

Soft Computing and Meta-heuristics: Using Knowledge and Reasoning to Control Search and Vice-versa*

Piero P. Bonissone

General Electric Global Research Center
One Research Circle
Niskayuna, New York 12309
Email: bonissone@research.ge.com

Abstract

Meta-heuristics are heuristic procedures used to tune, control, guide, allocate computational resources or reason about object-level problem solvers in order to improve their quality, performance, or efficiency. Offline meta-heuristics define the best structural and/or parametric configurations for the object-level model, while on-line heuristics generate run-time corrections for the behavior of the same object-level solvers. Soft Computing is a framework in which we encode domain knowledge to develop such meta-heuristics. We explore the use of meta-heuristics in three application areas: a) *control*; b) *optimization*; and c) *classification*. In the context of *control* problems, we describe the use of evolutionary algorithms to perform offline parametric tuning of fuzzy controllers, and the use of fuzzy supervisory controllers to perform on-line mode-selection and output interpolation. In the area of *optimization*, we illustrate the application of fuzzy controllers to manage the transition from exploration to exploitation of evolutionary algorithms that solve the optimization problem. In the context of discrete *classification* problems, we have leveraged evolutionary algorithms to tune knowledge-based classifiers and maximize their coverage and accuracy.

Keywords: Meta-reasoning, fuzzy controllers, evolutionary algorithms, control, optimization, classification.

Introduction

Origins and Definitions

Sometimes the term meta-heuristics is employed to describe the use of heuristic procedures, such as tabu search and scatter search, to extend search and optimization algorithms in order to avoid local optima (Glover, 1989; Glover and Laguna, 1995; Glover et al., 2000). However, in the context of our paper we will use the term meta-reasoning with the same semantics adopted by the AI community to indicate reasoning about an object-level problem-solver. Hence, meta-heuristics will describe the use of heuristic procedures at the meta-level to control, guide, tune, allocate computational resources for, or reason about object-level problem-solvers in order to improve their quality, performance, or efficiency. Meta-reasoning is a common approach in resource-bounded agent applications (Schut and Wooldridge, 2000), planning (Melis and Meier, 2000), machine learning and case-based reasoning (Cox, 1994; Fox, 1995), real-time fuzzy reasoning systems (Bonissone and Halverson, 1990), etc. Therefore, it is a natural step to extend it from its AI origins to the field of Soft Computing.

Offline Meta-heuristics

We can roughly divide meta-heuristics into *offline* and *on-line* architectures. This subdivision was suggested by Eiben et al. (1999) within the context of parameter setting for

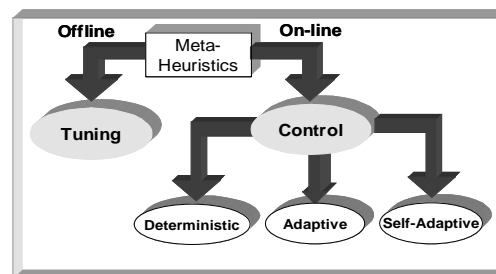


Figure 1. Offline and On-Line Meta-heuristics
(adapted from Eiben et al, 1999)

* An earlier version of this talk was given in Spanish as a keynote address at the *II Congreso español sobre Metaheurísticas, Algoritmos evolutivos y bioinspirados*, in Gijón, Spain on February 6, 2003. I want to thank the congress organizer, Prof. Luciano Sanchez Ramos, for giving me the opportunity to re-analyze some of my previous soft computing applications within the framework of meta-heuristics.

evolutionary algorithms, as illustrated in Figure 1. A similar partition can be easily generalized to meta-heuristics for other types of algorithms. We use *offline* meta-heuristics when we are not concerned with run-time modifications or control of the object-level problem-solver. In these cases, the goal is to define the best structural and/or parametric configuration for the model that is working on the object-level task. Most parameter tuning or optimization efforts follow this architecture. Figure 2 illustrates a typical schema for parametric tuning. A suite of representative problems from a problem class (or a sample of instances from the same problem) is used offline to test and optimize the solver. At run-time the problem solver is instantiated to perform its tasks with the resulting tuned parameters.

Among the many examples of such a tuning process, we cite the pioneering work of De Jong (1975), who popularized the use of static parameter sets for genetic algorithms (GA). Using five different landscapes as a test set, he explored a six dimensional space for GA parameters (composed of population size, probability of mutation, probability of crossover, percentage of replacement, scaling window, and selection strategy). From his analysis he suggested a set of parameter values, which have been generally accepted for several years by researchers and practitioners alike.

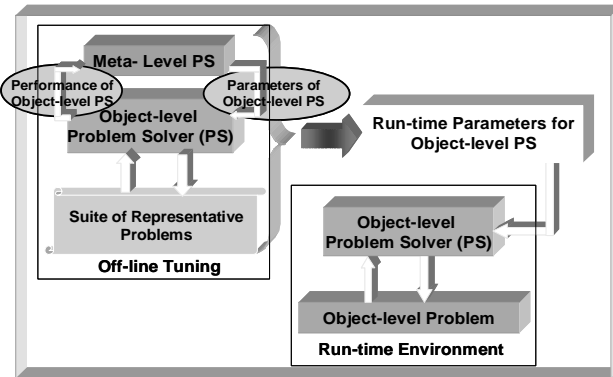


Figure 2. Schematic of Offline Meta-heuristics

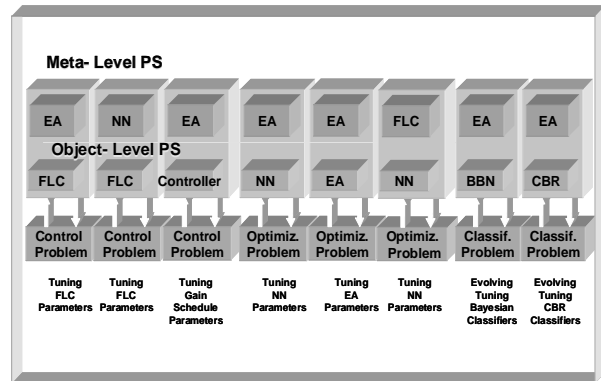


Figure 3. Example of Offline Meta-heuristics

Grefenstette (1986) proposed the use of a meta-level GA to search for the optimal parameter values for the same suite of five object-level landscapes used in De Jong’s early work. In this case, a meta-level GA replaced the manual analysis performed by De Jong. Each individual in the meta-GA represented an instance of six parameter values for the object-level GA. Each trial at the meta-level GA caused the object-level GA to run with the instantiated parameters for its maximum number of generations. The results of the runs, evaluated in terms of performance measures, provided the fitness value for the individual in the meta-level GA.

We have developed many instances of offline meta-heuristics applied to *control, optimization, and classification problems*, as shown in Figure 3. In these problems we considered multiple performance measures that needed to be satisfied or optimized. These measures were used to guide the offline parametric tuning process. In control problems, such as regulation tasks, we have used a combination of *tracking error* and *amount of energy* used by the actuators as the measures to guide EAs and gradient-based search methods in the parametric tuning of fuzzy logic controllers (scaling factors, term sets, and rule sets), and conventional PID controllers (gain vectors) - (Bonissone et al., 1996; Chen et al., 1999; 2000a; 2000b; 2000c). In optimization problems we have used the objective function (cost, time) and other characterizations of the solution quality, in conjunction with meta-EAs and FLCs to improve the optimization efficiency (Subbu et al., 1998b; Subbu and Bonissone, 2003). In classification problems we have relied on confusion matrices, and Type I and Type II measures to guide EAs in the tuning of configuration files of rule-based and case-based classifiers (Bonissone et al., 2002; Bonissone, 2003).

On-line Meta-heuristics

We use *on-line* heuristics when we want to generate run-time corrections for the behavior of the object-level problem solver. In these cases, we might want to steer the solver towards the most promising reasoning paths, identify opportunistic or critical situations, invoke the appropriate specialized routines or KBs, re-allocate computational resources to improve the solver performance, manage the transition between two different modalities, etc. As depicted in Figure 1, these corrections can be generated by an open loop, *deterministic* procedure (e.g., a scheduler),

by a closed-loop, *adaptive* system (e.g., a controller), or by a *self-adaptation* mechanism. We will focus on the implementation of on-line heuristics using a closed loop system, as shown in Figure 4. We will analyze examples of on-line meta-heuristics applied to *control* and *optimization* problems, as shown in Figure 5.

Control Problems. In control problems, we usually want to implement control laws that can efficiently handle scenarios in which the utility values of competing goals, such as response time, energy consumption, safety, and equipment life vary. We can think of these scenarios as different *modes* in which the system needs to operate. To achieve this versatility, we need to define a set of context-dependent tradeoff policies that will generate different solutions, corresponding to different compromises among the competing goals. These tradeoff policies are established at the meta-level (supervisory controller), while the objectives are tracked at the object-level (low-level

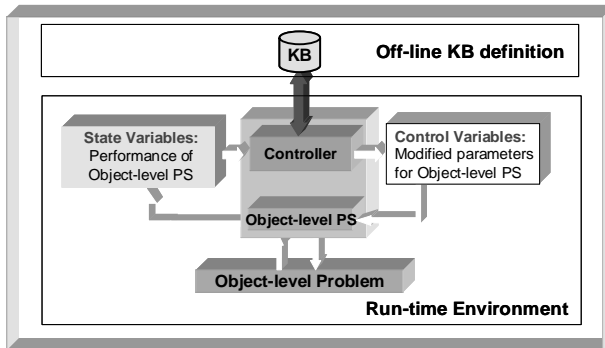


Figure 4. Schematic of On-line Meta-heuristics

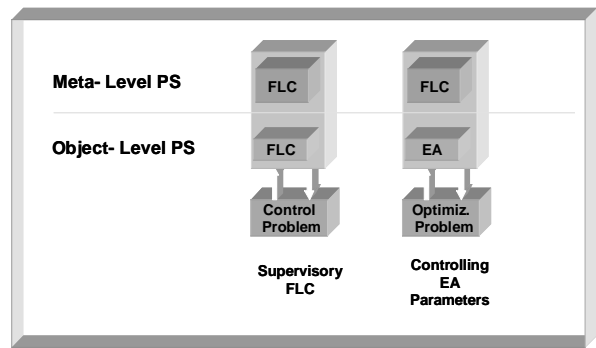


Figure 5. Example of On-line Meta-heuristics

controllers). A common approach to this problem is to build multiple controllers to deal specifically with the characteristics of each mode (normal operation, overheating, excessive acceleration, etc.) Traditionally, a *mode-selection supervisory* controller performs a gain-scheduling action by *switching* from one low-level controller to another that is better suited to handle the mode. Except for catastrophic modes, the system tends to have gradual mode transitions, and yet our control action changes are sudden. Therefore, these actions tend to cause *abrupt transitions (bumps)* during the mode-control switching, injecting high frequencies into the system, etc.

By using fuzzy supervisory controllers (FSC) to implement these policies, instead of performing a *mode switching*, we achieve a *mode melding* action that provides a *smooth transition* and tradeoff among competing goals (Bonissone and Chiang, 1995). The tradeoff policies are explicitly represented as rules in the knowledge base of a fuzzy supervisory controller. The FSC assigns control tasks to a subset of low-level controllers that are (partially) qualified to handle the situation. The partial matching process reflects the fact that during a mode transition, no single controller may be the only one qualified to handle the current state. Their outputs are weighted by the degree of applicability of the originating controllers and are recombined. This weighted recombination is a *soft mode switching* (or mode melding) and represents the result of a logical interpolation among applicable models. Another important feature of fuzzy supervisory control is its inherently simpler software maintenance. By explicitly representing the tradeoff policies as rules in the FSC knowledge base, we define our willingness to tradeoff performance under one criterion, such as speed, for better compliance under a different criterion, such as safety. These tradeoff policies are context-dependent, as they can only be triggered when the system state matches the left-hand side of the supervisory rules. Therefore, we can switch among different *supervisory modes*, such as *training* versus *combat* mode, by selecting different rule bases for the same supervisory controller (Bonissone and Chiang, 1995; Bonissone et al., 1995).

Optimization Problems. In optimization problems, we have used meta-heuristics to improve the efficiency and quality of the underlying optimization algorithm. Lee and Takagi (1993) proposed the use of a high-level genetic algorithm to identify and tune the rules of a fuzzy logic controller that is used to control the parameters of an object-level genetic algorithm. The object-level algorithm optimizes the five sample problems suggested by De Jong. They used a binary encoding for each three-parameter representation of a triangular membership function. They simultaneously identified the appropriate output rules and shapes of membership distributions. Their fuzzy logic controller was determined by three inputs (*Average Fitness/Best Fitness*, *Worst Fitness/Average Fitness*, and Δ *Best Fitness*), and three outputs (Δ *Population Size*, Δ *Crossover Rate*, and Δ *Mutation Rate*). Arnone et al. advocated a

similar form of fuzzy government of EAs in 1994. Subbu et al. (1998b) presented a comparison of the performance of a fuzzy logic controlled genetic algorithm (FLC-GA) and a parameter-tuned genetic algorithm (TGA) for an agile manufacturing application, which we will revisit in this paper. These strategies were benchmarked using a genetic algorithm (GA) that used a canonical static parameter set. In the FLC-GA, fuzzy logic controllers dynamically scheduled the population size, crossover rate, and mutation rate of the object-level GA using as inputs diversity (genotypic, and phenotypic) measures of the population. A fuzzy knowledge base was automatically identified using a meta-GA. In the TGA, a meta-GA was used to determine an optimal static parameter set for the object-level GA. The object-level GA supported a global evolutionary optimization of design, supplier, and manufacturing planning decisions for realizing printed circuit assemblies in an agile environment. The TGA performed superior searches, at the expense of larger search times. The FLC-GA performed faster searches than a TGA, and was slower than the GA that utilized a canonical static parameter set. However, search quality measured by the variance in search performance of the FLC-GA was comparable to that of the GA that utilized a canonical static parameter set. This latter negative result served as the key motivation for investigating the alternative, less complex approach discussed in (Subbu and Bonissone, 2003) and re-examined in this paper. Tettamanzi and Tommasini (2001, Chapter 7) discuss several combinations of fuzzy logic and evolutionary algorithms that include the development of fuzzy rule-based systems for adaptive control of an evolutionary algorithm.

Meta-Heuristics and The No Free Lunch Theorem

The use of heuristics to capture domain knowledge is compatible with results regarding the *no free lunch theorems (NFLT)*. These theorems state that for any optimization algorithm, an elevated performance over one class of problems is offset by degraded performance over another class (Wolpert and MacReady, 1995; 1997). In other words, the performance average of a given optimization algorithm over the entire class of potential problems is constant. If an algorithm performs better than random search for some problems, it will perform worse than random search for other problems, maintaining the performance average constant. These results suggest that any one set of potentially optimal algorithm parameters can be considered valid only for a limited subset of problems and should not be expected to result in consistently superior performance over the entire space of optimization problems, as was previously generally expected. The only way a strategy can outperform another is if “... *it is specialized to the structure of the specific problem under consideration*” (Ho and Pepyne, 2001). The NFLT results are important since virtually all decision and control problems can be cast as optimization problems. Therefore, it is essential to leverage specific problem domain knowledge and incorporate it into the algorithm.

Though this theoretical result could be used to discourage the search for a good general set of algorithm parameters, there is a wide body of experimental evidence that supports the online adaptation of parameters for improved search performance (Eiben et al, 1999). We can therefore conclude that: a) the use of meta-heuristics will improve the performance of optimization algorithms for a subset of problems; and b) the meta-heuristics will not be universal, but specific for a problem or a class of problems. Therefore, even the knowledge base (KB) used by the online adaptation scheme cannot be considered of general applicability. These conclusions have been validated by several experiments in the parametric control of EAs (Subbu and Bonissone 2003, Feng, 2003).

Hybrid Soft Computing: a Framework for Modeling and Developing Meta-Heuristics

Hybrid Soft Computing

The literature covering Soft Computing (also referred to as Computational Intelligence) is expanding at a rapid pace, as evidenced from the numerous congresses, books, and journals devoted to this issue. Its original definition provided by Zadeh (1994), denotes systems that “... *exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution cost, and better rapport with reality.*” As discussed in previous articles (Bonissone 1997; Bonissone et al, 1999), we view soft computing as the synergistic association of computing methodologies that includes as its principal members fuzzy logic, neuro-computing, evolutionary computing and probabilistic computing. We have also stressed the synergy derived from hybrid SC systems that are based on a loose or tight integration of their constituent technologies. This integration provides complementary reasoning and searching methods that allow us to combine domain knowledge and empirical data to develop flexible computing tools and solve complex problems. Thus, we consider soft computing as a *framework* in which we can derive models that capture heuristics for object-level tasks or meta-heuristics for meta-level tasks.

Using Hybrid SC to Develop Meta-Heuristics

Since leveraging domain knowledge and problem structure is so critical to the performance of the object-level problem solver, we will briefly review how such knowledge can be encoded in SC systems.

Design Tradeoffs. In the development of hybrid soft computing systems, we usually face one critical design tradeoff: *performance accuracy versus interpretability* (Guillaume 2001; Casillas et al., 2003). This tradeoff is usually derived from legal or compliance regulations and it constrains the underlying technologies used to implement the SC system. When we use SC techniques, the equation “*model = structure + parameters (& search)*”, takes a different connotation, as we have a much richer repertoire to represent the structure, to tune the parameters, and to iterate this process (Bonissone et al., 1999). This repertoire enables us to choose among different tradeoffs between the model’s interpretability and its accuracy. For instance, one approach aimed at maintaining the model’s transparency usually starts with *knowledge-derived linguistic models*, in which domain knowledge is translated into an initial structure and parameters. The model’s accuracy is further improved by using global or local *data-driven search methods* to tune the structure and/or parameters. An alternative approach, aimed at building more accurate models, might start with data-driven search methods. Then, we can embed domain knowledge into the search operators to control or limit the search space, or to maintain the model’s interpretability. Post-processing approaches can also be used to extract explicit structural information from the models. The commonalities among these models are the tight integration of knowledge and data, leveraged in their construction, and the loose integration of their outputs, exploited in their off-line use. For brevity we will focus on two SC components: fuzzy systems and evolutionary algorithms.

Fuzzy Logic Systems. Fuzzy systems provide the most explicit mechanism for representing heuristic knowledge directly into a knowledge base (**KB**). Domain knowledge is represented by a set of *if-then* rules that approximate a mapping from a state space \bar{X} to an output space \bar{Y} . In a Mamdani-type fuzzy system (Mamdani and Assilian, 1975) the KB is completely defined by a set of scaling factors (**SF**), determining the ranges of values for the state and output variables; a term set (**TS**), defining the membership function of the values taken by each state and output variable and a ruleset (**RS**), characterizing a syntactic mapping of symbols from \bar{X} to \bar{Y} . The *structure* of the underlying model is the ruleset, while the model *parameters* are the scaling factors and term sets. The inference obtained from such a system is the result of interpolating among the outputs of all relevant rules. The inference’s outcome is a membership function defined on the output space, which is then aggregated (defuzzified) to produce a crisp output. With this inference mechanism we can define a deterministic mapping between each point in the state space and its corresponding output. Therefore, we can now equate a fuzzy KB to a response surface in the cross product of state and output spaces, which approximates the original relationship. A Takagi-Sugeno-Kang (TSK) type of fuzzy system (Takagi and Sugeno, 1985) increases its representational power by allowing the use of a first-order polynomial, defined on the state space, to be the output of each rule in the ruleset. This enhanced representational power, at the expense of local legibility (Babuska et al., 1994) results in a model that is equivalent to radial basis functions (Bersini et al., 1995). The same model can be translated into a structured network, such as the adaptive neural fuzzy inference systems (ANFIS) (Jang, 1993). In ANFIS the ruleset determines the topology of the net (model structure), while dedicated nodes in the corresponding layers of the net (model parameters) define the term sets and the polynomial coefficients.

Evolutionary Algorithms. Problem domain knowledge can also be embedded in evolutionary algorithms. *Solution* and *constraint representation* are two critical design steps in which such knowledge can be leveraged. We want to use parsimonious representations to improve efficiency and avoid illegal solutions (e.g., use of Prim (1957) algorithm to represent trees instead of more general connectivity matrices). Similarly, we want to use proper encoding (e.g., Gray, integer, real, tree), appropriate data structures to capture static constraints (e.g., type, range, sets of possible values), penalty functions to assess the violation of dynamic constraints (e.g., generation-dependent penalties). *The initial population* is also a repository of best guesses, approximate results, and boundary conditions, all of which facilitate the search. Domain knowledge can also be engineered in the design of *customized variational operators* that preserve constraints and avoid generating illegal solutions (e.g., TSP mutation operators that will not introduce loops in the sequence of cities represented by a new individual). After all such structural design choices are made, the developer of an EA still has to define *parametric values* such as population size, mutation rate, etc. These parameter values can be tuned offline or controlled on-line (Eiben et al., 1999). In many situations we have noticed that the use of a systematically less aggressive mutation operation (as the number of generation increases) enables a smooth transition from an exploratory search to a more exploitative search. Similarly, when a population is

sufficiently diverse it is no longer necessary to maintain a large population. This is relevant especially for numerous real-world applications where fitness evaluations need to be conserved since they are time consuming, expensive, or both. The intelligent management of an evolutionary algorithm's mutation rate and population size seems to have a strong impact on the quality of the solution generated by the EA. We will further explore this topic in this paper. For a more detailed description, refer to (Subbu and Bonissone, 2003).

In the next three sections of this paper, we will briefly describe the role of soft computing in the implementation of these meta-heuristics for control, optimization, and classification problems.

Tuning and Controlling the Controller

The Problem: Regulation Control

We will analyze two different regulation control problems, for which a fuzzy logic controller was the object-level solution. In the first problem, we used evolutionary algorithms at the meta-level to tune the FLC parameters over a set of state trajectories. We then used the selected FLC parameters for its on-line implementation. In the second problem, we used a hierarchical fuzzy controller, whose KB encoded the heuristics for a fuzzy gain scheduling, to implement the meta-control. The first application was extensively described in Bonissone et al. (1996) while the second application was illustrated in Bonissone and Chiang (1995) and Bonissone et al. (1995).

First Regulation Control Problem: Automated Train Handler

The development of an automated train handler requires the control of a massive, distributed system with little sensor information. Freight trains consist of several hundreds heavy railcars connected by couplers. A typical freight train can be as long as two miles and requires up to four coupled locomotives (12,000 hp) to pull it. Each coupler between railcars may have a dead zone and a hydraulically damped spring. This implies that the railcars can move relative to each other while in motion, leading to a train that can change its length by 50 – 100 feet. The controlled forces on the train are due to a throttle and dynamic brake exerting a force on the locomotive, which is transmitted through the couplers, and a distributed air brake. Handling of these controls has a direct effect on the inter-car coupler dynamics and the forces and distances therein (called slack). The position of the cars and couplers cannot be electronically sensed. Couplers are subjected to high static forces and dynamic forces, which may lead to breakage of the coupler, the brake pipe, and the train. Violation of speed limits and excessive acceleration/breaking may lead to derailment and severe cargo damage. Smooth handling while following a speed target is therefore imperative. These boundary conditions make it hard to control the train so manual control depends heavily on the experience of the crew. Current locomotives are equipped with simplistic cruise control based on linear proportional integral (PI) controllers that can be used only below speeds of 10 mph. These conventional PI controllers are primarily used for uniform loading, yard movement, etc., and do not prescribe braking action. Furthermore, they do not consider slack or distributed dynamics, and are inappropriate for extended trains at cruising speeds over general terrain.

An automated system has to satisfy multiple goals: provide a certain degree of train handling uniformity across all crews, enforce railroad safety rules (such as posted speed limits), maintain the train schedule within small tolerances, operate the train in fuel-efficient regimes, and maintain a smooth ride by avoiding sudden accelerations or brake applications. This last constraint will minimize damage due to poor slack handling, bunching, and run-in. As described above, the handling of freight trains involves a *multi-body problem and proper slack management, without sensors for most of the state*. These constraints lead to a complex problem that cannot be solved by the simpler schemes used by cruise controllers for other vehicles, such as cars, trucks, boats, etc.

Object-Level Problem Solver: Fuzzy Logic Control (FLC)

Our solution used a fuzzy controller, composed of a *cruise planning module* and a *cruise control module* that can automate the controls of a freight train (Bonissone et al., 1996). The planning module created a velocity reference trajectory that the cruise control module had to track. In this description we will focus on the *cruise control module*, which was implemented as a fuzzy PI controller. The fuzzy PI used the tracking error (e) and its error change (Δe) to recommend a change in the control outputs throttle notch and brake settings (Δu). The computation of the error used by the PI incorporated a look-ahead to properly account for the train inertia. The algorithm predicted the future velocity of the train and incorporated not only the current error, but also the future predicted error, since the future reference velocity was known from the profile. The fuzzy PI was fully described by a knowledge base (**KB**)

composed of a rule set (**RS**), term sets of membership functions (**MF**), and scaling factors (**SF**). The rule set maps linguistic descriptions of state vectors $[e; \Delta e]$ into incremental control actions Δu . The term sets define the semantics of the linguistic values used in the rules; and the scaling factors S_e , S_d , and S_u determine the extremes of the numerical range of values for both the input and output variables. Ratios of the scaling factors are related to the integral and proportional gains in traditional PI controllers (Zheng 1992). The additional degrees of freedom provided by the MF and RS account for the FLC non-linearity.

Meta-Level Problem Solver: Evolutionary Tuning of FLC

We used a genetic algorithm to tune the fuzzy controller's performance by adjusting its parameters (the scaling factors and the membership functions) in a sequential order of significance. This approach resulted in a controller that was superior to the manually designed one. To test and tune the fuzzy controller, we used a simulator developed in-house, based on work done at GE and the Association of American Railroads. The overall scheme for the proposed train handling is shown in Figure 6. It consists of a fuzzy proportional integral controller (PI) closing the loop around the train simulation (TSIM), using only the current velocity as its state input. This velocity and look-ahead were compared with the desired profile to generate a predicted near-term error as input to the PI, which generated control actions back to the simulator. Finally, the PI was tuned off-line using an EA (implemented with GENESIS - GENetic Search Implementation System) that modify the controller's most sensitive parameters: *scaling factors* (SF) and *membership functions* (MF). The simulator TSIM was an in-house implementation, combining internal data with physical and empirical models. We then defined a *fitness function* that returns a corresponding value for any point in the search space.

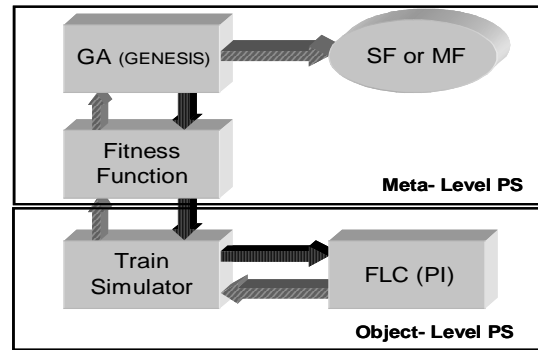


Figure 6. Example of EA Tuning of FLC Parameters

Fitness Functions. To study the effects of different objectives, we investigated two fitness functions:

$$f_1 = -\left(\sum_i |notch_i - notch_{i-1}| + |brake_i - brake_{i-1}|\right) \quad f_2 = -\left(w_1 \frac{\sum_i |notch_i - notch_{i-1}|}{K_1} + w_2 \frac{\sum_i |v_i - v_i^d|}{K_2}\right)$$

where v^d denotes the desired reference velocity and i is a distance or milepost index. The function f_1 captures jockeying of the throttle and brakes. The function f_2 combines a weighted sum of velocity profile tracking accuracy and throttle jockeying. The train simulator TSIM provided the environment in which the train's response to each controller could be tested, so that the values of the fitness function could be calculated over the entire journey.

Results

Tuning SF vs. MF with Fitness Function f_1 . We conducted four sets of tests to compare the significance of tuning scaling factors (SF) versus tuning membership functions (MF) with respect to fitness function f_1 . The results are shown in Table 1. The initial set of scaling factors was $[S_e S_d S_u] = [5.0, 0.2, 5000]$. After 4 generations of evolution, the set of GA-tuned SF was $[9.0; 0.1; 1000]$. As shown in Table 1, the fitness value was dramatically increased from -73.20 to -15.15 for the GA tuned SF with respect to f_1 , resulting in substantially smoother control. On the other hand, GA-tuned MF only reduced throttle jockeying slightly after 20 generations of evolution. This was reflected by a much small increase in fitness value from -73.20 to -70.93. From the above observations, we concluded that tuning scaling factors is more cost-effective, and has a larger impact than tuning membership functions.

Experiment Description	f_1	f_2
E1: Initial SF w/initial MF	- 73.20	- 1.00
E2: GA tuned SF w/initial MF	- 15.15	- 0.82
E3: Initial SF w/GA tuned MF	- 70.93	- 0.94
E4: GA tuned SF w/GA tuned MF	- 14.64	- 0.82

Table 1: Results of four sets of experiments for two types of fitness functions.

Then, we demonstrated that tuning membership functions only gave marginal improvements for a fuzzy PI controller with *tuned* scaling factors. We used GAs to optimize FPI's MFs with respect to f_i , while using the GA-tuned SF values of [9.0, 0.1, 1000]. After 10 generations of evolution, the fitness value was further increased from -15.15 to -14.64. The change in smoothness and the MF values was minimal. After observing the simulation results, we concluded that tuning membership functions alone provided only marginal improvements for a fuzzy PI controller with tuned scaling factors.

Tuning SF vs. MF with Fitness Function f_2 . We further verified our arguments by repeating the same four sets of tests using fitness function f_2 , which *balances both tracking error and control smoothness*. Recall that the initial set of scaling factors was [5.0, 0.2, 5000]. After four generations of evolution, the GA discovered a set of SF = [3.3, 0.9, 5571], with respect to function f_2 . As shown in Table 1, the normalized fitness value was increased from -1.00 to -0.82. On the other hand, GA tuned MF only increased the fitness value from -1.00 to -0.94 after 20 generations of evolution, confirming the claim that tuning SF is more cost-effective than tuning MF. We proceeded to experiment with MF tuning with tuned SF = [3.3, 0.9, 5571]. This time there were no significant improvements in fitness after 10 generations. Table 1 summarizes all eight test runs on a simulated 14-mile flat track. In addition, we performed the same experiments on a more complex 40-mile real track with the same train. The second set of experiments also showed substantial improvement in control accuracy and smoothness.

In (Bonissone et al., 1996) we developed an approach that used evolutionary algorithms to tune fuzzy systems for a complex control problem. We showed that all parameters do not need to be treated as equally important, and that sequential optimization can greatly reduce computational effort by first tuning scaling factors. Additional improvement was shown by the good performance of fairly coarse encoding. The scalability of the approach enables us to customize the controller differently for each track profile, though it does not need to be changed for different train configurations. We can thus efficiently produce a library of customized controller configurations.

Second Regulation Control Problem: Control of a Recuperative Turbo-Shaft Engine

Power plants for heavy vehicles must exhibit good acceleration and low fuel consumption. Gas turbines and diesel engines have comparable acceleration characteristics, but at idle conditions and under partial loads, gas turbines historically have had greater rates of fuel consumption. Present development in gas turbines is directed towards reducing these rates. Our efforts are focused on improving the performance and fuel consumption of a gas turbine power plant. The plant is a recuperative turbo-shaft, with a high-pressure spool supplying airflow to a free power turbine, in order to generate the high torque necessary for moving a heavy vehicle from rest. To address the fuel consumption problem, both a heat exchanger (recuperator) and a Variable Area Turbine Nozzle (VATN) are included to increase the efficiency of the thermodynamic cycle (Bonissone & Chiang, 1995).

Object-Level Problem Solver: Conventional and Fuzzy Logic Controllers (FLC)

In this plant, the engine was originally controlled by a set of *ten* low level controllers, designed to govern the engine when specific conditions, or modes, are sensed. At each time step, a *dominant controller* was *selected* from that group by a conventional mode selector and its output passed on to the actuators. Because only one controller is dominant at a given time, abrupt changes in control action may occur as regulation of the engine passes from one controller to another.

Meta-Level Problem Solver: Fuzzy Supervisory Control (FSC)

We first identified 24 individual operating modes and defined them as 24 fuzzy regions in the state space. We replaced the crisp mode-selector with a fuzzy supervisory controller (FSC) that compared the current state with each of the 24 modes, identified partial matches, fired the corresponding controllers, and interpolated among them by blending their outputs into a control action. The hybrid system control performed better than the baseline control, but still exhibited some limitations due to time lags that were intrinsically built in the low-level controllers. As a next step, we replaced the conventional low-level controllers with six customized fuzzy PI ones. Testing was performed on a component level simulation of the gas turbine and its associated transmission. Efficiency was improved, achieving nontrivial fuel savings.

One of the first uses of the FSC can be found in control of a small helicopter, proposed by Sugeno et al. (1991). In the helicopter control, three low level controllers were used for roll, pitch, and yaw stability, while a fuzzy supervisory control processed pilot commands and routed the inputs to the appropriate stability controllers. In the

gas turbine power plant control, six low level controllers are used to govern fuel flow and turbine nozzle area, while a fuzzy supervisory control processes driver commands and combines the outputs of the low level controllers.

Using fuzzy logic to implement a mode selector provided a number of advantages:

- a) Instead of having only a single controller active at a given time, modes can be defined with many controllers active at once. When switching is performed between two or more modes, the inference method of fuzzy logic *interpolates* between the control actions for those modes, resulting in smoother transitions.
- b) The actions of the fuzzy mode selector are *transparent* because the dominant modes can be found by examining an explicit rule set based on engine variables, whereas traditional mode selector are typically implemented by convoluted logical statements that are entangled with the code of the object-level controllers.
- c) Fuzzy logic mode selection allows us to transfer this knowledge to the mode selector, so that the low-level controllers handle the dynamics, while the higher level mode selector deals with quasi-steady state conditions, thus making the low-level controller design easier.

This application showed that fuzzy logic and fuzzy logic mode selection were experimentally tractable, providing performance comparable to that of the conventional control scheme. In the design of a conventional PI controller, there are a number of parameters available for adjustment, namely the integrator time constant, and the input and output gain vectors. The designer of a typical fuzzy logic PI controller has to determine, in addition to input and output scaling factors, the desired membership functions and rules. These additional degrees of freedom allow the controller to achieve better performance at the expense of more complex tuning procedures, whose automation is currently one of the most active research topics in FLC.

Results

Performance. The response of the plant improved with respect to the conventional control scheme. Shorter rise times (due to the non-linear gain of the controllers) were obtained in conjunction with reduced overshoots and faster settling times (due to highly tuned low-level PIs).

Fuel Consumption. Fuel consumption was lowered with respect to the conventional control scheme. The fuzzy logic mode selector was able to run the engine much closer to its optimal temperature, resulting in a more efficient thermodynamic cycle. Replacing the conventional controllers with fuzzy logic PIs and decoupling the actuator actions realized further fuel savings.

Component Life. The fuzzy control system provided excellent performance and fuel savings without sacrificing component life. All operational limits for gas and power turbines, transmission, and recuperator were maintained.

In summary, the reduced development cost, combined with the fuel savings, improved performance, and component life preservation, show the tremendous potential of this technology in addressing complex, hierarchical control problems.

Controlling the Optimizer

The Problem: Combinatorial Optimization

In flexible manufacturing problems, designers face increased complexity in coordinating suppliers and manufacturers of sub-assemblies and components to be competitive in cost, time, and quality. Design, supplier, and manufacturing decisions are made using the experience base in an organization and are often difficult to adapt to changing needs. Systems that support a coupling among design, supplier, and manufacturing decisions, simultaneously reduce supply and manufacturing costs and lead times, and better use available manufacturing facilities are critical to improved performance in these industries. A method and architecture for optimal design, supplier, and manufacturing planning for printed circuit assemblies is presented in (Subbu et al, 1998a). The problem formulation from this reference is used as a basis for generating object-level test problems. This formulation poses the optimal selection of designs that realize a given functional specification, the selection of parts to realize a design, the selection of suppliers to supply these parts, and the selection of a production facility to manufacture the chosen design, as a global optimization problem where each selection has the potential to affect other selections. The goal is to minimize an aggregate non-linear objective function of total cost and total time, $\min J_{ij} = \mathfrak{F}(Cost_{ij}, Time_{ij})$, of the i^{th} design assigned to the j^{th} manufacturing facility. The total cost and total time for realizing a printed circuit assembly are each coupled non-

linear functions dependent on characteristics of a chosen design, parts supply chain characteristics, and characteristics of a chosen manufacturing facility. This decision problem is in the class of *nonlinear discrete assignment problems*, and its characteristics do not support optimization using traditional techniques based on mathematical programming.

Object-Level Problem Solver: Evolutionary Algorithms (EA)

We used two EAs to test this problem. The first algorithm, *Standard Evolutionary Algorithm* (SEA), had a population size of 50, crossover rate of 0.6, mutation rate of 0.005, used proportional selection, applied uniform crossover to two selected parents to produce two offspring, and completely replaced with elitism the parent population with the offspring population. The second algorithm, *Steady State Evolutionary Algorithm* (SSEA), was identical to the SEA, except that only 25% of the population was replaced at each generation. The fuzzy controlled versions of these evolutionary algorithms were labeled *Fuzzy Standard Evolutionary Algorithm* (F-SEA), and *Fuzzy Steady State Evolutionary Algorithm* (F-SSEA), respectively.

The experimental setup was based on an underlying discrete decision problem space of the order of 6.4×10^7 feasible options. We used three different minimization problems by introducing various aggregation functions:

$$a) \min J_{ij} = Cost_{ij} \times Time_{ij} \quad b) \min J_{ij} = Cost_{ij} + Time_{ij}^2 \quad c) \min J_{ij} = Cost_{ij} \times \exp((Time_{ij} - 10)/3)$$

These objectives represented various heuristic tradeoffs between total cost and total time objectives in the design, supplier, and manufacturing planning. Experiments were conducted using 3000, 5000, 7000, 9000, and 11000 allowed fitness trials per evolutionary search, and for each experimental setup an algorithm's performance was observed over 20 repeat trials.

Meta-Level Problem Solver: EA and FLC to Control EA Parameters

The on-line meta-heuristics were implemented via a fuzzy logic controller. During the evolutionary search, the FLC observed the population genotypic diversity (GD) and percentage of completed trials (PCT), and using the embedded expert knowledge base (KB), specified changes to the population size (ΔPS) and mutation rate (ΔMR). For example, the DLC to control the change in population size used the following parameters: the *scaling factors* for the state variables and the output were [1 1 2]; the *membership functions* were five equally spaced triangular partitions, and the rule base is illustrated in Table 2.

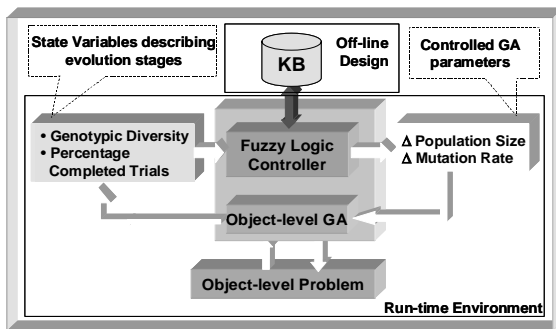


Figure 7. Schematic of FLC controlling EA's Parameters

GD\PCT	VL	L	M	H	VH
VL	PH	PH	PH	PM	NC
L	PH	PH	PM	NC	NM
M	PH	PM	NC	NM	NH
H	PM	NC	NM	NH	NH
VH	NC	NM	NH	NH	NH

Table 2: Rule base for Δ Population Size as a function of GD and PCT.

Results

We generated a total of 1200 experiments (20 trials for each configuration, for a total of 60 configurations: 3 fitness functions x 5 different trials x 4 types of EAs). We applied t-tests and F-tests to analyze the means and standard deviations of the experiments, and used a probability value of $p = 0.05$ to determine statistical significance. Overall, the μ performance of the F-SSEA was *statistically superior* to the μ performance of the SSEA 20% of the time, while the μ performance of the F-SSEA was *statistically inferior* to the μ performance of the SSEA 7% of the time, and the σ performance of the F-SSEA was *statistically superior* to the σ performance of the SSEA 47% of the time. The adaptive evolutionary algorithm generally resulted in a much tighter variance in search results, and is a significant improvement over evolutionary algorithms based on a static canonical parameter set. Moreover, this approach lead to a higher degree of search confidence as evidenced by a tighter spread in the search. What is also remarkable is that

except for one outlier in the space of experiments, the fuzzy controlled evolutionary algorithm approaches were inherently superior to their static parameter versions.

Tuning the Classifier

The Problem: Discrete Classification

We will focus on the process of underwriting insurance applications to illustrate a classification problem and its associated SC solutions. Insurance underwriting is a complex decision-making task that is traditionally performed by trained individuals. An underwriter must evaluate each insurance application in terms of its potential risk for generating a claim, such as mortality in the case of term life insurance. An application is compared against standards adopted by the insurance company, which are derived from actuarial principles related to mortality. Based on this comparison, the application is classified into one of the risk categories available for the type of insurance requested by the applicant. The *accept/reject* decision is also part of this risk classification, since risks above a certain tolerance level will typically be rejected. The estimated risk, in conjunction with other factors such as gender, age, and policy face value, will determine the appropriate price (premium) for the insurance policy. When all other factors are the same, higher risk entails higher premium.

Problem Abstraction. We represent an insurance application as an input vector \vec{X} that contains discrete, continuous, and attribute variables. These variables represent the applicant's medical and demographic information that has been identified by actuarial studies to be pertinent to the estimation of the applicant's claim risk. Similarly, we represent the output space \vec{Y} , e.g. the underwriting (UW) decision space, as an ordered list of rate classes. Due to the intrinsic difficulty of representing risk as a real number on a scale, e.g., 97% of nominal mortality, the output space \vec{Y} is subdivided into bins (rate classes) containing similar risks. For example 96%-104% nominal mortality could be labeled the *Standard* rate class. We consider the UW process as a discrete classifier mapping an input vector \vec{X} into a discrete decision space. This problem is not straightforward due to several requirements:

- 1) The UW mapping is *highly nonlinear*, since small changes in one of the inputs can cause large changes in the rate class output.
- 2) Most inputs require *interpretations*. The underwriter's subjective judgment will almost always play a role in this process. Variation in factors such as underwriter training and experience will cause different underwriters to issue different decisions for the same application.
- 3) These interpretations require an intrinsic amount of *flexibility* aimed at preserving a balance between risk tolerance, necessary to preserve price competitiveness, and risk-avoidance, necessary to prevent overexposure to risk.
- 4) Furthermore, legal and compliance regulations require that the models used to make the underwriting decisions be *transparent* and *interpretable*.

Object-Level Problem Solver: Knowledge-based Classifiers

For this application we built two classifiers. The first one, the *fuzzy rule-based classifier (FRB)*, was based on a set of constraints representing the underwriting knowledge currently used to guide human underwriters. The second one, the *fuzzy case-based classifier (FCB)*, was built on a set of episodic knowledge (cases) and retrieval guidelines that represent the result of applying such underwriting knowledge. We consider the first classifier to be a generative approach, since it generates a solution by applying the body of knowledge to input data, while the second classifier represents an analogical approach, as the input data is used as a probe to retrieve and rank the most similar cases. The commonality between these two approaches is that they both derived their structures and initial parameter sets from the domain knowledge. In both cases the parameters were tuned using offline meta-heuristics, based on evolutionary algorithms and a fitness function capable of capturing the tradeoff between price competitiveness and risk exposure.

Fuzzy Rule-Based Classifier (FRB). The fuzzy rule-based and case-based classifiers have been briefly described in two recent papers (Bonissone et al., 2002) and (Aggour et al., 2003). The fuzzy rule-based decision system uses rule sets to encode underwriting standards. Each rule set represents a set of fuzzy constraints defining the boundaries between rate classes. These constraints were elicited from knowledge engineering sessions with expert underwriters

to identify factors such as blood pressure levels and cholesterol levels, which are critical in defining the applicant's risk and corresponding premium. The goal of the classifier is to assign an applicant to the most competitive rate class, provided that the applicant's vital data meet all of the constraints of that particular rate class to a minimum degree of satisfaction. The constraints for each rate class r are represented by n fuzzy sets: $A_i^r(x_i)$ $i=1, \dots, n$. Each constraint $A_i^r(x_i)$ can be interpreted as the *degree of preference* induced by value x_i for satisfying constraint A_i^r . After evaluating all constraints, we compute two measures for each rate class r . The first one is the degree of intersection of all the constraints and measures the *weakest constraint satisfaction*¹, i.e.:

$$I(r) = \bigcap_{i=1}^n A_i^r(x_i) = \text{Min}_{i=1}^n A_i^r(x_i)$$

The second one is a cumulative measure of missing points (the complement of the average satisfaction of all constraints), and measures the *overall tolerance* allowed to each applicant i.e.:

$$MP(r) = \sum_{i=1}^n (1 - A_i^r(x_i)) = n \left(1 - \frac{1}{n} \sum_{i=1}^n A_i^r(x_i) \right) = n(1 - \bar{A}^r)$$

The final classification is obtained by comparing the two measures, $I(r)$ and $MP(r)$ against two lower bounds defined by thresholds τ_1 and τ_2 . The parametric definition of each fuzzy constraint $A_i^r(x_i)$ and the values of τ_1 and τ_2 are design parameters that were initialized with knowledge engineering sessions. Figure 8 illustrates an example of three constraints (trapezoidal membership functions) associated with rate class Z, the input data corresponding to an application, and the evaluation of the first measure, indicating the weakest degree of satisfaction of all constraints. For each constraint $A_i^r(x_i)$, let us label each pair of inflection points (indicated by circles in Figure 8) as (a_i^r, b_i^r) . Then, given the assumption of monotonically non-increasing preference, we can completely define a rule-based classifier \vec{C} as the vector:

$$\vec{C} = [\dots, (a_i^r, b_i^r), \dots, \tau_1, \tau_2] \text{ where } i=1, \dots, n \text{ and } r=1, \dots, (T-1).$$

Fuzzy Case-Based Classifier (FCB). Rule-based reasoning is a generative approach to the classification problem, since it generates the solution by applying rules to the input data. On the other hand, case-based reasoning (CBR) is an analogical approach to problem solving, since it relies upon finding previous solutions for *similar* problems and adapting them to the input data. Hence the definition of similarity plays a critical role in the performance of a CBR. Following this approach, the selection of the appropriate rate class (solution) for a new applicant (probe) is based on a set of previous problem / solution pairs (i.e. past cases) that are stored in a case base (CB), rather than the direct application of underwriting rules. The CBR process, which uses this CB, consists of the following steps (Aamodt & Plaza, 1994):

- Retrieval of similar cases from the CB,
- Reuse of the retrieved cases for the new application,
- Revision of the solution based on an underwriter evaluation, and
- Retention of the newly created application and rate class in the CB.

The retrieval step consists of finding all cases in the CB that are similar to the probe. The cases (and the probe) can be seen as points in an n -dimensional feature space. For each dimension we define a *truncated generalized bell function*, $GBF_i(x_i; a_i, b_i, c_i)$, centered at the value of the probe, that represents the degree of similarity along that dimension. Specifically:

$$\text{Truncated GBF}(x; a, b, c) = \begin{cases} \left[1 + \left| \frac{x-c}{a} \right|^{2b} \right]^{-1} & \text{if } \left[1 + \left| \frac{x-c}{a} \right|^{2b} \right]^{-1} > \varepsilon \\ 0 & \text{otherwise} \end{cases} \text{ where } \varepsilon \text{ is the truncation parameter, e.g. } \varepsilon = 10^{-5}$$

¹ This expression implies that each criterion has equal weight. If we want to attach a weight w_i to each criterion A_i we could use the weighted minimum operator. $I'(r) = \bigcap_{i=1}^n W_i A_i^r(x_i) = \text{Min}_{i=1}^n (\text{Max}((1-w_i), A_i^r(x_i)))$, where $w_i \in [0,1]$.

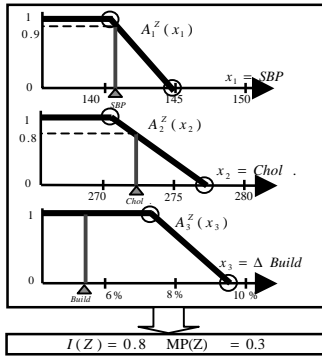


Figure 8. Example of Three Fuzzy Constraints for Rate Class Z

Since parameters c_i are instantiated by the values of the probe, each GBF_i has only two free parameters, a_i and b_i , controlling its spread and curvature. In a coarse retrieval step, we extract a case in the CB if all its features are within the *support* of the truncated GBF. The similarity measure is the intersection of all membership values (Bonissone & Cheetham, 1997), i.e.:

$$S = \bigcap_{i=1}^n GBF_i(x_i; a_i, b_i, c_i) = \text{Min}_{i=1}^n GBF_i(x_i; a_i, b_i, c_i)$$

The distribution of the retrieved cases is weighted by the similarity measure, reflecting the influence of the closest cases in selecting the decision. The mode of the distribution is used to identify the best rate class for the probe. The shape and cardinality of the distribution can be used to determine the confidence value in such output (Bonissone & Cheetham, 2001). We used knowledge engineering to determine the most efficient indices that represent the cases, to define the similarity metric that induces the best ranking among the retrieved cases, and to adapt the solution of the closest cases to the probe. The parameters used by the FCB to search for and rank similar cases (e.g., the free parameters, a_i and b_i) are also design choices that need to be determined and maintained over time for optimal performance. Details about the Java implementation of both decision engines can be found in (Aggour & Pavese, 2003).

Meta-Level Problem Solver: Tuning Classifiers' Parameters

Both FRB and FCB have design parameters whose values must be tuned to assure the classifier's optimal performance. To this end, we have chosen Evolutionary Algorithms (EAs - (Goldberg, 1989; Holland, 1994)). Our EA is composed of a population of individuals ("chromosomes"), each of which contains a vector of elements that represent distinct tunable parameters to configure the FRB or FCB classifiers. Examples of tunable parameters include the vector \vec{C} for the FRB, or the range of each parameter used to retrieve neighbor cases, e.g. a_i and b_i , for the FCB. A chromosome, the genotypic representation of an individual, defines a complete parametric configuration of the classifier. An instance of such classifier can be initialized for each chromosome, as shown in Figure 9.

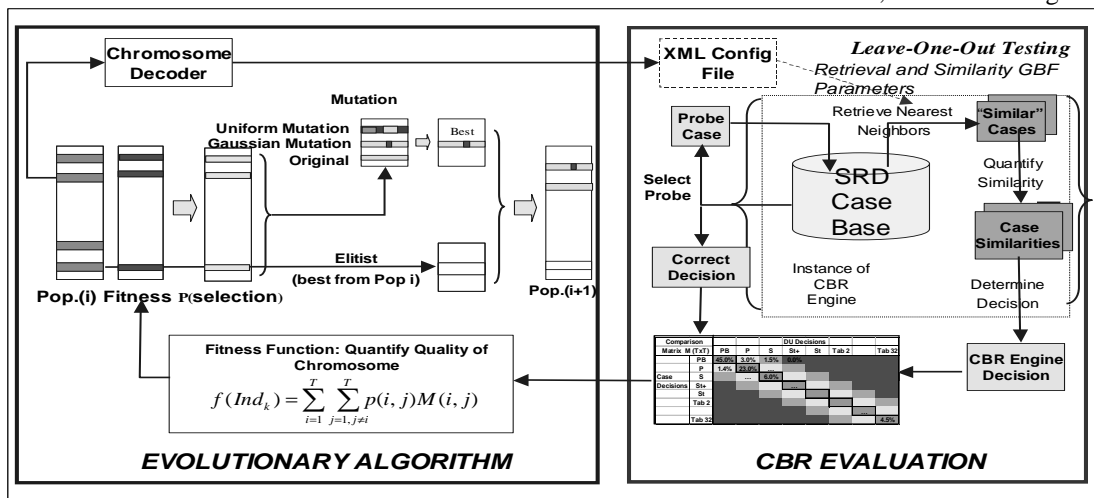


Figure 9. EA and FCB Interaction

Each chromosome c_i of the population $P(t)$ (left-hand side of Figure 9), goes through a decoding process to initialize the classifier on the right. Each classifier is then tested on all of the cases in the CB, assigning a rate class to each case. We can determine the quality of the configuration encoded by the chromosome (the "fitness" of the chromosome) by analyzing the results of the test. Our EA uses mutation (randomly permuting parameters of a single chromosome) to produce new individuals in the population. The more fit chromosomes in generation t will be more likely to be selected for this and pass their genetic material to the next generation $t+1$. Similarly, the less fit solutions will be culled from the population. At the conclusion of the EA's execution the *best* chromosome of the *last* generation determines the classifier's configuration.

Standard Reference Dataset (SRD). To test and tune the classifiers we generated a *Standard Reference* dataset (SRD) as our benchmark. The SRD contains approximately 3000 cases taken from a stratified random sample of the historical case population. Each of these cases received a rate class decision when it was originally underwritten.

However, to reduce variability in these decisions a team of experienced underwriters performed a *blind* review of selected cases to determine the *standard reference* decisions. These cases were then used to create and optimize both the FRB and FCB models.

Fitness Function. In discrete classification problems such as this one, we can use two matrices to construct the fitness function that we want to optimize. The first matrix is a $T \times T$ *confusion matrix* M that contains frequencies of correct and incorrect classifications for all possible combinations of the benchmark (SRD) and classifier decisions. The second matrix is a $T \times T$ *penalty matrix* P that contains the cost of misclassification. The fitness function f combines the values of M , resulting from a test run of the classifier configured with chromosome c_i , with the penalty matrix P to produce a single value:

$$f(c_i) = \sum_{j=1}^T \sum_{k=1}^T M(j,k) * P(j,k)$$

Function f represents the overall misclassification cost (Michie et al., 1994).

Results

Let us analyze the *confusion matrix* M , of dimension $T \times T$, which appears in the previous equation. The rows in M represent the standard reference decisions (rate classes), while its columns represent the classifier's decisions. Since rate classes are ordered by premium, without loss of generality we can consider that the leftmost column represents the most price-competitive rate class (i.e., the lowest risk associated to the lowest premium), and so on, with the most expensive rate class represented by column $(T-1)$. Column T represents the classifier's choice of not assigning any rate class, sending the case to a human underwriter. The same ordering is used to sort the rows. Then, the elements on the main diagonal represent the frequencies of correct classifications. Above the main diagonal we have frequencies of incorrect assignments to more expensive rate classes, a decision that might cause loss of potential business. Similarly, below the main diagonal we have the frequencies of incorrect assignments to less expensive rate classes, a decision that underestimates the risk of potential claims associated with the application. Using matrix M ,

$$\text{Coverage} = \sum_{i=1}^T \sum_{j=1}^{T-1} M(i, j) \quad \text{Relative Accuracy} = \sum_{i=1}^{T-1} M(i,i) / \sum_{i=1}^T \sum_{j=1}^{T-1} M(i, j) \quad \text{Global Accuracy} = \sum_{i=1}^T M(i,i) / \sum_{i=1}^T \sum_{j=1}^T M(i, j)$$

Table 3: Definitions of Performance Metrics for the Classifier

we defined three measures to evaluate the performance of the classifier, as shown in Table 3. With these measures we performed a test for all cases in the SRD. The results, partially reported in (Bonissone et al., 2002), are illustrated in tables 4 and 5 and show a remarkable improvement in all metrics.

METRIC	FRB SUB-OPTIMAL PARAMETERS	FRB OPTIMIZED PARAMETERS	FCB SUB-OPTIMAL PARAMETERS	FCB OPTIMIZED PARAMETERS
Coverage	90.38%	91.71%	47.97%	98.86%
Relative Accuracy	92.99%	95.52%	47.97%	90.80%
Global Accuracy	90.07%	95.52%	47.97%	89.77%

Table 4: Performance of the un-tuned and tuned rule-based (FRB) and case-based decision system (FCB)

After performing a five-fold cross-validation, we obtained stable parameters in design space and stable metrics in performance space. Specifically:

METRIC	FRB AVERAGE PERFORMANCE ON TUNING SETS	FRB AVERAGE PERFORMANCE ON TEST	FCB AVERAGE PERFORMANCE ON TUNING SETS	FCB AVERAGE PERFORMANCE ON TEST
Coverage	91.81%	91.80%	92.78%	92.15%
Relative Accuracy	94.52%	93.60%	93.18%	93.41%
Global Accuracy	92.74%	91.60%	92.45%	92.08%

Table 4: Average FRB and FCB performance over 5 tuning case sets compared to 5 disjoint test sets

Summary and Conclusions

We have addressed the importance of leveraging domain knowledge, embedding it in meta-heuristics to tune or control object-level problem solvers. “*Knowledge is power*” was a common AI slogan of the eighties. By paraphrasing it, we claim that “*meta-knowledge is power*” is still quite relevant nowadays, as we try to overcome some theoretical limitations derived from the NFLT. As we address control, optimization, and other decision-making problems, we need to exploit domain knowledge to properly initialize and modify the structures and parameters of the problem solvers. If “*meta-knowledge is power*”, then “*proper meta-knowledge representation is key*” to the success of this endeavor. Soft computing is a relatively new paradigm that provides a natural framework to represent such knowledge. With SC we can initialize models by using domain knowledge incorporated in their structures and initial parameter values. SC can also provide methods to incorporate domain knowledge in search operators and meta-controllers. With these goals in mind, we have explored the use of evolutionary algorithms and fuzzy systems to represent meta-heuristics for tuning and controlling a variety of object-level problem solvers. First, we illustrated two regulation control problems: the automation of a freight train handler and the control of a recuperative turbo-shaft engine. In the first application we used EAs to tune the FLC parameters, while in the second one we used an FSC to execute smooth mode transitions among low-level controllers. In both cases, the meta-heuristics improved the overall performance of the controllers. Then we addressed an optimization problem for a flexible manufacturing application. In this case we used an FLC at the meta-level to provide run-time parametric corrections for the EA that was performing the object-level optimization. Again, the quality of the solutions was improved by the use of the run-time meta-heuristics. Finally, we described a discrete classification problem: the underwriting of insurance applications. We addressed this problem by using fuzzy classifiers whose parameters were tuned by an EA at the meta-level. This offline tuning improved the classifiers performance, maximizing their accuracy while maintaining a high coverage of cases. These applications exemplify the increased efficiency achieved by object-level problem solvers when they are guided by offline or run-time meta-heuristics that leverage the relevant meta-knowledge.

References

- Aamodt A and Plaza E, (1994). “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches”, *AICOM*, Vol. 7, No. 1.
- Aggour K and Pavese M, (2003). “ROADS: A Reusable, Optimizable Architecture for Decision Systems”, *Proceedings of the Software Engineering and Knowledge Engineering Conference (SEKE '03)*, (to appear).
- Aggour K, Pavese M, Bonissone P and Cheetham, W. (2003). “SOFT-CBR: A Self-Optimizing Fuzzy Tool for Case-Based Reasoning”, *Proceedings of ICCBR 2003*, (to appear).
- Arnone S., Dell’Orto M., and Tettamanzi, A, (1994). “Toward a fuzzy government of genetic populations”, Proc. 6th IEEE Conference on Tools with Artificial Intelligence (TAI’94), pp. 585-591, IEEE Computer Society Press, Los Alamitos, CA.
- Babuska R, Jager, R, and Verbruggen HB, (1994). “Interpolation Issues in Sugeno-Takagi Reasoning,” in *Proc. Third IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE’94)*, pp. 859-863, Orlando, FL..
- Bersini H., Bontempi G, and Decaestecker C, (1995). “Comparing RBF and fuzzy inference systems on theoretical and practical basis,” in *Proc. of Int. Conf. on Artificial Neural Networks. ICANN '95, Paris, France*, vol.1, pp. 169-174.
- Bonissone P and Halverson P, (1990). “Time-constrained reasoning under uncertainty”, *The Journal of Real-Time Systems*, 2(1/2):25-45.
- Bonissone P and Chiang K, (1995). “Fuzzy Logic Hierarchical Controller for a Recuperative Turboshaft Engine: from Mode Selection to Mode Melding”, *Industrial Applications of Fuzzy Control and Intelligent Systems*, Yen, Langari, Zadeh (eds.), IEEE Press, pp. 131-156.
- Bonissone P, Badami V, Chiang K, Khedkar P, Marcelle K. and Schutten, M. (1995). Industrial Applications of Fuzzy Logic at General Electric, *Proceedings of the IEEE*, pp 450-465, vol. 83, no. 3, March 1995.
- Bonissone P, Khedkar P, and Chen Y, (1996). Genetic Algorithms for Automated Tuning of Fuzzy Controllers: A Transportation Application”, *Proceedings of the 1996 IEEE Conference on Fuzzy Systems (FUZZ-IEEE’96)*, pages 674-680, New Orleans, Louisiana.

- Bonissone P, (1997). "Soft Computing: the Convergence of Emerging Reasoning Technologies", *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 1, no. 1, pp 6-18.
- Bonissone P and Cheetham W, (1997). "Financial Applications of Fuzzy Case-Based Reasoning to Residential Property Valuation", *Proc 6th IEEE Conf. on Fuzzy Systems*, Barcelona, Spain.
- Bonissone P, Chen Y-T, Goebel K, and Khedkar K, (1999). "Hybrid Soft Computing Systems: Industrial and Commercial Applications", *Proceedings of the IEEE*, vol. 87, no. 9, pp 1641-1667.
- Bonissone P and Cheetham W, (2001). "Fuzzy Case-Based Reasoning for Decision Making", *Proceedings of the IEEE International Conference on Fuzzy Systems*, Melbourne, Australia.
- Bonissone P, Subbu R and Aggour K, (2002). "Evolutionary Optimization of Fuzzy Decision Systems for Automated Insurance Underwriting", *Proceedings of the IEEE International Conference on Fuzzy Systems*, Honolulu, Hawaii, USA.
- Bonissone P (2003). "The Life Cycle of a Fuzzy Knowledge-based Classifier", *Proceedings of the 2003 NAFIPS Conference*, Chicago, IL.
- Casillas J, Cordón O, Herrera F and Magdalena L. (Eds.), (2003). *Interpretability Issues in Fuzzy Modeling, and Accuracy Improvements in Linguistic Fuzzy Modeling*, Studies in Fuzziness and Soft Computing, Vol. 128-129, Springer-Verlag.
- Chen, Y-T, Bonissone, P, and Khedkar, P (1999). *System and Method for Tuning Look-ahead Error Measurements in a Rail-based Transportation Handling Controller*, US Patent No. 5,943,144
- Chen, Y-T, Bonissone, P, and Khedkar, P. (2000a). *System and Method for Providing Raw Mix Proportioning Control in a Cement Plant with a Fuzzy Logic Supervisory Control*, US Patent No. 6,113,256
- Chen, Y-T, Bonissone, P, and Khedkar, P. (2000b). *System and Method for Providing Raw Mix Proportioning Control in a Cement Plant*, US Patent No. 6,120,172
- Chen, Y-T, Bonissone, P, and Khedkar, P. (2000c). *System and Method for Providing Raw Mix Proportioning Control in a Cement Plant with Gradient-Based Predictive Controller in a Cement Plant*, US Patent No. 6,120,173
- Cox M. (1994). "Machines that forget: learning from retrieval failure of mis-indexed explanations". *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp.225-230. Lawrence Erlbaum Associates.
- De Jong K A, (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Thesis, University of Michigan, Ann Arbor, MI. Also available as Dissertation Abstract International, 36(10), 5140B, University Microfilms no. 76-9381.
- Eiben, A. E. Hinterding R, and Michalewicz, Z.(1999). "Parameter Control in Evolutionary Algorithms". *IEEE Transactions on Evolutionary Computation*, 3(2),
- Feng Xue, (2003). *Fuzzy Logic Controlled Multi-Objective Differential Evolution*, EAMRI Research Report #ER03-4, Rensselaer's Electronics Agile Manufacturing Research Institute, Rensselaer Polytechnic Institute, Troy, NY.
- Fox S. (1995). *Introspective Learning for Case-Based Reasoning*. Ph.D. Thesis, Indiana University, Department of Computer Science, Bloomington, IN, Tech. Rep. No. TR462.
- Goldberg, D E, 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, MA.
- Glover F, (1989). "Tabu search|part I", *ORSA Journal on Computing* , 1, 190-206
- Glover F and Laguna M, (1995). "Tabu search". In *Modern Heuristic Techniques for Combinatorial Optimization*, C. R. Reeves, Ed. McGrawHill, ch. 3, pp. 70-150.
- Glover F, Laguna M and Marti' R. (2000). "Fundamentals of Scatter Search and Path Relinking", *Control and Cybernetics*, (39)3:653-684.
- Grefenstette J J, (1986). "Optimization of control parameters for genetic algorithms", *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-16(1):122-128.
- Guillaume, S., (2001). "Designing fuzzy inference systems from data: An interpretability-oriented review," *IEEE Trans. on Fuzzy Systems*, vol. 9, no.3, pp. 426-443.
- Holland, J. H, 1994. *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, The MIT Press, Cambridge, Massachusetts, 3rd edition.

- Jang J.S.R. (1993). "ANFIS: Adaptive-network-based-fuzzy-inference-system," *IEEE Trans. on Systems, Man, and Cybernetics*, 233, pp. 665-685.
- Lee M and Takagi H, (1993). "Dynamic control of genetic algorithms using fuzzy techniques", *Proc. Of the 5th International Conference on Genetic Algorithms*, S. Forrest (ed.) , pp. 76-83, Morgan Kaufmann, San Matteo, CA.
- Mamdani E.H. and Assilian, S, (1975). "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. J. Man Machine Studies*, vol. 7, no. 1, pp. 1-13.
- Melis E and Meier A, (2000). "Proof Planning with Multiple Strategies", *Lecture Notes in Computer Science 1861*.
- Michie D, Spiegelhalter D, and Taylor C, (1994). *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, NY.
- Prim, R.C., (1957). "Shortest connection networks and some generalizations", *Bell Systems Technology Journal* 36,1389-1401.
- Schut M. and Wooldridge J (2000). *The control of reasoning in resource-bounded agents*", Knowledge Engineering Review.
- Subbu, R, Hocaoglu, C, and Sanderson, A. C.(1998a). "A Virtual Design Environment Using Evolutionary Agents", *Proceedings of the IEEE International Conference on Robotics and Automation*.
- Subbu, R, Sanderson, A. C., and Bonissone, P (1998b). Fuzzy Logic Controlled Genetic Algorithms versus Tuned Genetic Algorithms: An Agile Manufacturing Application, *IEEE International Symposium on Intelligent Control*, NIST, Gaithersburg, MD.
- Subbu R and Bonissone, P (2003). "A Retrospective View of Fuzzy Control of Evolutionary Algorithm Resources", *Proc 2003 IEEE International Conf. on Fuzzy Systems*, (FUZZ-IEEE'03), St Louis, Missouri.
- Sugeno M, Murofushi T, Nishino J, and Miwa, H (1991). "Helicopter Flight Control Based on Fuzzy Logic", *Proceedings of the International Fuzzy Engineering Symposium '91 (IFES'91)*, pp. 1120-1121", Yokohama, Japan.
- Takagi T. and Sugeno M (1985). "Fuzzy Identification of Systems and Its Applications to Modeling and Control," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 15, pp. 116-132.
- Tettamanzi A and Tommasini M., (2001). *Soft Computing – Integrating Evolutionary, Neural and Fuzzy Systems*, Springer-Verlag, Berlin, Germany.
- Wolpert DH and MacReady WG, (1995). *No Free Lunch Theorems for Search*. Tech. Rep. No. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM.
- Wolpert DH and MacReady WG, (1997). "No Free Lunch Theorems for Optimization", *IEEE Trans. on Evolutionary Computation*, 1(1).
- Zadeh L A, (1994). "Fuzzy Logic and Soft Computing: Issues, Contentions and Perspectives", *Proc. of IIZUKA'94: Third Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 1-2, Iizuka, Japan.
- Zheng, L (1992). "A Practical Guide to Tune Proportional and Integral (PI) Like Fuzzy Controllers," in *Proc First IEEE Int. Conf. on Fuzzy Systems*, (FUZZ-IEEE'92), pp. 633-640, S. Diego, CA.