

Evolution of Cooperative Behavior in Simulation Agents

Phillip D. Stroud (stroud@lanl.gov)

Los Alamos National Laboratory

LA-UR-98-73, SPIE proc. vol. 3390 (April 1998)

Jan 8, 1998

ABSTRACT

A simulated automobile factory paint shop is used as a testbed for exploring the emulation of human decision-making behavior. A discrete-events simulation of the paint shop as a collection of interacting Java actors is described. An evolutionary cognitive architecture is under development for building software actors to emulate humans in simulations of human-dominated complex systems. In this paper, the cognitive architecture is extended by implementing a persistent population of trial behaviors with an incremental fitness valuation update strategy, and by allowing a group of cognitive actors to share information. A proof-of-principle demonstration is presented.

1. INTRODUCTION

In an automobile assembly plant, car bodies with attached doors are painted in the factory's paint shop. Each car has a manifest which specifies which color, of those available for that model, it is to be painted. Once in the paint shop, a car enters an accumulator and waits for one of the several paint bays to receive it. When a paint bay finishes painting a car, the paint bay operator (human or machine) has to decide whether or not to take one of the cars in the accumulator, and if so, it must select which color.

If a paint bay switches to a different color, the paint spraying system must be flushed out with the new color of paint. Flushing takes time, wastes expensive paint, and generates expensive waste. One way to reduce flushing is by maintaining a large inventory of unpainted cars in the accumulator, but there is also a cost penalty for each hour a car has to wait in the accumulator. The performance of the paint shop depends on two factors: holding down the number of flushes per car, and holding down the number of cars waiting in the accumulator. These performance metrics, in turn, depend on the car-selection decision-making behavior of the bay operators.

Each paint bay has an operator that generates its car selection behavior through a cognitive decision-making process. This cognitive process can span the dimensions of intelligence: it can be a simple state-behavior mapping or use an elaborate internal representation, use little or much domain knowledge, be completely pre-programmed or adaptive over small or large domains of conditions, have or not have ability to learn from experience, etc. In addition, a group of operators can work and learn together via a social communicative process, or they can be limited to a less direct observing of each other's behavior. An example of a simple, pre-programmed, non-communicative behavior employing little domain knowledge is produced by all bays using the following rule set: upon finishing a car, A) if the accumulator has a car needing the currently loaded color, take it; B) else if there are any other cars in the accumulator, take the one that has been waiting the longest; C) if there are no cars in the accumulator,

then wait. At the opposite extreme, human operators with authority to exercise their own judgment exemplify an adaptive, social learning decision-making process, employing significant domain knowledge and an elaborate internal representation of the paint shop.

Even when the bay operators employ simple behaviors, the dynamics of the paint shop are complex, and the performance can not be assessed with simple analytical models. Some of the complexity derives from uncertainty: the paint shop doesn't know which color to paint a car until it arrives, although the distribution of colors is known. Evaluation of the number of flushes and the average accumulator inventory can only be accomplished by a modeling and simulation approach. Models of the various elements of the paint shop are developed, and then production runs are simulated, so that flushes can be counted and inventory monitored. A discrete event simulation of a paint shop has been constructed, where the events are arrivals of cars into the paint shop, and bays becoming empty after finishing painting a car. The simulation is implemented in the Java computer language, as a collection of interacting objects, and so can be run on many computers.

This paint shop simulation is part of an on-going research effort into emulation of human decision-making behavior, with the ultimate motivation of building simulations of complex, human-dominated systems. The current project builds on an evolutionary cognitive architecture, which has been successfully used to construct cognitive simulation agents with the ability to adapt their behavior over large domains of unanticipated conditions¹⁻⁴. Each cognitive actor generates a population of evaluated trial behaviors. Each of these trial behaviors completely specifies the decision-making process which in turn controls the car-selection behavior of the paint bay for any paint shop state. An evolutionary computing process is used by the cognitive actor to evolve the population of trial behaviors, using an internal representation of the paint shop to evaluate trial behaviors. This architecture enables cognitive actors to adapt their behavior to changing, unpredictable conditions.

This project explores two extensions to cognitive capability. First, some avenues for social interaction are built into the intelligent actors. In the society formed by the collection of paint bay operators, each bay can observe what the other bays are doing, and in addition, can communicate discoveries. The ability to communicate enough information about a behavior for another bay to imitate it has an impact on the emergence of cooperative group behavior.

The second area explored in this project is the temporal nature of the cognitive process, as related to learning by experience, and memory. Memory and learning are treated by making the population of evaluated trial behaviors persistent in time. Learning from experience is accomplished by updating the fitness valuation of the currently implemented behavior based on performance of the paint shop with that behavior in use. Memory results from keeping previously evaluated trial behaviors in the population, but updating their valuation based on the current environment. A simple version of a Kalman filter is used to adjust the estimated fitness of a trial behavior in response to both internal re-evaluation and external observation.

2. THE PAINT SHOP SIMULATION

2.1 Modeling the paint shop as Java objects.

The purpose of this paint shop modeling effort is to explore emulation of human decision-making behavior. The aspect of the system that must be captured in the simulation is the environment in which the paint bay operators make their decisions. The vast decision space that confronts real humans has

been distilled down to the essential behaviors of deciding what to do next after finishing a car painting job. The primary objects in the simulation turn out to be the bays, the accumulator, and an event manager at the paint shop level. In addition there is a set of objects which enable the simulation user to set up, run, and observe simulated operation.

In the following brief description of the software implementation of the simulation, classes will be denoted by names beginning with an upper-case letter; while object, method and member names begin in lower case. For descriptive purposes, a typical object of class `ClassName` will be denoted `className`. Methods and variables will be denoted with dot notation (e.g. `ClassName.methodName`, or `className.methodName`).

At the top level of the paint shop computer simulation is an object of the class `PaintShopDriver`. This object implements either a Java applet or application. It allows the simulation user to specify the scenario parameters (including the car input rate, the number of paint bays, the number of different paint colors, the fraction of cars requiring each color, the cost penalty of a flush, the cost per hour for waiting in the accumulator, the time needed to paint a car, the time needed to flush the sprayer, the initial reactive behavior of the bays, and some parameters describing the paint shop cognition). The `PaintShopDriver` class also contains methods for presenting the simulation results to the user. Once the user specifies the scenario, the `paintShopDriver` object instantiates and runs the paint shop simulation.

The paint shop simulation itself is implemented in the `PaintShop` class, which contains the objects that make up the paint shop (an array of bay objects of the `Bay` class, a car color sequence generator, an event queue, an accumulator) and methods to initialize and step through a simulated run of cars. The simulation runs by a discrete event handler in the `PaintShop.run()` method, which invokes an event-handling-method according to what the next event is, and then advances to the next event, and so on. The set of event types includes 1) a new car entering the paint shop, 2) a bay finishing painting a car, 3) a dump of recent paint shop performance, 4) a bay is scheduled to think, or 5) the end of the simulation. The `paintShop.eventManager` maintains a time sorted event queue. The queue contains a `bayFinishing` event for each currently active bay, associated with the expected time at which the bay will finish its current painting job. The queue also contains the next expected car arrival event, the next scheduled performance evaluation event, the next scheduled think event, and the end of simulation event. New events are added to the queue by the appropriate event handling methods. For example, when a new car arrival event is handled, the next car arrival event is added to the queue according to the expected inter-car arrival time. The `paintShop` holds an array of objects of the class `Bay`, one for each bay in the simulated paint shop. The `Bay` class has the method `Bay.takeCar`, which causes the bay to begin painting a car.

When the next event is a new car entering the paint shop, the `PaintShop.receiveNewCar()` method first pops the event from the event queue, adds the new car to the accumulator, and advances the `paintShop.clock`, a car counter, and some statistical measures such as the hours waiting in the accumulator. The accumulator keeps a list of the number of each color car currently waiting to be painted, which all bays can access. Each idle paint bay is queried (by calling its `Bay.submitBid()` method) to determine if it is willing to take any car in the accumulator. A simple bid system is used to ensure that a cars are sent to bays that don't have to flush their sprayers, if possible. If the accumulator holds a car of the color currently loaded in the idle bay's sprayer, that bay's `Bay.submitBid()` method will return a high bid for that color. If there are none of the currently loaded color in the accumulator, but the bay is willing to flush its sprayer and take a car of another color, the `bay.submitBid()` method will

return a low bid for the color it is willing to take. After soliciting bids from all idle bays, the `PaintShop.receiveNewCar()` method will send the requested car to a bay that submitted a high bid. If more than one bay submitted a high bid, one is selected at random. If no bay submits a high bid, but one or more submit a low bid, a car is sent to one of the low bidding bays at random. If no bay submits a high or low bid, the `PaintShop.receiveNewCar()` event-handler-method is finished, and the event manager goes on to the next event. The sequence of car colors entering the paint shop is produced by a roulette wheel sampling of a uniform random variate. The color of the next car is not known until it enters the paint shop. The baseline scenario has seven different colors, with 40% of the cars painted color 1 (white), 30% of the cars painted color 2 (red), and 6% painted each of 5 other colors.

When the next event is a bay finishing a car, the `paintShop.eventManager` invokes the `PaintShop.bayFinishes()` method, passing the identity of the event's newly idle bay as an argument. This method solicits a bid from that bay, and sends whatever color the bay asks for, if any. It also advances the clock and updates the event queue and the performance-tracking counters.

When the next event is an output to the user, a measure of performance is computed and output. The measure of performance is an incremental cost penalty, relative to a zero level corresponding to no flushes and no waiting in the accumulator. This cost penalty is accumulated over a user specified time interval (e.g. 1 week), by counting the number of flushes and the average accumulator inventory over that time interval, and dividing by the number of cars painted during the interval. After evaluating and outputting the performance, the counters are re-zeroed to begin another evaluation cycle, and the next evaluate-performance event is added to the event queue.

2.2 The cognitive task of selecting cars to paint.

The decision-making behavior of the bays is implemented in the `Bay.submitBid()` method. This car selection behavior is produced by three rules, which are evaluated in order. The first rule says that if there are any of the bay's currently loaded color in the accumulator, the bay will take a car of that color (it returns a high bid for that color). The criteria for invoking this rule is simply the existence in the accumulator of a car of the currently loaded color. The bay has a mechanism to sense whether such a car is available, and this sensory data is stored in a sensory buffer in the bay's memory.

If the first rule's conditions are not met, the second rule decides whether or not to take a car of a different color. Each bay has a palette, which specifies the sub-set of colors the bay will paint. The second rule counts the number of cars in the accumulator with colors in the palette. If this number is below a threshold, n_{Targ} , the bay will not take a car (it will submit a bid of *pass*). It will remain idle. The condition for this rule requires some manipulation of direct sensory input. The result of this manipulation is a bit in memory which indicates whether the number of in-palette cars in the accumulator exceeds the target inventory threshold. This data is perceptive or cognitive, since it represents internal or hidden knowledge. If there is a sufficient number of in-palette cars in the accumulator, the third rule will be invoked to decide which color to switch to.

This third rule forms a value for each color in the bay's palette, and selects the color with the highest value. The value of a color is given by $v_i = (n_i - b m_i)/p_i$. n_i is the number of cars in the accumulator requiring to be painted color i . m_i is the number of bays currently loaded with color i . The factor b sets the relative priority of two decision criteria: for low values of b , the value depends on the distribution of car colors in the accumulator, while a high value of b causes the valuation to be based on the number of bays currently loaded with the various colors.

2.3 Performance evaluation.

The performance of the paint shop can be quantified in terms of the number of flushes per car painted, and the average number of hours a car spends waiting in the accumulator. The performance can only be defined with an associated time averaging interval. The performance is defined in terms of the following observable parameters. Over a time interval beginning at paint shop clock time $t = t_0$ and having duration T hours, there are $N_P(t_0, T)$ cars painted and $N_F(t_0, T)$ flushes. The number of cars in the accumulator at time t is $N_A(t)$, so that during the interval (t_0, t_0+T) , the total number of car-hours spent waiting in the accumulator is $N_{WH}(t_0, T) = \int_{t_0}^{t_0+T} dt N_A(t)$. Since N_A can only change at events, this integral can be easily performed by incrementing a running total at those events, which advance the clock. The incremental cost per flush is $\$F$, and the incremental cost per hour spent waiting is $\$_{WH}$. The cost increment per car painted due to flushing and waiting, during the time interval (t_0, t_0+T) is given by $C = [\$F N_F + \$_{WH} (\tau_F N_F + N_{WH})] / N_P$. If the manufacturing process were scheduled so that whenever a car arrived at the paint shop, a paint bay loaded with the right color of paint finishes its previous job just in time, the incremental cost would be zero. Such scheduling is assumed to be impractical.

2.4 A Typical simulation run.

A baseline scenario was arbitrarily constructed with the following characteristics:

- 7.3 cars per hour into the paint shop, 15 paint bays
- 7 different colors, with likelihoods p_i in $\{0.4, 0.3, 0.06, 0.06, 0.06, 0.06, 0.06\}$
- $\$F = 200\$$ cost increment per flush,
- $\$_{WH} = 10\$$ per hour per car for waiting in the accumulator
- $t_p = 2.0$ hours to paint a car, with no flush,
- $t_f = 0.25$ additional hours to flush
- All bays have all 7 colors in their palette
- Bays will take another car immediately upon becoming idle, if possible ($n_{Targ} = 0$, for all bays)
- If a bay must switch colors, it selects the color with the greatest value of n_i/p_i . ($b=0$ for all bays)

The operation of the paint shop, with the baseline scenario, was simulated for a year of operation, with performance evaluated at one week intervals. The resulting weekly incremental cost per car due to flushing and waiting is shown in Fig. 1.

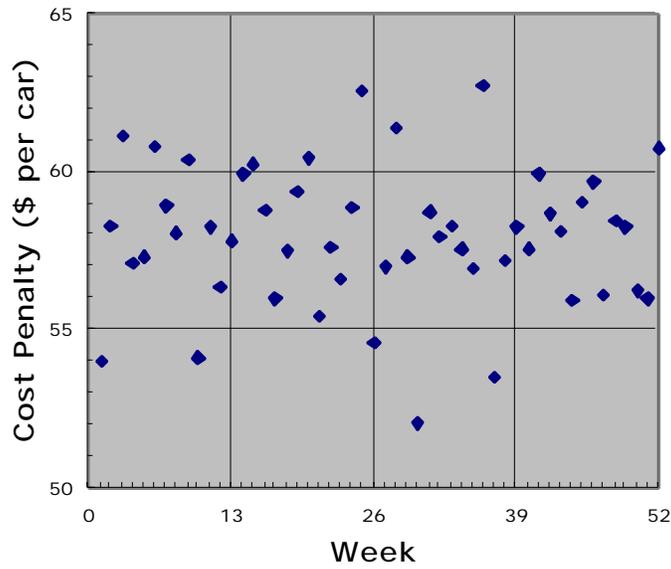


Figure 1. Weekly cost penalty from simulated paint shop with baseline scenario.

During this year of operation, the paint shop painted a total of 63772 cars. The weekly performance statistics are: the number of cars painted per week (1226.5), the average inventory of cars in the accumulator (10.0), and the average wait per car in the accumulator (1.37 hours), the fraction of paint jobs in which the paint sprayer is flushed (21.86%). The mean cost penalty over the year was \$57.97 per car. The standard deviation of the weekly performance values was $\pm \$2.20$. The error-in-the-mean, which characterizes the uncertainty in the mean value, is given by the standard deviation divided by the square root of the number of observations reduced by one: $\$2.20 / \sqrt{51} = \0.31 .

3. HIGHER COGNITION IN THE PAINT BAY

3.1 Elements that give higher cognitive abilities to the bays.

The reactive behavior described above enables a bay to make decisions based on the state of the accumulator, the status of the other bays, and the conditions under which the paint shop is operating. This decision-making process is static: it is constructed by the model builder or simulation user based on knowledge, and remains constant throughout a run of cars. This does not provide an accurate simulation of a paint shop, because the operators of the paint bays actually have the ability to adapt their behavior to changing conditions, in particular the conditions provided by the changing behavior patterns of the other bays. An architecture is under development whereby the bays can adapt their behavior as external conditions change, generate cooperative behavior, and automatically discover improved behavior patterns. The architecture uses an evolutionary apparatus to generate and evolve a population of trial behaviors, and an internal representation of the paint shop to evaluate the fitness of the trial behaviors. A fitness-based competition between trial behaviors provides the problem solving aspect of intelligence. Slower changing aspects of the behavior represent the adaptive nature of intelligence. Learning and memory are implemented by having a persistent population of trial behaviors that are stored between successive invocations of the Bay.think() method. A simple Kalman filter provides a mechanism to assess the impact that a behavior has on the achievement of the bay's goals, by prescribing how much to adjust the perceived fitness of the currently implemented behavior after observing the effects of its action.

By invoking the Bay.think() method in a bay object, the paint shop tells the bay to update its evaluation of its population of trial behaviors, and then adapt the population for the current environment. The user can control the frequency with which these calls are made, and the amount of computational resources the bays can expend. The Bay.think() calls are made in response to think events, which the event manager handles like any other event.

3.2 Representation of reactive behavior

The reactive behavior of a bay is produced by a set of rules, currently in force. There are three parameters which control the behavior produced by the rule set. The three adjustable parameters in this reactive decision-making behavior are 1) a palette of colors to which the bay restricts its action, 2) a target inventory number, and 3) the b factor that adjusts the influence of cars in the inventory relative to the influence of the status of the other bays. The palette specifies which of the alternative colors the bay is willing to paint. A bay's palette can be specified by a string of bits, with one bit for each of the possible colors. A value of one indicates the bay will paint that color; a value of 0 indicates a bay will not paint that color. For the baseline scenario with seven available colors, the palette is specified by seven bits (equivalent to a seven bit integer in [0...127]). The target inventory is also taken to be an integer in [0...127]. Finally, the b factor is allowed to take any of the 128 half integral values in [0, 1/2, 1, ..., 63 1/2]. Again, a seven bit integer can represent the 128 alternative values of b. The reactive behavior of a bay requires a total of 21 bits to specify. There are more than two million alternative behaviors that can be represented by these control parameters. If each bay can exhibit different behaviors, the domain of alternative behavior for the paint shop as a whole is enormous.

In order to evaluate the rule set, a bay must have access to some data about the current state of the paint shop. This data is the bay's sensory data, and includes n_i (the number of each color of car in the accumulator), and m_i (the number of bays currently loaded with each color). In addition, the bay has some scenario perception: it knows the car input rate, the cost factors, the time to paint and flush, and the expected distribution of colors. Like the paint shop state sensor data, the sensory channels for receiving, and the memory locations for holding the scenario perception data is built into the bay model. The control parameters, and memory locations which hold intermediate values for evaluating the rules (e.g. the v_i for each color in the palette) are cognitive data.

3.3 Representation of a population of alternative trial reactive behaviors.

The behavior of a bay is specified by three parameters, each having 128 alternative values. When a bay finishes a car and must decide what to do next, these three parameters control the decision-making process. These three parameters are stored in a particular memory location and are used by the behavioral rules when needed. In addition, a 21 bit string representation of these control parameters can be maintained elsewhere in the bay's memory. So can alternative strings representing alternative behaviors. In the evolutionary cognitive architecture, a population of alternative reactive control parameters is maintained within the bay's memory. This population is simply an array of chromosomes, where a chromosome is an array of three 7-bit integers. By using Java's bit manipulation operators, the 21 bits of these three integer genes can be manipulated just as readily as an array of 21 bits. The chromosome is expressed as three integers for purposes of concreteness of the gene, and for more efficient memory utilization. In one example described below, a larger chromosome is used to represent trial behaviors for the thinking bay and for other bays subordinate to it.

There is a fitness measure associated with each of the chromosomes, stored in an array of values. The array of fitness values and the array of chromosomes are always sorted, deleted from, and added to together, so that the correct fitness is always associated with each chromosome in the population. In addition, there is an uncertainty value associated with each chromosome, which characterizes the confidence in the mean value.

3.4. Evaluation of the fitness of trial behaviors.

There are two different mechanisms for evaluating the fitness of trial behaviors, one based on internal modeling within the bay, and the other by observation of the impact of implemented behaviors on the external system. When a new trial behavior is created by the evolutionary apparatus, it must be evaluated using an internal representation of the paint shop. This internal representation is a simulation within the bay, which contains models of the other bays, the accumulator, the job queue, and the event handler. The internal simulation can be set to match the current paint shop state, and then advanced through time on a what-if basis. The three parameters controlling the behavior of each of the 14 other bays are available to the bay, so that it can model their behavior. Since the bay doesn't know the future sequence of car color arrivals, it uses a Monte Carlo approach by drawing possible sequences with the proper color likelihoods. The bay runs its internal simulation through a specified number of imaginary cars, and evaluates the number of flushes per car and wait hours per car, which are then combined to give an estimated cost per car associated with the trial bay behavior. Then the simulation is rerun with another random sequence of cars, until a specified number of runs have been performed. An overall estimate of the trial behavior's fitness is obtained by averaging the results from each of these runs.

When a trial behavior has been evaluated previously, it will already have a fitness value associated with it. These valuations, however, would have been obtained in the past, and the current paint shop state and the behaviors of the other bays may have changed. Rather than using either the previous valuations or completely new ones, the previous and new valuations are combined. A Kalman formulation is used to assign relative importance to the old and new valuations, according to the confidence in their values. When a trial behavior is first evaluated, the result of the evaluation can be seen as a set of N time-averaged possible fitnesses. In the example of Section 5, for instance, the internal evaluation of a trial behavior produces a set of $N=60$ values $X=\{x_i\}$, each representing a one-week-averaged cost penalty that might be obtained with the prospective behavior. The initial fitness is taken as the mean of these 60 values, $f = \text{Mean}(X)$. The uncertainty in this mean, $P = \text{Variance}(X)/(N-1)$, has a direct interpretation as the element of a one-by-one Kalman covariance matrix. The greater the range of variation in the set of values, and the fewer the number of values, the less certainty can be ascribed to the estimated mean value. Now suppose that at a future time (15 weeks later in the Section 5 example), this trial behavior is to be evaluated again. Because there is no dynamic model for fitness, the most likely fitness of the trial behavior (i.e. the *prior* estimate of fitness) still will have the value f . The uncertainty that f represents the trial's expected fitness in the current situation (prior to the re-evaluation of the trial behavior), increases by an amount Q in the intervening time interval: ($P_{\text{prior,new}} = P_{\text{old}} + Q$). In the Kalman formulation, Q is a variance characterizing the distribution of process noise. The process noise has two components: variations due to changes in factory parameters (such as a change in the fraction of cars requiring each color, or car input rate), and variations due to changes in the behavior of the other bays. For the example given in Section 5, Q is taken to be 0.04 square dollars per square car for a 15 week time interval. This means that $\pm\$0.20$ fluctuation in the fitness value change over 15 weeks is attributed to

process phenomena. Now a new set of 60 one-week-averaged values is generated by the bay's internal model, this time using the new conditions and other bay's behaviors. The new set of values X_{new} has an associated measurement noise. The measurement uncertainty can be found from the set of N values, and is given by $R = \text{Variance}(X_{new}) / (N-1)$. This measurement noise is combined with the prior uncertainty to produce the Kalman gain factor: $K = 1 / (1 + R/P_{prior,new})$. To the prior fitness value is added an increment equal to the gain factor K times the difference between the new fitness and the prior fitness: $f_{new} = f + [\text{Mean}(X_{new}) - f]K$. Finally, the uncertainty is updated after the new evaluations, according to $P = P_{prior}(1 - K)$. This posterior uncertainty is then increased by the process noise as the whole process is repeated for the next re-evaluation.

A completely separate mechanism exists for evaluating a behavior: use it to control the reactive behavior and observe the impact of the behavior on the accomplishment of the goal. The currently implemented behavior always belongs to the population of trial behaviors. The recent occurrences in the paint shop, when the current behavior was in place, provides information on the valuation of the fitness of the behavior. The mechanics of the Kalman correction is identical to that described above, except a set of actual observations are used instead of a set of internally generated possible valuations. In the example of Section 5, the currently implemented behavior is observed over 15 one-week periods, and then is reevaluated. These 15 observations are used to adjust the valuation of the behavior. The measurement noise takes a denominator of 14 instead of 59.

3.5 Strategy adaptation by genetic algorithm.

An implementation of the basic genetic algorithm is used in the bay to maintain the population of evaluated trial behaviors. This population of trial behaviors evolves, and thus can adapt to changes in the bay's environment. First, the fitness value and uncertainty of the currently implemented behavior are updated according to the recent 15 weekly cost per car observations. Next, the behaviors currently implemented by all the other bays are added to the population if they are not already there. Duplicate chromosomes are not maintained, because unlike many applications of the genetic algorithm, in this case the fitness evaluations are much more expensive than the genetic manipulations. Any behaviors that are added replace the least fit trial behaviors in the population. If there is already an existing population, the trial behaviors are reevaluated using the internal simulation, starting with the current state of the paint shop. The fitness evaluation associated with the behavior from the previous think call is combined with the new evaluation, using the Kalman filter. After all trial behaviors in the population have been evaluated (or re-evaluated) and sorted, the evolutionary process begins. Two trial behaviors are selected from the population and used to generate a new trial behavior. The selection of the two parent behaviors is based on fitness. The likelihood that a behavior is selected is a linear function of rank in the population, with the best behavior being a factor of three more likely to be selected than the least fit behavior. The two parents are combined using two-point cross-over of their bit-string representation. The middle of one parent chromosome is sandwiched between the first and last part of the other chromosome. The location of the two cross-over points are uniform random. A mutation operator is then applied to the resulting child chromosome, where each bit has some probability of flipping. The population is checked prevent duplication. The fitness of the new trial behavior is evaluated via the internal simulation. If the fitness is better than the worst trial behavior in the population, the worst trial behavior is discarded and the new behavior is sorted into the population.

Several parameters control the operation of the evolutionary process. For the example in Section 5, the evolutionary apparatus maintains a population of 45 trial behaviors. Each call of the think() method causes 100 trial behaviors to be evaluated, including the 45 re-evaluations of existing behaviors. The mutation probability is initially set to 15%, and it decreases by a factor of 0.96 after every trial behavior evaluation. When the mutation probability drops so low that less than one bit is likely to flip, the mutation probability is increased so that there would be 6 bit flips expected. This punctuated equilibrium approach can both search local regions for optima and escape local optima. The parameter that determines how much computational resources will be spent by a bay's think() method is the number of simulated cars to imagine when it evaluates trial behaviors. If the internal evaluation of trial behaviors examines too few cars, the resulting fitness evaluation will be a poor characterization of the actual utility of the behavior, with a high measurement noise and corresponding small gain factor. For each of the 100 trial behavior evaluations per think event, the internal model is run forward 15 weeks from the current paint shop state four times, for a total of 60 weeks. Each think event causes the bay to model a total of nearly 8 million car arrivals., Each think event takes approximately 10 minutes on a 180 MHz 601 PowerPC based Macintosh, or about 30 seconds on a 250Mhz DEC Alpha workstation. A simulation which uses 60 think events thus takes 10 hours to run on the Macintosh, or a half hour on the faster workstation. To run a simulation without any think calls, however, requires only about five seconds to simulate a year of the paint shop operation (63772 cars) on the Macintosh.

4. CENTRALIZED COGNITION

In the first example of centralized cognition, one of the bays is designated as the thinker. The thinker follows the principle of doing that which would generate the greatest good if everybody else would do the same. The thinker uses its cognitive machinery to try to find improved, better adapted behaviors, represented by the three car color selection process control parameters. The remaining 14 bays simply imitate the thinker's behavior by setting their control parameters to match. Since all bays are implementing identical behaviors, it follows that all bays must paint all colors, else there would be colors that never get painted. The higher cognitive function of the thinking bay is then to increase expected performance by adjusting the two remaining parameters, nTarg and b. Although this search could be implemented efficiently as a gradient descent, the evolutionary cognitive apparatus was able to find a well adapted behavior for the given scenario after evaluation of 100 trial parameter sets (using about 10 minutes total on the Macintosh described above).

The thinker bay was sent a think command at the beginning of the second week of a simulated run. The chromosome had 21 bits, but the seven bits corresponding to the palette were always reset (repaired) to all 1's. The thinker bay constructed and maintained a population of 45 trial behaviors. The initial population contained the baseline behavior (palette=127=1111111 binary, nTarg=0, b=0) and 44 random chromosomes. Each of these trial behaviors was evaluated by 1) setting the behavior of all 15 internal bay models to the trial behavior, 2) initializing the internal model state to the current state of the paint shop and 3) averaging 4 independent internal simulation runs of 15 weeks of operation each. After evaluating 100 trial behaviors, the best behavior in the population was found to be (palette=127, nTarg=15, b=20). This behavior was then implemented as the reactive behavior for all 15 bays. The simulation of the paint shop then continued, with a performance over the subsequent year of \$35.78 cost penalty per car. With this behavior and the baseline scenario, the average inventory in the accumulator was 14.15 cars, and the ratio of flushes to cars painted was 8.07%.

A second example of centralized learning uses an expanded domain of representable behaviors for the paint shop. One of the bays is again designated as the thinker. Instead of looking for a single behavior for all bays to use, the thinker can try out four different behaviors, and assign these behaviors to the other bays. The trial behaviors are represented by a set of 16 integers, representing four castes of behavior. For each of the four castes, three integers, as before, represent the palette, the target inventory

and the weight factor b . The fourth caste parameter specifies the relative number of bays that will exhibit the caste behavior.

The thinker bay was sent a think command at the beginning of the second week of a simulated run. The chromosome had 112 bits, to represent the 16 adjustable parameters. The population had 45 trial behaviors. The initial population contained the baseline behavior (4 castes each with palette=127, nTarg=0, b=0, with 4, 4, 4, and 3 bays per caste, respectively) and 44 random chromosomes. After the same evolutionary process used for the 1-caste case described above, the best collection of bay behaviors was found to be

6 bays with palette=127(paints all colors), nTarg=13, b=16.5,

1 bay with palette=127, nTarg=4, b=18.5,

5 bays with palette=89(paints the first color and three of the five rare colors, nTarg=16, b=51,

3 bays with palette=2(paints the second color only), nTarg=0, b=51.

The reactive behavior of the other bays was set accordingly. The simulation of the paint shop then continued, with a performance over the subsequent 52 weeks of \$32.16 cost penalty per car. With this behavior and the baseline scenario, the average inventory in the accumulator was 14.28 cars, and the ratio of flushes to cars painted was 6.20%.

These new behaviors produce significant improvements in the paint shop performance (from \$57.97 per car with the baseline behavior down to \$35.78 and \$32.16 per car for the improved 1-caste and 4-caste behaviors). This approach can simulate a situation where a centralized agent (e.g. a supervisor or process designer) does all the thinking. It does not, however, simulate a group of interacting intelligent bay operators very well.

5. EACH BAY AS AN INDEPENDENT COGNITIVE ACTOR

With individual cognition, each bay continuously adapts its own behavior within the dynamic environment provided by the other bays and the paint shop. A bay's ability to perceive the currently implemented reactive behavior of the other bays is used to mimic an ability for the bays to communicate with each other. This is accomplished by giving each bay access to the three control parameters (its palette, nTarg, and b values) of the other bays. In particular, if a bay discovers a particularly effective behavior, it can communicate this behavior directly to the other bays, which may then imitate that behavior.

A continuous adaptive process is modeled by the paintShop sending each bay in turn a regular Bay.think() message. In the following example, one think message is sent every week, to successive bays, so that each bay gets to think and adjust its behavior once every 15 weeks. On each think command, the bay receiving the command generates and evaluates 100 trial behaviors for itself, under the assumption that the other bays behavior would continue as their currently implemented behavior. When beginning the think method, the bay first makes sure that all of the behaviors currently implemented in the other bays appear in the population of trial behaviors. On the first think command, the population is then filled with random chromosomes, which are then evaluated. The remaining allocation of evaluations is used to evaluate new trials generated by the evolutionary apparatus. On subsequent think commands, any currently implemented behaviors that aren't in the population are put in and evaluated, replacing the least fit chromosomes. The rest of the chromosomes that carried over from the previous think command are reevaluated using the currently perceived environment, and the new fitness valuation is combined with the old with the Kalman filter described in Section 3.4.

At the start of the simulation, all fifteen bays used the reactive behavior specified by (palette=127=all colors, nTarg=0, b=0), which gives the performance shown in Fig. 1. The simulation ran for 60 weeks, giving each bay four opportunities to think. The performance of the paint shop gradually improved as each bay in turn discovered better adapted behaviors for itself. The cost penalty is shown on a weekly basis in Fig.2. Improvement is rapid at first, with much of the gains occurring after each bay had only one opportunity to adjust its behavior. The collection of bay behaviors after the 60 weeks of adaptation are shown in Table 1. This adapted group of bays was then characterized by using it to control the bays for 52 weeks with no further adaptation: the resulting cost penalty was $\$34.10 \pm 0.27$ per car.

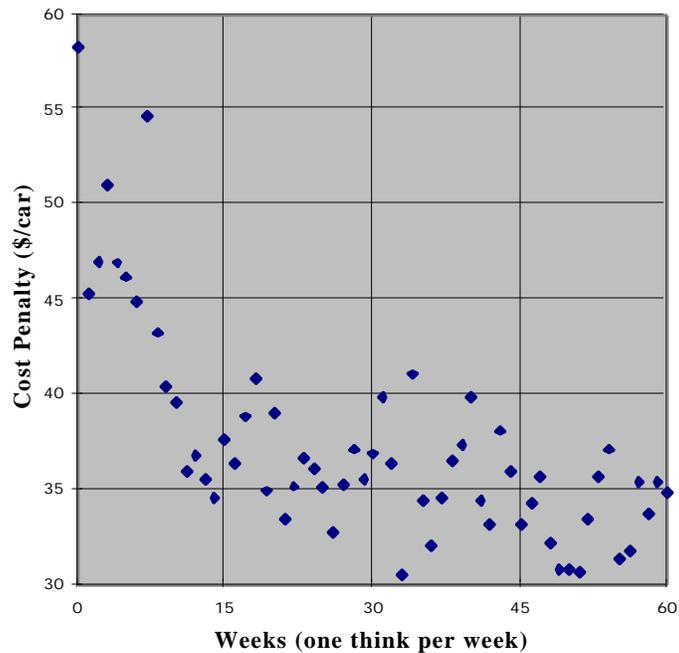


Figure 2. The week-averaged cost penalty due to flushing and waiting, when each bay adapts its own behavior in turn, at a rate of one bay adaptation per week, as observed in a simulation of the paint shop with the baseline scenario.

A second simulation with individual cognitive bay operators was run using a compressed learning period. During the second week of the run, each bay was given four opportunities to think (a think command was sent every 2.67 hours to the next bay in sequential order). The difference with the above is that only 20 cars enter the paint shop between successive think calls, whereas the weekly think calls of the preceding simulation were separated by the entry of 1226 cars. In this second simulation, the bays are adapting their behavior over a narrower range of conditions. In the 52 subsequent weeks, using the behaviors evolved during the second week, the paint shop operated at a cost penalty of $\$34.56$ per car. While these improvements are not as great as those obtained by a centralized cognitive agent, it is in the same neighborhood. More importantly, though, this approach simulates the paint shop's interacting intelligent bay operators with higher fidelity.

Bay ID	palette	Target Inventory	b-factor
0	85 -all but colors 2, 4, & 6	12	41
1	127 -all colors	12	8
2	127 -all colors	17	56
3	95 -all but color 6	8	56
4	87 -all but colors 4 & 6	12	76
5	103 -all but colors 4 & 5	9	57
6	111 -all but color 5	17	48
7	127 -all colors	17	56
8	101 -all but colors 2, 4, & 5	9	79
9	127 -all colors	12	50
10	127 -all colors	12	50
11	107 -all but colors 3 & 5	20	17
12	115 -all but colors 3 & 4	17	45
13	127 -all colors	12	8
14	63 -all but color 7	14	52

Table 1. The collection of bay behaviors after the 60 weeks of adaptation with one think event per week.

6. CONCLUSION

The purpose of this research is to explore some approaches to better emulation of human decision-making behavior. The starting point is an evolutionary cognitive architecture that uses the genetic algorithm to evolve a population of trial behaviors, which has the demonstrated capability to adapt behavior and solve problems in the face of unpredictable system dynamics. A promising extension of the evolutionary cognitive architecture is to maintain a persistent population of evaluated trial behaviors within the cognitive apparatus of the actor, and to use an incremental valuation adjustment strategy. This approach provides a natural way for cognitive actors to remember well-adapted alternatives to the current behavior, ready for implementation should the conditions warrant. It also provides a natural way to learn from experience. The incremental adjustment of valuation allows new information (from either observation or internal re-evaluation) to be combined with old in such a way that the actor can **adapt to change without overreacting to noise**. It also provides a means of using less reliable information: instead of having to evaluate the fitness of a trial behavior to a high confidence level, a quicker evaluation with worse confidence can be combined with the remembered fitness to get a high confidence updated fitness estimate. The common genetic algorithm is extended by 1) maintaining the population of chromosomes over time, and 2) maintaining an uncertainty value for each chromosome, along with the fitness value. This uncertainty and the fitness value are incremented after an observation or an internal re-evaluation using the Kalman filter algorithm, which implements a simple Bayesian predictor-corrector. This approach also enables the actor to perform its thinking in small chunks, so that continuous cognition by multiple actors can be emulated in simulation. An implementation of this approach is demonstrated in the simulation runs described in Section 5.

The second extension enables a group of cognitive agents to share the results of their on-going search for better-adapted behaviors. It is implemented by giving the bays access to information which, in the actual system, would be passed by social interaction or communication between the human operators. In the demonstration described in Section 5, three of the bays have found that their most effective behavior was to put into place behaviors discovered and implemented by other bays. In addition, some bays show behaviors that borrow genes from other bays. Clearly, this access to communal knowledge has a strong impact on the emergence of the behavior of a group of interacting cognitive actors.

There is a domain wherein a centralized cognitive agent (which dictates the behavior followed by the other actors) can find better adaptations than the individual agents were they to perform the cognition for themselves. This occurs when the central cognitive actor is able to generate and try out combinations of behaviors that are inaccessible to the individuals. If two bays can make simultaneous adjustments that improve performance, but neither bay could improve performance by adjusting its behavior alone, the centralized cognitive actor could find the adaptation, while the individuals could not. In simulating complex human-dominated systems, higher fidelity simulation based on better emulation of decision-making behavior may be of more interest than discovering better global solutions that the actual system could never reach.

7. REFERENCES

1. P. Stroud, "Simulation-based Learning in Knowledge-based Controllers," Proc. 1996 IEEE Int'l Symposium on Intelligent Control, pp. 168-174, (1996).
2. P. Stroud, "Learning and adaptation in an airborne laser fire controller," IEEE Transactions on Neural Networks, vol.8, issue 4, 1997.
3. P. Stroud, and R. Gordon, "Automated military unit identification in battlefield simulation," SPIE proc. volume 3069, April 1997
4. P. Stroud, "Adaptive Simulated Pilot," Journal of Guidance, Control, and Dynamics, Eng. notes, vol. 21, no. 2, pp352-354 (March-April 1998)