immediately follow. The special characteristic of the breakthrough is that this case will mate with as many final descendant of every branch of the hierarchy as possible, making its presence felt. All children will then be evaluated and those deemed fit will be recruited to the casebase. Since we see this new case as new blood to the population, we do not attempt to perform dual-parent or uni-parent searches to find its ancestors in the existing hierarchy.

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 explosively to bring it up to par with other established branches. This breakthrough may contaminate the population and, hence, the injection should not be done too often.

4.9 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 reduced, then that means the case has become less fit in the current environment. Since all generated descendants cannot be degenerated, we simply propagate the effect down to reduce the fitness of each descendant. If a descendant has previously been promoted to the casebase, we double-check the case to see whether it is still worthy to remain in the casebase. If the case has a very low utility value, then we demote the case. If the case has a very high utility value, then the case remains and we stop the propagation along that branch at that point.

If the fitness of a case has been increased, then that means the case has become fitter in the current environment. Two lines of actions follow. First, we update the change down to all its descendants. For all descendants that have consequently become fit, we send them for a possible promotion to the casebase. Second, depending on the increase, we also perform a certain amount of evolving at each affected node to compensate the missed opportunities when the branch was dominated by other fitter cases.

4.10 FITNESS MEASURE

As we have mentioned in previous subsections, 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 casebase for possible promotion. A promotion is based on the uniqueness of a case in the casebase. A demotion is based on the utility or the usage statistics of a case in the casebase. Then, the fitness of a case in the hierarchy can change due to evolutionary incompatibilities, adjustments, refinements, and breakthroughs. There are two types of fitness measurements: (1) *intrinsic fitness*—measuring the case by itself, and (2) *environmental fitness*—measuring the case by its membership in the casebase. This enables a very useful synergy between our genetic learning mechanism and our CBR module.

4.10.1 Intrinsic Fitness

The proposed intrinsic fitness is computed for each newly generated child in the genetic evolution phase. It includes the following:

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

Second, our agent is based on a case-based reflective negotiation model. To be reflective, an agent should know about itself, i.e., the profile of itself and the sensor under its control, the resources that it has, the neighbors, and the world. The information translates into arguments that an initiating agent uses to convince its counterpart, and into persuasion values that a responding agent uses to evaluate its counterpart's arguments. How good the arguments are and how cooperative the persuasion values are depends on the specific combinations of status and profiles that he agent has of itself, its neighbors, and the world. Thus, the fitness of a case is also measured by the status and profiles of the agent in terms of the strength of the arguments that they produce and the cooperativeness of the persuasion values that they produce. In addition, we also infuse local and global goals in the mix. Cases that optimize local resources are favored. Cases that drive towards the global objective of target tracking are favored. For example, a case where the initiating agent is extremely busy and the responding agent is relatively idle is fit because it would make the arguments from the initiator side strong and the persuasion threshold on the responder side relaxed. A case that has three potential partners for an initiating agent to look for help is favored because that means it is more likely to have a three-sensor triangulation if the negotiation is successful. These fitness evaluation steps are encoded in domain-specific and common sense heuristics and are the cornerstone of the intrinsic fitness measure.

Third, as discussed earlier in Section 4, an agent favors negotiations that are short, with minimal information passing, minimal number of steps, minimal number of changes in the negotiation behavior, and 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 intrinsic measure of fitness of a child (or a case) is a weighted combination of (1) parental fitness, (2) status and profile of self, neighborhood, and world, (3) quality of the negotiation outcome, and (4) outcome of the negotiation. The vector of weights allows us to examine the health of the population and the casebase from different angles.

There is an important implication 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 casebase will undergo refinement and modification, which will affect the genetic hierarchy. An initial genetic case will eventually be either replaced (due to evolutionary incompatibility), or migrated (due to evolutionary adjustment), or supported with additional, more accurate cases (due to evolutionary refinement).

4.10.2 Environmental Fitness

In addition to the fitness of a case by itself, we also measure the environmental fitness of a case—that is the fitness of a case being a member in the casebase. Only intrinsically fit cases are considered for a promotion to the casebase. To be promoted, a case has to be unique, and uniqueness is part of the environmental fitness of the case. Similarly, a resident case in the casebase faces a possible demotion if its environmental fitness such as utility value falters. Changes in the environmental fitness affect the intrinsic fitness in the hierarchy. For example, a highly environmentally fit case will have a low intrinsic fitness, to protect the case's uniqueness in the casebase.

Our proposed fitness measure for a case being a member of a casebase includes the following:

First, a case is more fit in a casebase if it is more unique. The uniqueness can be measured by the multivariate distance of the case from its closest neighboring case. The further away a case is from its neighbors, the more unique it is. This also means that the case is representative of the domain it covers.

Second, a case is more fit in a casebase if it has been retrieved more frequently. If a case is frequently retrieved, then that means the case is very useful. Thus, the case's intrinsic fitness should be reduced in the genetic hierarchy to prevent further growth and to enhance the dominance of the case in the casebase. We propose a simple cycle-based frequency calculator. Every time the CBR module retrieves a case, it stamps the case with the current agent cycle index. Next time around, if the same is retrieved again, then the CBR module can use the interval between the old and new cycle stamps to compute the frequency; and then move the window for the next retrieval by replacing the old stamp with the new. This also implies that we favor recency in our fitness measure.

Third, we measure the proximity of a best case to the new case at every retrieval. Suppose we retrieve the best case *B* for a new case *A*. After adaptation, the new case *A* is used in a subsequent negotiation. Finally, the new case is completed with the negotiation outcome. Now, we review the fitness of the best case *B*. We compute the proximity of the new case with all existing cases in the casebase. If it is found that the new case is closer to another case (other than *B*), then we lower the fitness of *B*. If *B* is found to be the closest neighbor to *A*, then we increase the fitness of *B*.

Further, we collect the average proximity of new cases stemmed from each best case. If the average is large, then that means the best case is over-strained and is used to cover too large of a domain. If the best case remains as the centroid of the new cases, then population granularization and evolutionary refinement is needed. If the best case and the new cases form two poles (with only the best case in one pole), then population migration and evolutionary adjustment is performed. When one of the modifications occur, we lower the fitness of the best case and promote the fitness of the newly adjusted or newly evolved case.

Fifth, to keep the number of cases in the casebase small and asymptotic, we also incur a cost on promoting a new case to the casebase. The motivation for this is to make sure that the casebase is small such that the similarity evaluation during the retrieval and selection process and the diversity measurement during the storage and learning process can be conducted quickly. Suppose that we set the maximum size of the casebase to *K* and adding the new case to the casebase would exceed *K*. When this occurs, we locate the most similar case (situation, solution, and outcome spaces included) in the existing casebase for the new case. Then, we compute the diversity of the casebase as it is, and then we compute the diversity of the casebase with the most similar case replaced with the new case. If the latter yields a higher diversity, which is the average of the Euclidean distance measure between each pair of cases, then we eliminate the old case from the casebase and introduce the new case. Thus, the fitness of the old case is reduced since it is no longer in the casebase. The new case will inherit part of the environmental fitness from the old casebase that it replaces based on the similarity of the two.

Sixth, one may assign domain-specific utility to the impact that leads from each case on the environment. For example, suppose the case ultimately leads to a successful tracking of a target with the accuracy of 95%. This may be factored into the environmental fitness measure of the case. Note that between a successful negotiation and a successful tracking, there are many steps that must be right—timing, sensing accuracy, communication speed, noise, and others and therefore this sixth factor should only contribute minimally to the fitness measure.

Note that the environmental fitness strategically collaborates with the intrinsic fitness used in the genetic hierarchy to allow the various population maneuvers discussed in previous sections. It also allows us to modularize our domain modeling into two different stages.

5 DISCUSSIONS

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

During the creation of the casebase, the genetic hierarchy serves as a bootstrapping basis. It is bound to provide some seemingly fit cases to the casebase, and some cases are bound to be accepted into the casebase, 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 casebase, 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 may not have the luxury to perform the aforementioned steps to adopt the case into the genetic hierarchy since the computation is non-trivial. Thus, one may wish to look into separating the genetic learning from the case-based reasoning. First, all new cases will be added to the casebase. 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 casebase has seen significant changes, housekeeping is invoked. The housekeeping module will then inspect the casebase and time-sequentially update each case to the hierarchy.

Another issue is that, in our multi-agent environment, each agent maintains its own casebase and 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, one may expect to see interesting differences among the hierarchies at the end of the lifecycle of the agents.

6 CONCLUSIONS

We have proposed and described a conceptual framework for a genetic learning approach to create and maintain a casebase for a multi-agent, real-time, dynamic resource allocation domain in which agents negotiate to share resources and cooperate to perform tasks. This approach combines genetic algorithms and CBR to first build a genetic hierarchy and then a casebase to create and maintain useful cases. In the beginning, the genetic hierarchy will supply the main population of the casebase since there is a lack of cases. This initial casebase 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 and re-evaluation of the genetic hierarchy, thereby creating new possibilities that will re-stock the casebase. 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. We have also proposed two different types of fitness measure, intrinsic and environmental.

Acknowledgments

The work described in this paper is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0502. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

A. Aamodt, and E. Plaza (1994). Case-based reasoning: foundational issues, methodological variations and system approaches. *AI Communications* **7**(1):39-59.

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

J. H. Boose (1993). A survey of knowledge acquisition techniques and tools. In B. G. Buchanan and D. C. Wilkins (eds.), *Readings in Knowledge Acquisition and Learning*, 39-56. San Mateo, CA: Morgan Kaufmann.

S. Flinter 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.

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

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

R. Lopez de Mantaras and E. Plaza (1997). Case-based reasoning: an overview. *AI Communications* **10**:21-29.

M. L. Maher and A. Gomez de Silva Garza (1996) Design case adaptation using genetic algorithms. In J.Vanegas (ed.). *Computing in Civil Engineering*, ASCE.

H. S. Nwana, 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.

L. Purvis and S. Athalye (1997). Towards improving case adaptability with a genetic algorithm. *Proceedings of the 2nd Int. Conf. on Case-Based Reasoning (ICCBR'97).*

I. Watson and F. Marir (1994). Case-based reasoning; a review. *Knowledge Engineering Review* **9**(4):327-354.