A Rule Based Evolutionary Algorithm for Intelligent Decision Support

Eric Sandgren [a],
David Webb [b]
and Jan B. Pedersen [a,*]

[a] *College of Engineering, University of Nevada, Las Vegas, 4505 Maryland Parkway, Box 454005, Las Vegas, NV 89154-4005.*
E-mail: eric.sandgren@unlv.edu, matt@cs.unlv.edu

[b] *Philip Morris USA, 6601 West Broad St., Richmond, VA, 23230*
E-mail: david.J.webb@pmusa.com

Genetic or evolutionary search algorithms seek and exploit the structure of the problem encoding employed in the application. This exploitation can be slow and result in solutions containing obvious flaws. A rule based structure is proposed which seeks to inject domain specific knowledge into the search process and allow for a more intuitive design encoding. This approach significantly reduces the size of the problem encoding which in turn reduces solution time as well as being able to produce a more robust solution. Success or failure in improving the decision strategy as related to the history of which rules or combination of rules are successful allows for rule refinement as well as knowledge capture. This, in turn, yields a better understanding of the decision process. An approach for implementing rules within the design encoding is demonstrated and several problems are solved using the technique with the results compared to those generated by a conventional genetic algorithm.

Keywords: Decision support systems, Planning, Scheduling, Learning, Information Systems

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, scheduling, bin packing and game strategy. 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, slow final convergence and the potential of "noise" or unwanted characteristics appearing in the final solution. Evolutionary algorithms possess the trait of discovery, 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 decision strategy.

In practice, most decision support activities are highly dependent upon human intervention. Limited success in the application of evolutionary algorithms to various decision support application areas has been achieved by several investigators [1,4,6,10,11], 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. For these reasons, it makes sense to add a level of computational sup-

*Corresponding author: Jan B. Pedersen, College of Engineering, University of Nevada, Las Vegas, 4505 Maryland Parkway, Box 454005, Las Vegas, NV 89154-4005.

AI Communications
ISSN 0921-7126, IOS Press.

port to aid the human in making key decisions. 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, but 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. Rule based approaches have been applied to agent based systems involving communication networks [8] and in some cases, genetic algorithms were employed to optimize the rule selection. The application of simulated annealing to the traveling salesman problem [7] uses rules including moving groups of cities and reversing the order of a group of cities from the current city visit schedule or strategy. 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 [9] or in searching for rules that predict outcomes in data sets [5]. 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 [12]. 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 where 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:

$$\text{Minimize } f(x); \text{where } x = [x_1, x_2, \ldots, x_n]^T \quad (1)$$

Subject to

$$g_j(x) > 0; j = 1, 2, \ldots, J \quad (2)$$

$$h_k(x) = 0; k = 1, 2, \ldots, K \quad (3)$$

With

$$x_i^{low} < x_i < x_i^{high} \quad (4)$$

A design configuration or a decision strategy to a genetic algorithm is 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:

1. The genetic algorithm operates by manipulation of the coding of the set of gene values composing the chromosome.
2. A population of designs or decision strings is considered rather than a single design or decision point.
3. The transition from one set of designs or decisions to the next are probabilistic rather than deterministic.
4. 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 not as 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 as a parent for the next generation. The chromosomes possessing the greatest fitness have the highest probability of becoming parents, but even the least fit member has a finite probability of being selected. The parent strings are combined using 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 and the rules which govern how parents are selected and combined to produce offspring must be defined. Special operators such as mutation are included in order to guard against the loss of important design or decision information 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 which 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. Even problems requiring significant simulation times to access the outcome of a decision strategy and those that require large populations due to the size of the decision matrix are practical due to the parallel nature of the solution process. Additional detail concerning conventional genetic algorithms is given by Goldberg in [3] and Davis in [2].

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 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 optimizer. Some randomness must be maintained in the execution of the rule base, and there must be a mechanism for simultaneously firing multiple rules in order to transform a single decision string to a better state. The randomness helps to keep the search global in nature and the ability to operate on a single decision strategy with multiple rules 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

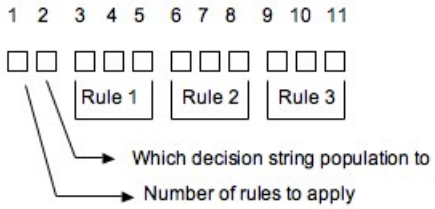


Fig. 1. A rule based encoding for a genetic algorithm.

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 subse-

quent groups of elements are used to define which specific rule or rules to apply with specific information blocks which define how each rule is to be executed. The fact that there are only three information blocks within each rule definition group in Figure 1 is of no consequence as the number can be arbitrarily expanded as needed. In order to ensure consistency in the crossover operation for generating offspring, however, there is a benefit in keeping 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 which when 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

A number of other potentially effective rules could be easily 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 could be executed to form this population or it could be formed through a random selection, much as the population for the first generation in a conventional genetic solution process. A subset 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 is far more efficient than solving a very large problem. This has a significant impact on convergence and the computational time required.

4. An 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

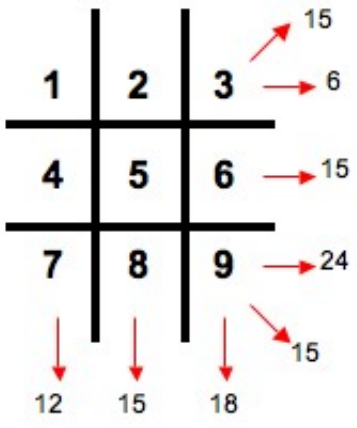


Fig. 2. Simple sum game at initial setting.

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 one, is not easily solved. The approaches considered here 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 implemented approach. The first formulation of the problem may now be expressed in mathematical form as follows:

$$\text{Minimize } F(x) =$$
$$max(\sum(row_i, column_j, diagonal_k))$$
$$-min(\sum(row_i, column_j, diagonal_k)) \qquad (5)$$

Where

$$x = x_1, x_2, x_3, \ldots, x_9, 0 < x_i < 10 \wedge$$
$$x_i \text{ is an integer} \qquad (6)$$
$$i = row(1,2,3)$$
$$j = column(1,2,3)$$
$$k = diagonal(1,2)$$

And

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

where

$$N_{dup} = \text{total \# of reps. of integers in } x.$$

This formulation was executed using a conventional genetic algorithm and the objective function solution history is documented in Figure 3. The solution history of the constraint value is plotted in Figure 4. An exterior penalty function was uti-

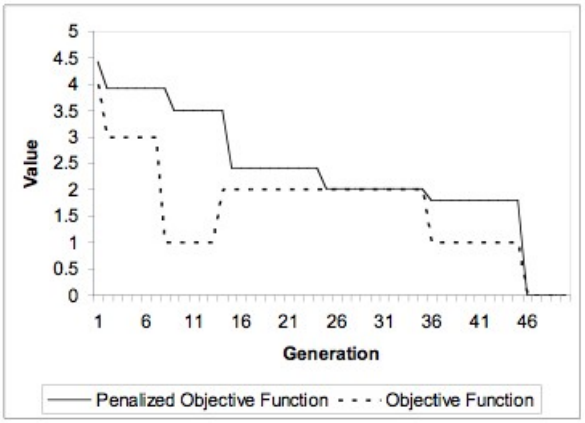


Fig. 3. Objective function solution history for sum game problem.

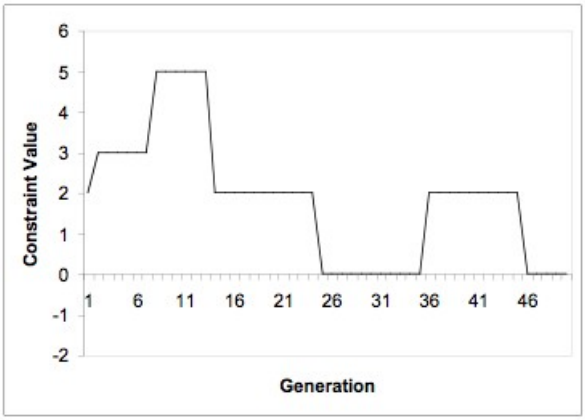


Fig. 4. Constraint value solution history for sum game.

lized 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

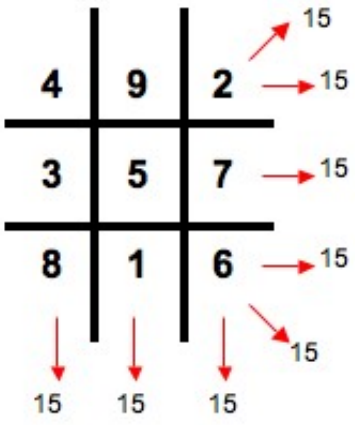


Fig. 5. Solution Developed for the Sum Game.

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 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 were 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 build 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 al-

lowing 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, where the number of exchanges are 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:

$$x = [N, r_1^a, c_1^a, r_2^a, c_2^a, {}_1^b, c_1^b, r_2^b, c_2^b, r_1^c, c_1^c, r_2^c, c_2^c]$$

where N indicates the number of exchanges to execute and $r_1^a, c_1^a, r_2^a, c_2^a$ represent the first pair of positions to exchange $(row, column)_1 = (r_1^a, c_1^a)$ with $(row, column)_2 = (r_2^a, c_2^a)$ and $r_1^b, c_1^b, r_2^b, c_2^b$; $r_1^c, c_1^c, r_2^c, c_2^c$ 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:

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

and let the encoding string by set at

$$x = [2, 2, 3, 1, 2, 2, 1, 3, 3, 1, 1, 3, 2]$$

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

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

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 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.

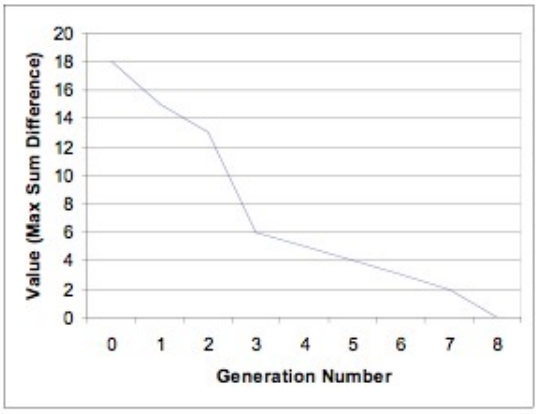


Fig. 6. Solution history for sum game using a rule based approach.

5. A More Challenging Problem: The Maze

The solution of an arbitrary maze is a good representation for most decision support problems. 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. 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. 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 encoding is given as:

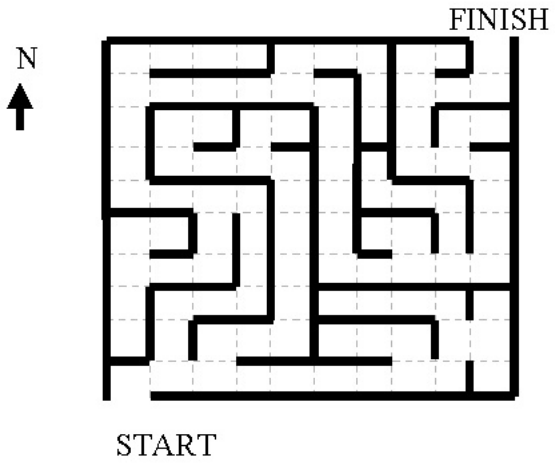


Fig. 7. A simple maze.

$$x = [x_1, x_2, x_3, \ldots, x_n] \text{ where}$$

$$0 < x_i < 3, x_i \text{ is an integer} \quad (8)$$

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 moves 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:

$$\text{Minimize } f(x) = \sqrt{(x_{current} - x_{final})^2 + (y_{current} - y_{final})^2}$$

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, but 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 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. The algorithm was allowed to progress through 100 generations. 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. 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

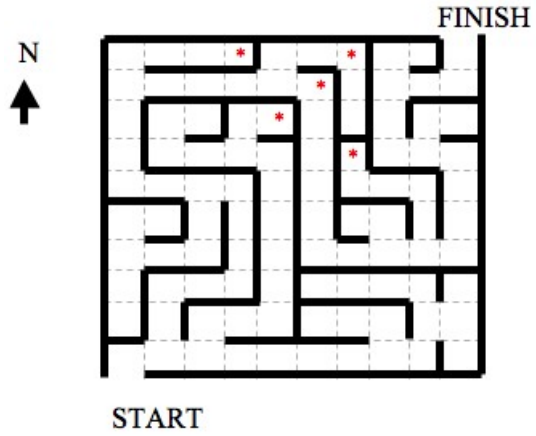


Fig. 8. End points for conventional genetic algorithm solution

perspective, starting from an initial path or set of paths defined identically as in the previous formulation, the object is to execute a 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.

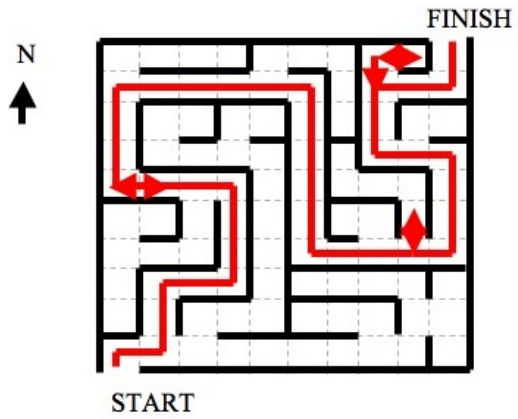
The first rule consolidates all successful moves at the beginning of the string, which allows the remaining positions to explore the maze from a given point in the maze. The second rule alters a set of encoding positions to random values. This rule was inserted in order to allow for a global search. The third rule, operates on the same principle as does rule 2, but it only alters a single position to a value included 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. With the first position defining the number of rules to execute and the second defining which path to modify, this requires a total of 17 string positions to define the maximum 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 the conventional genetic algorithm as defined in the original solution. 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 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, but there is nothing in the formulation which seeks to avoid this situation. Figure 10 documents the percentage use of each of the five rules. This percentage is based upon the number of times a specific rule was executed successfully. This information can be utilized to improve or alter the rule set. Note that each 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 successfully 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 is important from maintaining a global search perspective, but it would not be expected to form a large percentage of the successful moves. The remaining 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.



Final string: ENNEENNNWWWEWNNNEEEEESSSSS EEENSENNNWWNNNEWSEENSNSN

Fig. 9. Final solution generated by rule based approach.

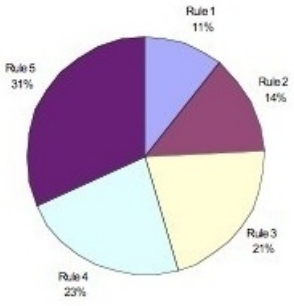


Fig. 10. Successful rule execution distribution.

6. Summary and Conclusions

A rule based approach has been successfully demonstrated on a small subset of decision support problems. The rule base is conveniently encoded within the decision string of the genetic algorithm which makes the implementation of the approach a straightforward matter. The two phase approach allows for many combinations of a conventional genetic search with a rule based approach. In both problems considered here, significant improvement in both solution efficiency and efficacy were noted. The ability to incorporate problem specific knowledge is of significant interest and 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. Extensions to other problem classes including the traveling salesman and manufacturing scheduling problem are currently being investigated.

References

[1] N. Adachi, M. Sato, and S. Kobayashi. Application of Genetic Algorithm to Flight Schedule Planning. *Systems and Computers in Japan*, 35(12):83–92, 2004.

[2] L. David. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

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

[4] M. Hosny and C. L. Mumford. Single Vehicle Pickup and Delivery with Time Windows: Made to Measure Genetic Encoding and Operators. In *Proceedings of the 2007 Genetic and Evolutionary Computation GECCO 07*, pages 2489–2496. ACM Press, 2007.

[5] H. Ishibuchi and T. Yamamoto. Fuzzy Rule Selection by Multo-Objective Genetic Local Search Algorithms and Rule Evaluation Measures in Data Mining. *Fuzzy Sets and Systems*, 141(1):59–88, 2004.

[6] M. T. Jensen. Generating Robust and Flexible Job Shop Schedules Using Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):175–288, June 2003.

[7] S. Lin and B. W. Kernighan. An Efficient Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, 21(2):498–516, 1973.

[8] E. Nonas and A. Poulovassilis. *Optimising Self Adaptive Networks by Evolving Rule-Based Agents*, volume 1596 of *Lecture Notes in Computer Science (LNCS)*. Springer Verlag, 1999.

[9] T. Onoyama, T. Maekawa, S. Kubota, Y. Taniguchi, and S. Tsuruta. Intelligent Evolutional Algorithm for Distribution Network Optimization. In *Proceedings of the IEEE 2002 International Conference on Control Applications*, pages 802–807, 2002.

[10] A. Persson, J. Ekberg, H. Grimm, S. Falk, A. Ng, P. Stablum, and T. Lezama. Simulation-Based Multi-Objective Optimization of a Real-World Scheduling Problem. In *Proceedings of the IEEE 2006 Winter Simulation Conference*, pages 1757–1764, 2006.

[11] M. Russel and G. B. Lamont. A Genetic Algorithm for Unmanned Aerial Vehicle Routing. In *Proceedings of the 2005 Genetic and Evolutionary Computation GECCO 05*, pages 1523–1530. ACM Press, 2005.

[12] D. Webb and E. Sandgren. Topological Design via a Rule Based Genetic Optimization Algorithm. Submitted to Computers and Structures.