Abstract

Short Retraction Notice

The paper does not meet the standards of "Open Journal of Optimization".

This article has been retracted to straighten the academic record. In making this decision the Editorial Board follows COPE's Retraction Guidelines. The aim is to promote the circulation of scientific research by offering an ideal research publication platform with due consideration of internationally accepted standards on publication ethics. The Editorial Board would like to extend its sincere apologies for any inconvenience this retraction may have caused.

Editor guiding this retraction: Prof. Moran Wang (EiC of TEL)

The full retraction notice in PDF is preceding the original paper, which is marked "RETRACTED".

Scientific Research Publishing

Open Journal of Optimization, 2017, 6, 47-64
http://www.scirp.org/journal/ojop
ISSN Online: 2325-7091
ISSN Print: 2325-7105

A Rule Based Evolutionary Algorithm for Intelligent Decision Support

Wissam M. Alobaidi[1*], David J. Webb[2], Eric Sandgren[1*]

[1]Systems Engineering Department, Donaghey College of Engineering & Information Technology, University of Arkansas at Little Rock, Little Rock, AR, USA
[2]Axalta Coating Systems, Front Royal, VA, USA
Email: *wmalobaidi@ualr.edu, *exsandgren@ualr.edu

How to cite this paper: Alobaidi, W.M., Webb, D.J. and Sandgren, E. (2017) A Rule Based Evolutionary Algorithm for Intelligent Decision Support. *Open Journal of Optimization*, **6**, 47-64.
https://doi.org/10.4236/ojop.2017.62005

Received: April 11, 2017
Accepted: June 13, 2017
Published: June 16, 2017

Abstract

Genetic or evolutionary search algorithms seek and exploit the structure of a problem by operating on an encoding which represents the problem variables. The algorithms employed are generally designed to handle a wide variety of problems and while exploitation and progress can be rapid at the initial stages, the algorithms ultimate convergence rate can be slow. In order to speed up the solution process as well as produce a more refined solution, a rule based encoding is proposed. The rule based structure injects domain specific knowledge into the optimization process. This allows for an intuitive encoding and mimics the process utilized in many decision support applications such as scheduling. The encoding for the approach is often reduced in size as well. Specific rules for a particular problem class are coded into the formulation. This requires additional programming effort, but is valuable for specific applications which are repeated over time. During the solution process, a record is maintained concerning which rules or sequences of rules were successfully applied. This allows the rules to be continuously updated over time. An approach for implementing rules within the design encoding is demonstrated and several simple game problems are solved using this technique. The results are compared to a solution generated by a traditional encoding.

Keywords

Decision Support Systems, Planning, Scheduling, Learning

1. Introduction

One of the many practical applications of genetic or evolutionary algorithms is in the area of decision support. Generating a solution to this class of problems involves the ordered selection of a large number of individual decisions, each of which is influenced by every decision made up to that point. Common examples of decision support problems include routing [1], scheduling [2], bin packing [3] and game strategy [4]. Genetic algorithms are particularly well suited for solving

decision support problems as they seek a global, rather than a local optimum and can easily handle integer and discrete variables which are commonly required in the problem formulation. Significant hurdles are inherent, however, due to the presence of a large number of decision variables. This can result in slow final convergence and the potential of "noise" or unwanted characteristics appearing in the final solution. Unwanted characteristics could include two small batch runs of a part separated by a run of another part in a manufacturing schedule or a void in a packing problem that could easily have been filled. Evolutionary algorithms possess the trait of discovery [5], but the discovery process is heavily dependent upon randomness built into the search algorithm. For most decision support applications, specific domain knowledge is available which, if embodied in the problem encoding, could be used to greatly enhance both the convergence of the solution process and the quality of the final strategy.

In practice, most decision support activities are highly dependent upon human intervention. Some success in the application of evolutionary algorithms to various decision support application areas has been achieved by several investigators [5] [6], but the problem is far from being resolved. Factories are scheduled by production planners who have a wealth of experience and understanding of the intricacies of day to day operations. Truck scheduling and routing is directed by a dispatcher who has a knowledge base which is also difficult to emulate with a brute force optimization approach. On the other hand, this knowledge base is local in nature and often works against system level goals and objectives. Additionally, it is difficult to maintain a high level of human performance on a day in and day out basis [7]. For these reasons, it makes sense to add a level of computational support to aid the human in making key decisions. The idea of a self-evolving decision support system has been a goal for a considerable length of time [8]. The structure of the support, however, must be better aligned with the existing process which relies heavily on a knowledge based approach. This can be accomplished to some extent through the implementation of expert systems. On the other hand, a rule based genetic or evolutionary approach has the potential to deliver a knowledge driven process within the framework of a traditional optimization setting. A knowledge or rule based approach holds the promise of not only improving the effectiveness of the decisions being made, but also in capturing the knowledge to allow for the possibility of obtaining improved performance each day, every day.

The implementation of rules within the framework of an optimization algorithm is well established in the most general sense. One of the original computational approaches to solve the traveling salesman problem combined a simulated annealing algorithm, with a sequence of path modification rules [9]. The rules including moving groups of cities and reversing the order of a group of cities from the current visit sequence. Rule based approaches have been also applied to agent based systems involving communication networks [10]. In some cases, genetic algorithms were employed to optimize the rule selection. If the current strategy was viewed as the chromosome of a genetic algorithm, this approach could be considered a rule based evolutionary approach. Attempts in applying heuristic

knowledge within the broad framework of genetic algorithms has been attempted, but these efforts have generally been focused on modification of the genetic operators or in searching for rules that predict outcomes in data sets [11]. Rule mining for multi-objective association has been implemented in this area [12]. Chaotic sequences and cultural algorithms have also been employed to improve the performance of an evolutionary algorithm [13] [14] [15]. All of these approaches are designed to improve the performance of the evolutionary algorithm by guiding the evolution of the population based on the knowledge obtained during previous iterations.

The next logical step is to implement a genetic encoding which executes rules in the fundamental sense of being contained in the problem encoding itself. A rule based version of an evolutionary algorithm which implements this concept was constructed and applied to the area of structural design [16]. A trade-off exists between the amount of coding required by the user in order to solve a particular problem. If the problem is solved repeatedly over time, such as a factory schedule, the coding of specific rules can be justified. A generalization of this approach to the decision support problem is presented herein and several example problems are solved using this concept.

2. Genetic Formulation

A traditional nonlinear programming formulation employs a set of design variables which are modified through a search strategy. Success is measured by a combination of an objective function value(s) and the satisfaction of a set of constraints. This formulation, for a single objective search, is expressed mathematically as follows:

$$\text{Minimize } f(x); \text{ where } x = \left[x_1, x_2, \cdots, x_n\right]^{\mathrm{T}} \tag{1}$$

Subject to

$$g_j(x) \geq 0; j = 1, 2, 3, \cdots, J \tag{2}$$

$$h_k(x) = 0; k = 1, 2, \cdots, K \tag{3}$$

With

$$x_i^{\text{low}} < x_i < x_i^{\text{high}} \tag{4}$$

A design configuration or a decision strategy to a genetic algorithm is represented by an encoded string of information which is analogous to a chromosome in a living organism. Each position or gene in the string represents a specific design or decision characteristic which is directly linked to a specific application. The collection of all possible gene states represents the number of possible design or decision strings. If the encoding is thought of as a replacement for the vector of design variables in Equations (1)-(4), the relationship between a standard nonlinear programming and genetic optimization approach can be clearly seen. There are, however, several key distinctions which differentiate the two approaches. Among these distinctions are:

(i) The genetic algorithm operates by manipulation of the coding of the set of

gene values composing the chromosome.

(ii) A population of designs or decision strings is considered rather than a single design or decision point.

(iii) The transition from one set of designs or decisions to the next are probabilistic rather than deterministic.

(iv) The algorithm operates in a discrete rather than a continuous design or decision space.

These differences may not seem that dramatic, but they produce an algorithm which is more globally oriented in its search and is less likely to be trapped by local minima compared to other traditional approaches.

The overall suitability of a chromosome is termed its fitness. The property of fitness may be related to any function or functions of the design performance or the outcome of a decision strategy. This fitness is the characteristic which determines the probability of a particular chromosome to be selected. The selected chromosomes become parents for the next generation. The chromosomes possessing the greatest fitness have the highest probability of becoming parents. Even the least fit member, however, has a finite probability of being selected. The parent strings are combined using a variety of genetic operators in order to produce offspring. The process is repeated with the expectation that both the average fitness of the population as well as the best fitness contained in the population will improve.

In order to initiate the solution process, an initial population is generated randomly. The rules which govern how parents are selected and combined to produce offspring are also defined. Special operators such as mutation are included in order to guard against the loss of important design or decision information. This is necessary as the population represents only a small subset of the design or decision space. The search requires no gradient information and produces a number of design or decision alternatives. Alternatives can be useful when considering multiple design or decision criteria and accounting for unforeseen events. The process relies on the randomness present in natural selection, but exploits information gathered in order to produce a viable solution in a reasonable amount of time. The parallel nature of the solution process allows for the solution of problems requiring significant simulation times to access the outcome of a decision strategy. It also allows for solutions of problems that require large populations due to the size of the decision matrix. Additional detail concerning conventional genetic algorithms is given by Goldberg [17] and Davis [18].

3. A Rule Based Encoding

In a traditional genetic optimization, an encoding of the design is generated so that the structure of the design or decision support strategy may be directly manipulated by the genetic algorithm. This encoding structure is important, as it allows the genetic algorithm and associated genetic operators to modify the design or decision string. The genealogy of the design or decision string can be tracked from the initial population, through each subsequent generation, to the final outcome. Some

sense of discovery can be captured by observing what structural changes are made to the encoding during the process. There is no reason why this same approach cannot be extended to allow for design or decision rules which directly modify the genetic encoding. This process allows the genetic operators to manipulate the original problem encoding structure utilizing the rules, or to actually alter the rules themselves. This allows for a natural capture of knowledge or intelligence or even corporate memory concerning the problem solution as well as the computational path to the solution. The rules may be applied to a population of decision encodings which have been created randomly or through the application of a traditional genetic algorithm.

The implementation of a rule based encoding must be done carefully in order to provide as much flexibility as possible while avoiding the creation of a local search algorithm. Some randomness must be maintained in the execution of the rule base in order to preserve a global view. There must also be a mechanism for simultaneously firing multiple rules in order to transform a single decision string to a better state. Utilizing multiple rules simultaneously is essential, as in many cases, no one rule in itself is sufficient to improve the overall performance. Ideally, all of these features should be built into a standard genetic encoding structure. There are a number of encoding strategies that might be implemented and only one possibility is presented herein. The goal is not to generalize the concept, but to demonstrate the capability of such an approach.

At an elementary level, the process can be defined by a rule string as shown in **Figure 1**. Here, the first element represents the number of the rules defined by the string to actually apply. This allows for both a single rule or multiple rules to be executed on a given decision string in the current population. The second element defines which member of the decision strategy population to apply the rules to. If a single decision strategy is being considered for modification, this element may be eliminated. The subsequent groups of elements are used to define which specific rule or rules to apply with associated information blocks. These information blocks provide specific information as to how each rule is to be executed. The fact that there are only three information blocks within each

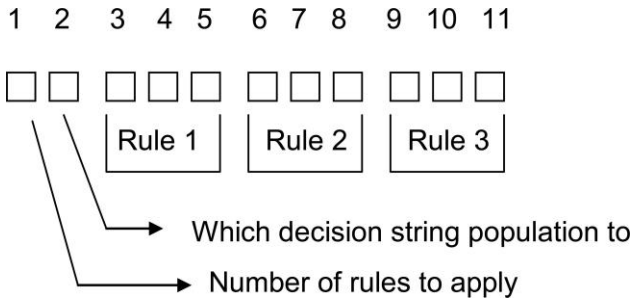


Figure 1. A Rule Based Encoding for a Genetic Algorithm.

rule definition group in **Figure 1** is of no consequence. The number can be arbitrarily expanded as needed. In order to ensure consistency in the crossover operation for generating offspring, it is convenient to keep the number of blocks in each rule execution group equal.

The rules, as structured, are applied to a conventional encoding string which represents a member of the design or decision support population. For example, in a manufacturing scheduling example, the problem encoding being operated upon could represent the individual job priorities. The priorities could be fed into a simulation, would provide the performance metrics with which to evaluate the effectiveness of the decision string. The rules would then operate on the job priority encoding and could include concepts such as:

- Increase job priority on a job which is delivered late.
- Decrease job priority on a job which is ready early.
- Alter a priority of a selected job (as specified in the rule encoding).
- Switch priorities between two jobs (each specified in the rule encoding).
- Divide a product batch order into two separate orders or jobs for separate processing.
- Combine two product batch orders so they are processed together.

A number of other potentially effective rules could be implemented as well. The rule based encoding operates on the decision encoding population to modify the overall schedule. Good or bad rules will be identified by the process. Good rules will be exploited while bad rules will eventually evolve out of the rule population. This allows for a straightforward capture of knowledge.

The process may now be thought of as a two phase process. The first phase involves the generation of a population of decision strategies. A traditional genetic optimization can be executed to form this population. Alternatively, it could be formed through a random selection, much as the population for the first generation in a conventional genetic solution process. A subset (one or more) of this population could then be subsequently processed by the rule based encoding. There are many possibilities to move between the phase one and phase two processes. Care must be taken not to have such specific rules as to eliminate the global nature of the search. This is why generic rules such as the one to alter a priority of a selected job or switch the priorities of two jobs are included. This allows the algorithm to select a schedule as part of the genetic selection process which is then subsequently influenced by the application of rules. One final point of note is the relative size of the design or decision encoding. The original problem encoding may involve hundreds or thousands of elements while the rule based encoding will typically involve tens of elements. The reduction in overall solution time relies on the fact that solving a set of smaller problems sequentially is far more efficient than solving a very large problem. This has a significant impact on convergence and the computational time required.

4. A Simple Example—The Summation Game

In order to demonstrate how a decision support problem is formulated for solution via a rule based genetic approach; consider the summation game shown in **Figure 2**. The goal of the game is to place the consecutive integers, one through nine, so that the sum of each row, column and diagonal is identical. For the example placement of the integers in **Figure 2**, the sums are shown for each row, column

and diagonal and are seen to be far from equal or optimal. The objective function utilized in this example will be the difference between the highest and lowest sum of any row, column or diagonal. The row, column and diagonal sums are shown for the initial placement of integers in **Figure 2** by the small numbers following the red arrows. For this initial placement, the objective function is eighteen (difference of lower row sum of twenty four and upper row sum of six). In this instance, the integers are simply placed in order by rows. Certainly, this placement does not represent a solution to the game.

Using a conventional genetic optimization algorithm, there are many ways to formulate this problem for solution. Remarkably, even a simple problem like this is not easily solved. The approach considered here is to compare a traditional, brute force approach, with a rule based decision support approach. A conventional approach would consist of implementing a nine element encoding where each element in the encoding is allowed to take on an integer value from one to nine. Each position of the encoding string translates directly to a position in the sum game, starting at the top left corner and proceeding across and down. The sums of the rows, columns and diagonals are easily calculated and provide a means of evaluating an appropriate objective function. In this example, the objective function is taken to be the difference between the largest and smallest of the sums. If this difference is zero, then all of the sums are equal and a potential solution has been located. The term potential is used as there is no guarantee in this formulation that each integer value is utilized once and only once. For example, if any integer value were repeated nine times, all sums would be zero, but it would not represent a solution to this particular game.

In order to guarantee that each encoding position represents a unique integer at the end of the search process, a constraint is added to the formulation which successively penalizes the multiple use of an integer in the encoding string. This is not the only way to avoid the multiple value issue, but it represents an easily

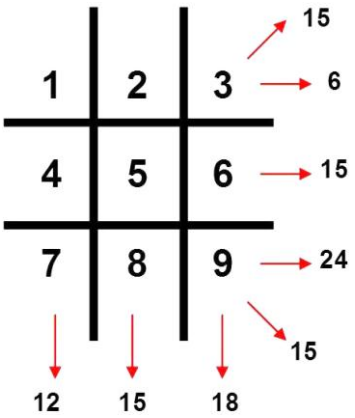


Figure 2. Simple Sum Game at Initial Setting.

implemented approach. The first formulation of the problem may now be expressed in mathematical form as follows:

$$\begin{aligned}\text{Minimize } f(x) = &\text{Maximum}\left[Sum\{row_i\}, Sum\{column_j\}, Sum\{diagonal_k\}\right] \\ &- \text{Minimum}\left[Sum\{row_i\}, Sum\{column_j\}, Sum\{diagonal_k\}\right]\end{aligned} \tag{5}$$

where

$$x = \{x_1, x_2, x_3, \cdots, x_9\}; 1 < x_i < 9 \text{ and } x_i = \{\text{integer}\} \tag{6}$$

$i = 1, 3$

$j = 1, 3$

$k = 1, 2$

and

$$g_1(x) = N_{dup} \tag{7}$$

where

N_{dup} = total number of repetitions of integers in x

This formulation was executed using a conventional genetic algorithm and the objective function solution history is documented in **Figure 3**. This solution represents a particular run. Additional runs were made where the population size, random seed and number of generations required were investigated. The solution history for this solution for the constraint value is plotted in **Figure 4**. An exterior penalty function was utilized for this solution. A population size of 2000 was selected with the best five design encodings being carried over from generation to generation. A valid solution to the game was generated on the forty fifth generation which is documented in **Figure 5**. Note that the sum of each row, column and diagonal is equal at a value of 15. From the constraint history in **Figure 4**, initial generations utilized multiply repeated integers in the decision

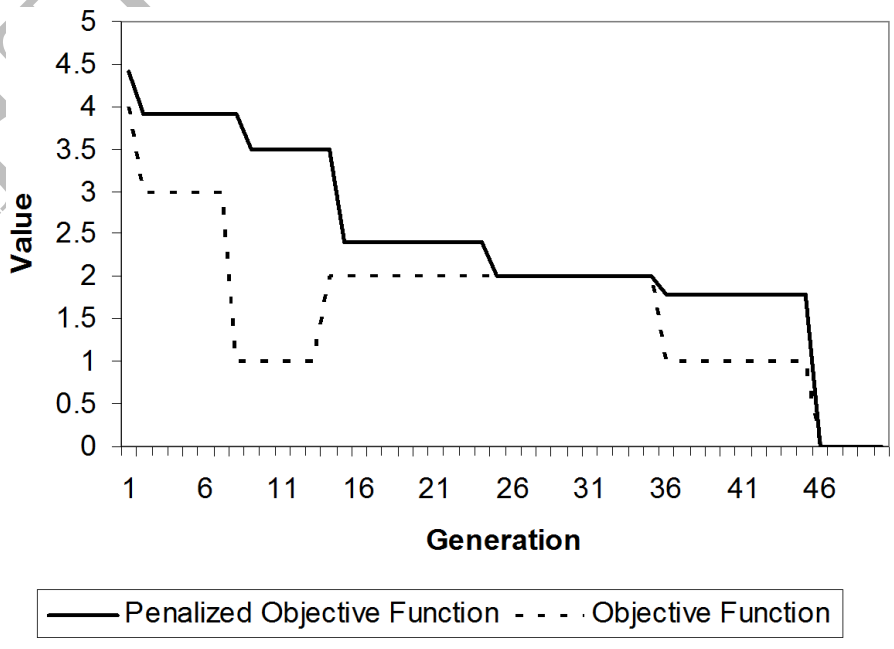


Figure 3. Objective Function Solution History for Sum Game Problem.

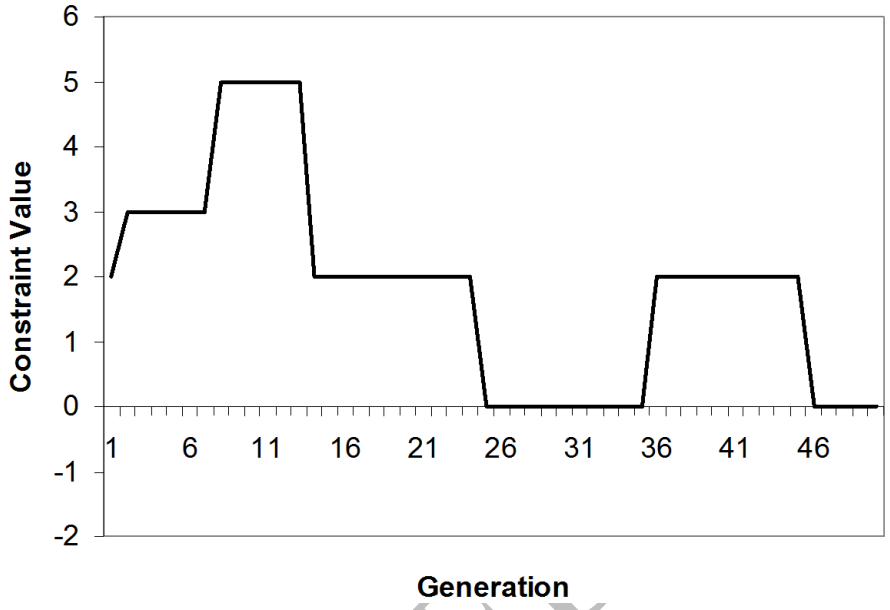


Figure 4. Constraint Value Solution History for Sum Game.

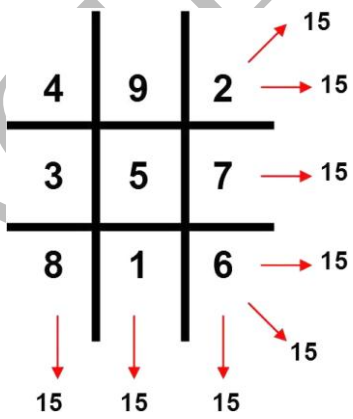


Figure 5. Solution Developed for the Sum Game.

string, but by generation 45, all integer values in the best decision string were unique. Note that there are multiple solutions to the game which can be seen by simple switching the first and third rows or columns. This issue is not considered here, as any valid solution to the game is being sought.

The way in which a problem is formulated can have a significant impact on the efficiency and effectiveness of a genetic optimization algorithm. The first formulation considered is valid, but by adding a constraint, the difficulty of the solution was considerably magnified. This increase in difficulty required the inclusion of a large population and additional generations to be executed. If algorithm efficiency is measured by the number of objective function and constraint evaluations, this formulation can be seen to have potential efficiency problems. If the calculation of the objective function and/or constraints is computationally expensive, this loss of efficiency can become a significant issue. Each objective function and constraint evaluation in this simple case requires little computational effort, but even for this case, it does raise the issue of formulation versus efficiency and effectiveness.

Certainly there is a benefit in linking the problem formulation and the encoding of the decision string as closely as possible to the physical nature of the problem itself. This strategy can often be developed for decision support problems through consideration of how an expert might approach the problem and mimicking their solution strategy. This leads naturally to a rule based formulation and encoding of the problem.

The initial encoding and solution approach is valid in that it did generate the desired solution. On the other hand, it may be regarded as a brute force approach where the encoding captures little of the strategy of solving the game from a given initial placement of integers. When a decision support task is being envisioned, the lack of capture or understanding of the strategy is a distinct disadvantage. A rule based encoding of the problem can be implemented in order to provide the link which leads to a viable decision support formulation. For the sum game, consider how a human player would work toward a solution to the game. It is most likely that a hand solution to the game would involve assigning the integers 1 through 9 to various positions in the game matrix. The user would then start exchanging the position of pairs of numbers in the game matrix to improve the larger differences among the various row, column and diagonal sums. Using this approach, some explicit strategy or rule set is applied in order to decide which game matrix positions to exchange. A positive outcome of this approach is that only feasible solutions can be generated in that no replication of integers can occur during the process. This exact approach may be built into the rule based encoding.

Care must be taken so as not to reduce the global search characteristic of the genetic approach, but other than this caution, virtually any rule encoding will work. In this particular case, the caution revolves around locating a temporary placement of the integers for which no single swap of two integers will improve the difference in the sums. This situation may be avoided by a number of strategies, including the implementation of a simulated annealing algorithm in order to accept intermediate designs which are not as good as the current one, particularly in the early stages of the search. Another approach would be to include some randomness in the rule set, which enhances the global nature of the algorithm. In this case, the rule encoding itself will be created so that the local minimum situation can be overcome by allowing the solution process to apply a number of simultaneous moves at any iteration.

The rule set developed can be very complicated, or alternatively, very simple in nature. The genetic algorithm has the capability of discovery and exploitation. This means that the user need not be overly concerned with the development of an intelligent rule set. In this case the rule set is based only upon the concept of selecting two integers in the matrix and exchanging them. No consideration of row and column sums are built into the rule set as the genetic algorithm will apply the rules in a way that the best objective is located. In order to execute the basic rule structure, four items need to be included in the encoding. These items define the row and column of each of the two positions to be switched. In order to avoid the trap of a local minimum, a number of simultaneous exchanges will be allowed.

For this example from one to three exchanges will be allowed. The number of exchanges is specified in the encoding and thus controlled by the genetic algorithm. Taking into account the above considerations, a rule based encoding for the simple sum game may be expressed as follows:

$$x = \left[N, r_{1a}, c_{1a}, r_{2a}, c_{2a}, r_{1b}, c_{1b}, r_{2b}, c_{2b}, r_{1c}, c_{1c}, r_{2c}, c_{2c}\right]$$

where

N indicates the number of exchanges to execute.

and

r_{1a}, c_{1a}, r_{2a}, c_{2a} represent the first pair of positions to exchange $(\text{row, column})_1 = (r_{1a}, c_{1a})$ with $(\text{row, column})_2 = (r_{2a}, c_{2a})$

and

r_{1b}, c_{1b}, r_{2b}, c_{2b}, r_{1c}, c_{1c}, r_{2c}, c_{2c} represent the other two exchange possibilities. This form of the encoding string is directly in line with the general structure presented in **Figure 2**.

As an example, consider the current game matrix configuration to be given by:

$$\text{Matrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and let the encoding string by set at.

$$x = \left[2, 2, 3, 1, 2, 2, 1, 3, 3, 1, 1, 3, 2\right]$$

the current game matrix would be altered to the form given by

$$\text{Matrix} = \begin{bmatrix} 1 & 6 & 3 \\ 9 & 5 & 2 \\ 7 & 8 & 4 \end{bmatrix}$$

The operation switches (row_2, column_3) with (row_1, column_2), which exchanges the 6 and 2 integer values. The encoding also switches (row_2, column_1) with (row_3, column_3) which exchanges the 4 and 9 integer values. The last possible exchange of (row_1, column_1) with (row_3, column_2) is not executed since the number of rules specified to be executed, N, is two. Whether or not this new configuration is accepted is a function of the objective function which is formulated as the difference between the maximum and minimum row, column and diagonal sum as defined in the first formulation.

On the surface the new formulation seems a bit complex in that a nine element decision string from the encoding in the first example has been replaced by a 13 element encoding string. The elimination of the constraint is a positive step, but the performance can be compared by looking at the results of executing the second or rule based formulation. Since the execution of the rule based formulation requires an initial game configuration, the integers are selected randomly without replacement. The objective function history is plotted in **Figure 6** as a function of the number of generations executed by the genetic algorithm. It is

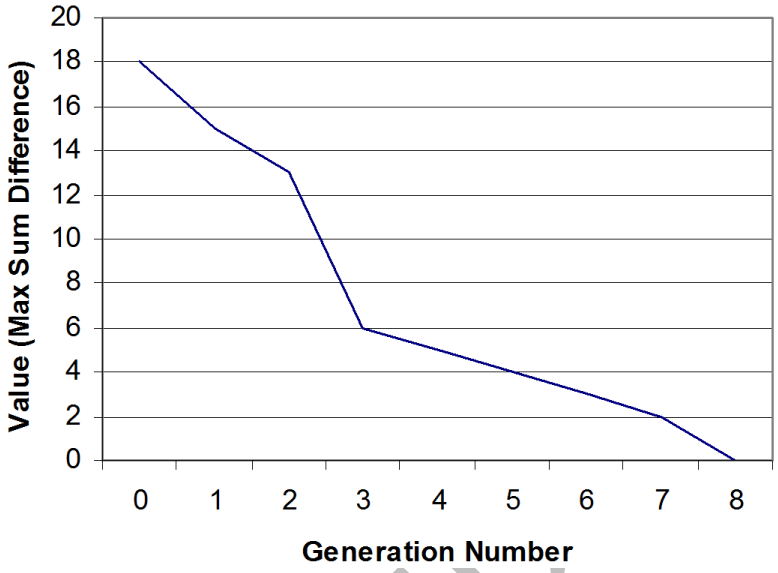


Figure 6. Solution History for Sum Game Using a Rule Based Approach.

seen in **Figure 6** that the solution was located by the eighth generation with a population size of 200. While additional coding was required to implement the rule interpretation and to modify the game matrix accordingly, the number of function evaluations required to solve the problem was reduced by a factor of almost 50. This is a significant difference and indicates the potential promise in a rule based approach for more difficult decision support problems.

5. A More Challenging Problem: The Maze

The solution of an arbitrary maze is a good representation for most decision support problems. Here, however, the simple rule based genetic algorithm will be considered. For a given maze configuration, the goal is to develop a sequence of moves that will lead from the starting point to the designated end point. A simple maze with one starting point and one exit point will be considered. The success of each move is dependent upon all moves made previously and as such which it shares with virtually any form of decision support application. The data file required to describe a particular maze must define the size of the maze (assumed here to be rectangular), and have some mechanism to define feasible and non-feasible moves from each block in the maze. The possible move directions are simple north, east, west and south which requires four potential move states be built into the encoding for each successive move through the maze. A sample maze is shown in **Figure 7** which will be utilized to demonstrate the approach. In **Figure 7**, the walls are delineated by solid lines, while the individual matrix locations are shown in lighter, dashed lines. Both a traditional genetic formulation and a rule based formulation will be considered. It should be noted that the solution could be potentially be generated using another type of evolutionary algorithm such as ant colony or swarm optimization [19] [20] [21].

For the traditional formulation, the encoding is expressed as a series of moves, where each move is represented by an integer value from zero to three. Thus the

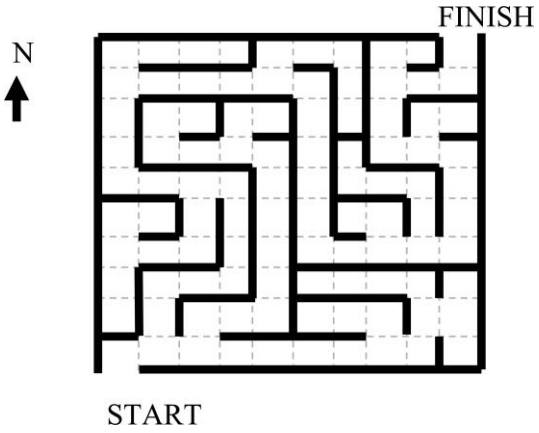


Figure 7. A Simple Maze.

encoding is given as:

$$x = \{x_1, x_2, x_3, \cdots, x_n\};\ \text{where } 0 \le x_i \le 3,\ x_i \text{ is an integer} \tag{8}$$

Here, n represents the total number of moves allowed for the decision support. The constraint of not being able to travel through a wall may be handled in a number of ways. Here it is arbitrarily dealt with by not allowing a move which passes through a wall. For example, if a north move is specified and this move would pass through a wall in the maze, the move is disallowed and the next move in the decision encoding is considered.

The objective function may also take on a number of forms as long as an indication of success is measured within the function. For this example, the objective function will be the distance from the position in the maze after executing every move specified in the decision encoding and the desired end point. The goal is to minimize this distance, with a value of zero being the limit which indicates a solution has been generated. This objective function may be expressed as follows:

$$\text{Minimize } f(x) = \left\{ \left(x_{\text{current}} - x_{\text{final}}\right)^2 + \left(y_{\text{current}} - y_{\text{final}}\right)^2 \right\}^{\frac{1}{2}} \tag{9}$$

This objective function has a problem when a path is found which terminates near the endpoint but is a dead end. This solution will be rated better than other members of the population which do not progress as far through the maze, but remain on productive tracts. Some penalty may be assigned to a dead end situation. This moves directly into the realm of problem specific knowledge and leads naturally into the rule based approach. For this example, the objective function defined above will be implemented without consideration to dead paths.

The comparison of the rule based and traditional genetic algorithms was limited in the number of trials for each method. Several population sizes were investigated, as were different random seeds and the number of generations allowed. Considerable attention was directed toward the standard genetic algorithm, as the ability to make significant progress in the maze solution was limited. The rule based code required very few trial runs, as the solution was able to be generated with very little tweaking of algorithm parameters. As the number of different problems addressed is limited, this is not a comprehensive comparison of the two approach-

es. Several different mazes of the same size were tested, and the results presented here on a single one of those tested. The results from the other maze solution attempts were similar in the performance of the algorithms. The goal here was not to make a detailed study on the comparative utility of the methods [22], but to demonstrate how the rule based method can be implemented.

The formulation developed above was executed using a standard genetic algorithm with a population of 500 with each decision string having a length of fifty elements. This resulted in the evaluation of 50,000 decision strings. The results from several runs started with different random seeds are shown in **Figure 8**. As can be readily seen, the final distance to the exit point in the maze was not located. This continued to be the case for larger population sizes and for larger numbers of generations. The best paths found terminated at local dead ends, which is not unexpected. By dramatically expanding the population size and executing many more generations, this difficulty could potentially be resolved. For a problem such as this, where the evaluation of the objective function is relatively inexpensive, this may represent a viable approach. As the objective function becomes increasingly expensive, however, this approach is not attractive.

From the rule based perspective, starting from an initial path or set of paths defined identically as in the previous formulation, the object is to execute a defined rule set in order to improve the path(s). The rule set for this problem consists of five separate rules. These rules are defined as follows:

1) Group all successful moves at the beginning of the encoding string.
2) For a selected group of encoding string positions, randomly alter the values.
3) For a selected move in the encoding string, alter the direction as specified.
4) For the first successful move in the encoding string after a specified position, try the same direction for the next number of specified moves (encoding positions).
5) Rectify the first specified number of back/forth moves, starting from a specified position in the encoding string.

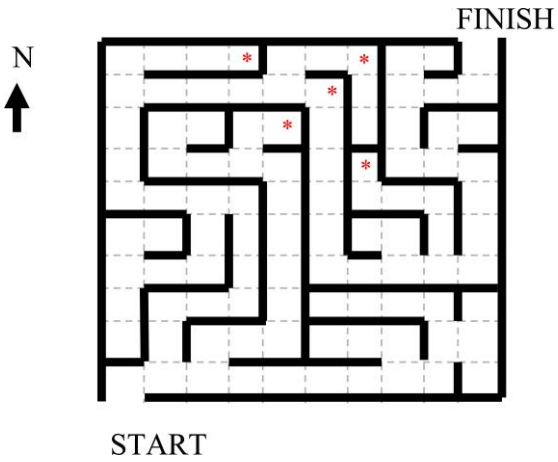


Figure 8. End Points for Conventional Genetic Algorithm Solution.

The first rule consolidates all successful moves at the beginning of the string, which allows the remaining positions to explore the maze from a given point in

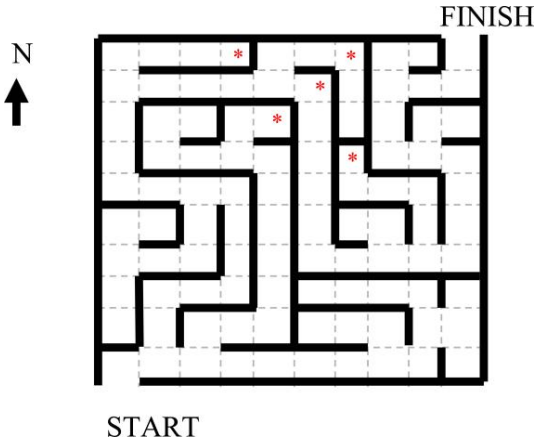
the maze. The second rule alters a set of encoding positions to random values. This rule was inserted in order to allow for a global search. The third rule operates on the same principle as does rule 2, but it only alters a single position to a value specified in the information segment of the rule encoding string. The fourth rule seeks to take advantage of the situation where a series of moves can be made in the same direction and the final rule eliminates situations where a move is directly countered by an opposite direction move (*i.e.* north followed immediately by south). These rules are easily implemented and while they insert some problem specific knowledge, they are very general in scope. More specific rules dealing with situations such as dead ends could be included, but they require additional code for the identification of such conditions.

The rule encoding for this example takes on a form very similar to that pictured in **Figure 2**. For this example, a maximum of five rules was allowed, each of which was assigned two informational positions in the string. The first position defines the number of rules to execute and the second defines which portion of the path to modify. This requires a total of 17 string positions to define the maximum length rule set (5 sets of three positions plus the first two). Note that this is a considerably smaller encoding string than for the conventional genetic solution approach which required a decision string of 50 elements.

Ten initial paths were generated by executing a conventional genetic algorithm. Only one generation was executed, with a population of 500 designs. Ten paths were selected based both of the distance to the end point of the maze and the difference among the paths traveled. This portion of the solution process is termed the phase one search. The second phase, executes the rules to modify the paths as described in the rule based approach. A population of 500 was maintained for the phase two search. The process was terminated as soon as the distance to the end point of the maze was reduced to zero. The rule based algorithm located the solution after 12 generations and required less than one third of the function evaluations for the failed searches of the conventional algorithm. The solution was consistently generated from multiple starting point groups.

The final path and prescribed moves are shown in **Figure 9**. Note that there are a few back and forth moves present in the final solution. As implemented, there is nothing in the formulation which seeks to avoid this situation. **Figure 10** documents the success frequency of each of the five rules. This success frequency is based upon the number of times a specific rule was executed successfully. Information such as this can be utilized to improve or alter the rule set. Note that all five of the rules were used successfully, although rules 3 and 4 and particularly rule 5 accounted for the largest number of successful moves. This makes some sense as the first rule, once executed, moves all successful moves to the front of the decision string and has little impact beyond that point.

Rule 2 allows for random move alterations which are important from maintaining a global search perspective. It would not, however, be expected to form a



Final String = ENNEENNNWWWEWNNNEEEEESSSSSEEENSENNNWWNNNEWSEENSNSN

Figure 9. Final Solution Generated by Rule Based Approach.

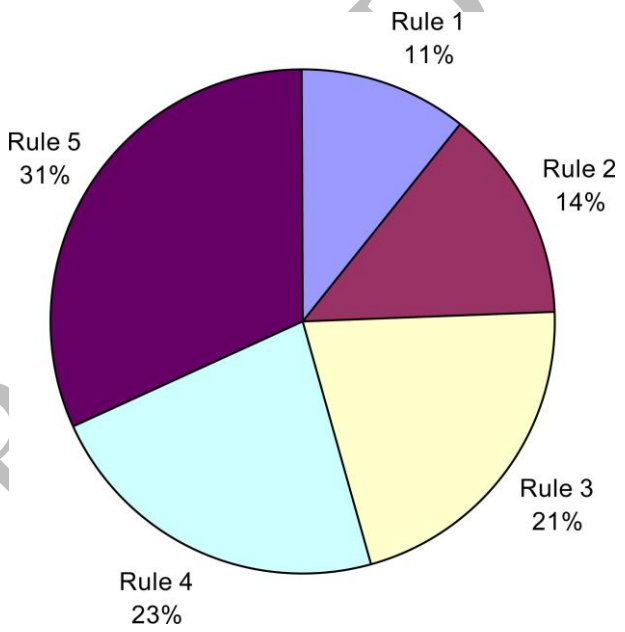


Figure 10. Successful Rule Execution Distribution.

large percentage of the successful moves. The remaining rule based moves perform specific functions during the search and each can be shown to have had a positive impact. Combinations or groups of rules applied simultaneously to generate a better decision string were not tracked, but this could be done quite easily.

6. Summary and Conclusions

A rule based approach has been successfully demonstrated on a small subset of decision support problems. While the ultimate effectiveness of the rule based genetic algorithm has not yet been determined, the concept has been presented. The rule base is conveniently encoded within the decision string of the genetic algorithm which makes the implementation of the approach a straightforward matter. For the sample problems, the rules are hard coded into the algorithm, which requires additional programming effort. The two phase approach allows for many

combinations of a conventional genetic search with a rule based approach. Phase one is utilized to locate suitable decision strings to operate upon. Phase two contains the execution of the rule based method. Hybrid methods, which automate the rule updates, as well as other genetic parameters during the solution, may be possible.

In both problems considered, significant improvement in both solution efficiency and efficacy were noted. The problems are simplistic in nature, but the concept may be implemented as presented for the solution of more difficult and significant applications. The comparison was only made to a standard generic algorithm, and there are other evolutionary methods that could be applied to the problems tested. The purpose herein was not to present the rule based method as a better approach to any other, but to demonstrate the basic approach as an impetus to consider more relevant applications. The ability to incorporate problem specific knowledge is of significant interest and potential benefit to any decision support problem.

The ability to determine which rules are being exploited successfully is also demonstrated and this feature can be further exploited to improve the rule base over time. The successes of the rules implemented were presented, but no effort was made to leverage this information on the problems tested. The balance between discovery based on randomness and a rule directed search is important. If randomness is removed completely, the search will be local which may not be very effective. The impact on the rules allowing for discovery in the genetic algorithm can be seen by looking at the rules which were successfully applied in the search. Extensions to other problem classes including the traveling salesman, manufacturing scheduling and structural design are currently being investigated.

References

[1] Kumar, S. (2016) Selection of Optimal Path in Routing Using Genetic Algorithm. *International Journal of Latest Trends in Engineering and Technology*, **7**, 259-263.

[2] Jensen, M.T. (2003) Generating Robust and Flexible Job Shop Schedules Using Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, **7**, 275-288. https://doi.org/10.1109/TEVC.2003.810067

[3] Alobaidi, W., Sandgren, E. and Alkuam, E. (2017) Decision Support through Intelligent Agent Based Simulation and Multiple Goal Based Evolutionary Optimization. *Intelligent Information Management*, **9**, 97-113. https://doi.org/10.4236/iim.2017.93005

[4] Sun, KT., Lin, Y.C., Wu, C.Y. and Huang, Y.M. (2009) An Application of the Genetic Programming Technique to Strategy Development. *Expert Systems with Applications*, **36**, 5157-5161. https://doi.org/10.1016/j.eswa.2008.06.066

[5] Compare, M. and Martini, F. (2015) Decision Support: Genetic Algorithms for Condition Based Maintenance Optimization under Uncertainty. *European Journal of Operations Research*, **16**, 611-623. https://doi.org/10.1016/j.ejor.2015.01.057

[6] Guido, R. and Conforti, D. (2015) A Hybrid Genetic Approach for Solving an Integrated Multi-Objective Operating Room Planning and Scheduling Problem. *Computers and Operations Research*.

[7] OnkHam, W., Karwowski, W. and Ahram, T.Z. (2012) Economics of Human Performance and Systems Total Ownership Cost. *Work*, **41**, 2781-2788.

[8] Liang, P.L. and Jones, C. (1987) Design of a Self-Evolving Decision Support System. *Journal of Management Information Systems*, **4**, 59-82.

https://doi.org/10.1080/07421222.1987.11517786

[9] Lin, S. and Kernighan, B.W. (1973) An Efficient Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, **21**, 498-516. https://doi.org/10.1287/opre.21.2.498

[10] Nonas, E. and Poulovassilis, A. (1999) Optimising Self Adaptive Networks by Evolving Rule-Based Agents. *Lecture Notes in Computer Science*, **1596**, 203-214. https://doi.org/10.1007/10704703_17

[11] Ishibuchi, H. and Yamamoto, T. (2004) Fuzzy Rule Selection by Multo-Objective Genetic Local Search Algorithms and Rule Evaluation Measures in Data Mining. *Fuzzy Sets and Systems*, **141**, 59-88. https://doi.org/10.1016/S0165-0114(03)00114-3

[12] Hadian, A., Nasiri, M. and Minaci-Bidgoli, B. (2010) Clustering Based Multi-Objective Rule Mining Using Genetic Algorithm. *International Journal of Digital Content Technology and Its Application*, **4**, 37-42. https://doi.org/10.4156/jdcta.vol4.issue1.5

[13] Caponetto, R., Fortuna, S.F. and Xibilia, M.G. (2003) Chaotic Sequences to Improve the Performance of Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computing*, **7**, 289-304. https://doi.org/10.1109/TEVC.2003.810069

[14] Mozaffari, A., Emami, M., Azad, N.L. and Fathi, A. (2015) On the Effcacy of Chaos-Enhanced Heuristic Walks with Nature-Based Controllers for Robust and Accurate Intellegent Search. *Journal of Experimental & Theoretical Artificial Intelligence*, **27**, 389-422. https://doi.org/10.1080/0952813X.2014.954632

[15] Srinivasan, S. and Ramakrishnan, S. (2013) A Social Intellegent System for Multi-objective Optimization of Classification Rules Using Cultural Algorithms. *Computing*, **95**, 327-350. https://doi.org/10.1007/s00607-012-0246-4

[16] Webb, D. and Sandgren, E. (2017) Topological Design via a Rule Based Genetic Optimization Algorithm. *Computers and Structures*, Submitted.

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

[18] Davis, L. (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

[19] Imam, A.A, Much, A.M. and Alamsyah, A. (2016) Comparison of Genetic Algorithm and Ant Colony Optimization in Course Scheduling Optimization. *Scientific Journal of Informatics*, **3**, 51-60.

[20] Wang, L., Cai, J., Li, M. and Liu, Z. (2017) Flexible Shop Scheduling Problem Using an Improved Ant Colony Optimization. *Scientific Programming*, **2017**, Article ID: 9016303. https://doi.org/10.1155/2017/9016303

[21] Schyns, M. (2015) Discrete Optimization: An Ant Colony System for Responsive Dynamic Vehicle Routing. *European Journal of Operational Research*, 245, 704-718.

[22] Sandgren, E. and Ragsdell, K.M. (1980) The Utility of Nonlinear Programming Algorithms: A Comparative Study—Part I. *Journal of Mechanical Design*, *Transactions of the ASME*, **102**, 540-546. https://doi.org/10.1115/1.3254782