Ann Oper Res (2010) 174: 185–199
DOI 10.1007/s10479-009-0562-z

NeuroGenetic approach for combinatorial optimization: an exploratory analysis

Anurag Agarwal · Selcuk Colak · Jason Deane

Published online: 26 May 2009

Abstract Given the NP-Hard nature of many optimization problems, it is often impractical to obtain optimal solutions to large-scale problems in reasonable computing time. For this reason, heuristic and metaheuristic search approaches are used to obtain good solutions fast. However, these techniques often struggle to develop a good balance between local and global search. In this paper we propose a hybrid metaheuristic approach which we call the NeuroGenetic approach to search for good solutions for these large scale optimization problems by at least partially overcoming this challenge. The proposed NeuroGenetic approach combines the Augmented Neural Network (AugNN) and the Genetic Algorithm (GA) search approaches by interleaving the two. We chose these two approaches to hybridize, as they offer complementary advantages and disadvantages; GAs are very good at searching globally, while AugNNs are more proficient at searching locally. The proposed hybrid strategy capitalizes on the strong points of each approach while avoiding their shortcomings. In the paper we discuss the issues associated with the feasibility of hybridizing these two approaches and propose an interleaving algorithm. We also provide empirical evidence demonstrating the effectiveness of the proposed approach.

Keywords Optimization · Metaheuristics · NeuroGenetic · Genetic algorithms · Neural networks

A. Agarwal (✉)
Department of Information Systems and Decision Science, College of Business, University of South Florida, Sarasota, FL 34243, USA
e-mail: agarwala@sar.usf.edu

S. Colak
Department of Business, College of Economics and Administrative Sciences, Cukurova University, Adana, Turkey
e-mail: scolak@cu.edu.tr

J. Deane
Department of Business Information Technology, Pamplin College of Business, Virginia Tech, Blacksburg, VA, USA
e-mail: jason.deane@vt.edu

1 Introduction

Discrete combinatorial problems such as knapsack, scheduling and the traveling salesman problem appear frequently in many domains including computer science, industrial engineering, manufacturing and operations management. Given the NP-hard nature of many of these problems, obtaining optimum solutions for large problems using exact methods is impractical in terms of computational time. Heuristics and metaheuristics are therefore commonly used as search mechanisms for obtaining satisfactory solutions in reasonable time. Over the last two decades, various intelligent metaheuristic search procedures such as tabu search (Glover 1989), genetic algorithms (Goldberg 1989; Chu and Beasley 1998), neural networks (Hopfield and Tank 1985), augmented neural networks (Agarwal et al. 2003, 2006), simulated annealing (Kirkpatrick et al. 1983; Cho and Kim 1997), ant-colony optimization (Dorigo et al. 1999; Merkle et al. 2002), electromagnetism (Birbil and Fang 2003), the great-deluge algorithm (Dueck 1993) etc. have been proposed. Some of these metaheuristics, such as genetic algorithms, neural networks and ant-colony optimization have been inspired by biological phenomena. Many others such as simulated annealing, electromagnetism and the great-deluge algorithms have been inspired by various other phenomena observed in nature.

The idea behind metaheuristic search is to iteratively search potentially good feasible solution neighborhoods, so that a reasonably good solution is obtained quickly. The tradeoff in search lies between what are termed global and local search. Strategies that tend to intensify search in a local neighborhood run the risk of being trapped in that neighborhood, thus making it difficult to explore other good neighborhoods. Alternatively, strategies that thin out the search to gain access to many neighborhoods often fail to sufficiently intensify their local search. Attempting to balance the local-search and global-search performance so as to generate solutions as close to optimal as possible is a recurring challenge faced by most metaheuristic procedures. Tabu search, for example, proposes a diversification strategy for global search and an intensification strategy for local search; Simulated Annealing proposes a high temperature value for global search and a lower temperature value for local search; the magnitude of temperature parameter being correlated to the magnitude of perturbation. Generally, mechanisms that tend to be strong in global search tend to be weak in local search and vice versa. For example, genetic algorithms have been found to be particularly strong in global search, but somewhat weak in local search (Gen and Cheng 1997). On the other hand, the augmented-neural-network approach has been found to be strong in local search, but weak in global search (Agarwal 2009). Hybrid searches which draw on the strengths of two or more approaches have also been proposed in the literature (Osman 1993; Glover et al. 1995; Debels et al. 2006). Combining a deterministic local search, such as 2-OPT or 3-OPT for the traveling salesman problem (Lin and Kernighan 1973) or double justification for scheduling problems (Tormos and Lova 2001), with global search techniques such as genetic algorithms is quite common. Combining non-deterministic local search approaches such as tabu search with genetic algorithms is also common (Glover et al. 1995).

In this paper we consider the augmented-neural-network (AugNN) approach, which is a non-deterministic local-search approach, as a candidate for combining with genetic algorithms. We call the proposed hybrid metaheuristic the NeuroGenetic approach (NG). Although the AugNN approach is relatively new compared to other long-established approaches such as Tabu Search and Simulated Annealing, the AugNN approach has been shown to outperform Tabu Search and Simulated Annealing for certain scheduling problems (Colak and Agarwal 2005; Colak et al. 2006). In the next section we discuss the motivation behind our choice of combining the AugNN and the GA approaches and discuss

	GA	AugNN
1.	Good at global search	Not so good at global search
2.	Not so good at local search	Good at local search
3.	Fast	Slow
4.	Needs thousands of iterations for good solutions	Needs tens or hundreds of iterations for good solutions
5.	Does not rely on heuristics	Relies on heuristics
6.	Slow and steady improvement in the solution quality	Finds most of the improvement early in the search process

Fig. 1 Contrasting advantages and disadvantages of GA and AugNN approaches

the challenges involved in hybridizing them. In Sect. 3, we discuss the feasibility of such hybridization and provide some algorithms and strategies for this hybridization process. In Sect. 4 we discuss some empirical results to demonstrate and validate the effectiveness of the approach. In the concluding section, we summarize the paper, offer some concluding remarks and propose some ideas for future work.

2 NeuroGenetic approach

2.1 Motivation

First we attempt to explain our motivation for hybridizing the AugNN and GA approaches. Figure 1 provides some contrasting advantages and disadvantages of the AugNN and the GA approaches.

In comparing these two techniques, we noticed that the advantages and disadvantages of the two approaches are very complementary and this observation was the motivation behind our initial consideration of potentially hybridizing these two approaches. The most striking difference in the techniques is obviously their contrasting strengths in local and global search. But in addition to that, in our experience with the AugNN and GA approaches, we noticed that most of the improvement with the AugNN occurs early in its search process, i.e. the first 100 iterations, whereas GAs on the other hand tend to improve slowly and steadily throughout a much longer search process. We posit that if we could successfully interleave the searching of these two techniques, iteratively feeding solutions from one to the other, that the overall search performance should be better than either approach alone. The basic idea being to allow our GA to run on a population of chromosomes derived at least partially from AugNN solutions and then to allow the AugNN to attempt to improve upon the best found GA chromosomes and so on. In completing this iterative process we are essentially alternating between global search steps using the GA and local search steps using the AugNN. In an effort to take advantage of the fact that the AugNN tends to exhaust its potential solution improvement relatively quickly in terms of the number of iterations required, whereas the GA provides slow and steady improvement, we recommend that the AugNN interludes be much shorter than those of the GA. Figure 2 shows a schematic of this interleaved approach.

2.2 Challenges

While in theory, the idea of hybridizing to draw upon the strengths of two (or more) meta-heuristics is appealing, the actual implementation may not necessarily be trivial, mainly on

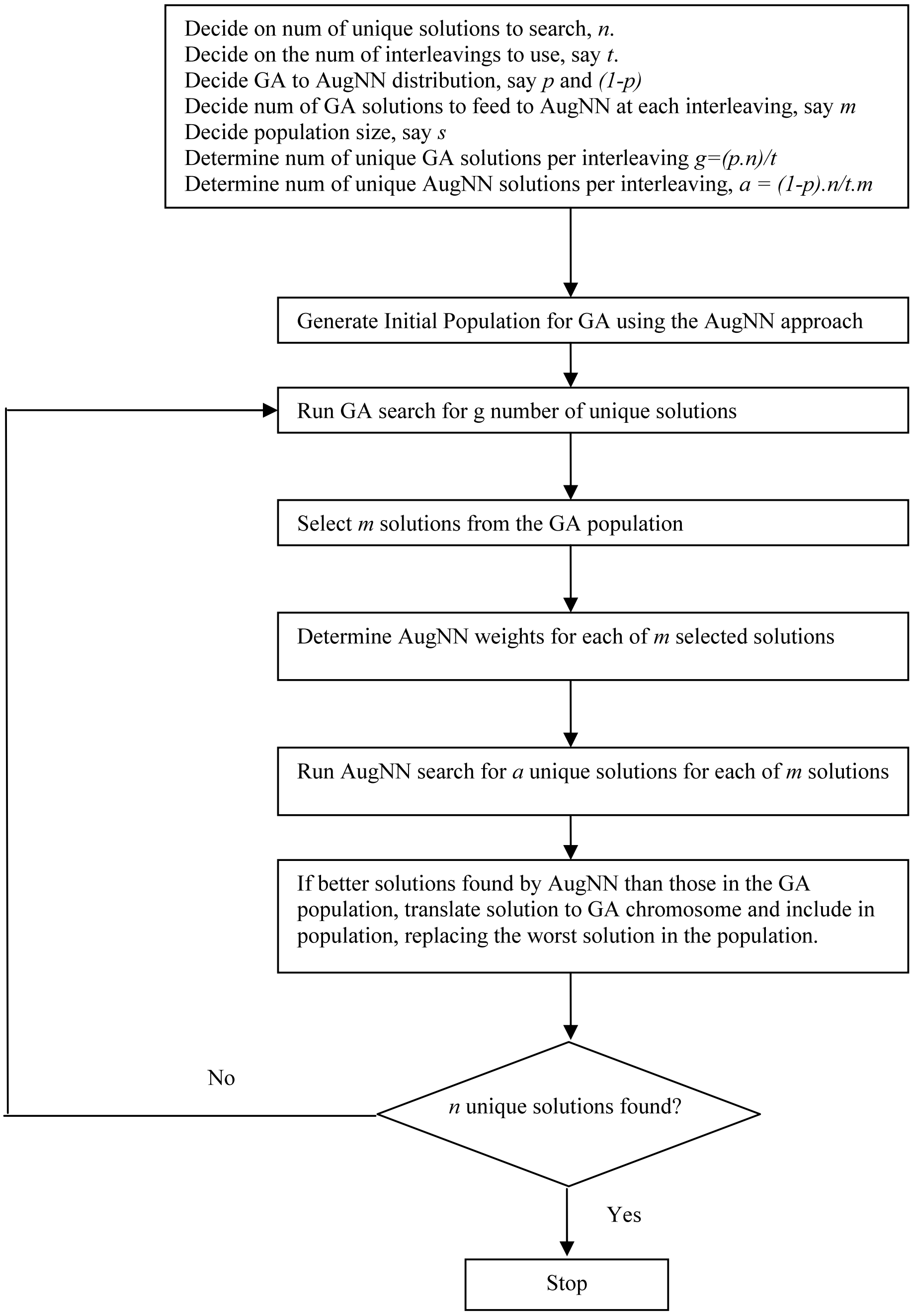
Decide on num of unique solutions to search, *n*.
Decide on the num of interleavings to use, say *t*.
Decide GA to AugNN distribution, say *p* and *(1-p)*
Decide num of GA solutions to feed to AugNN at each interleaving, say *m*
Decide population size, say *s*
Determine num of unique GA solutions per interleaving *g=(p.n)/t*
Determine num of unique AugNN solutions per interleaving, *a = (1-p).n/t.m*
Generate Initial Population for GA using the AugNN approach
Run GA search for g number of unique solutions
Select *m* solutions from the GA population
Determine AugNN weights for each of *m* selected solutions
Run AugNN search for *a* unique solutions for each of *m* solutions
If better solutions found by AugNN than those in the GA population, translate solution to GA chromosome and include in population, replacing the worst solution in the population.
n unique solutions found?
No
Yes
Stop

Fig. 2 Schematic of the NeuroGenetic approach

account of the differing nature of the encoding schemes required by the various approaches for a given problem type. For example, when using GAs, depending on the problem type, an appropriate chromosome string encodes the solution, either explicitly or implicitly. For solving the 0–1 knapsack problem, for instance, a chromosome string of 0s and 1s of length equal to the number of items represents a solution explicitly. Each position in the string represents one item. Those items represented by a one in the string have been added to the knapsack and those represented by a 0 have not. For scheduling problems, on the other hand, a chromosome of length equal to the number of activities represents a potential solution implicitly. The string merely represents the order in which the activities are to be assigned—the actual clock-times of assignment for each activity is determined from the given chromosome's ordering of activities. When using the AugNN approach, on the other hand, no such chromosome strings are used for encoding solutions; instead, a weight matrix in conjunction with a heuristic is used to build a solution. Simulated annealing, ant-colony optimization and tabu-search approaches use encoding schemes which are altogether different from those of genetic algorithms and AugNNs and also different from each other. The challenge of hybridizing, therefore, is to develop a strategy that permits switching between the encodings of the alternative metaheuristics.

Next we discuss the challenges specific to AugNN and GA interleaving. We observe that in the AugNN approach, the solutions are obtained in a sequence with each iteration requiring a weight matrix. In order to obtain the solution in the tth iteration, the weight matrix at the end of the $(t-1)$th iteration is required. The first solution assumes an identity matrix. So an issue that needs to be addressed is—if we are to interleave AugNN search with GA search, then we should be able to take a given GA solution and improve upon it using AugNN search. This, in turn, requires that there be a way to determine the weight matrix which in conjunction with a heuristic will generate a given GA solution in essence feeding a good GA solution into the AugNN approach. Whether determining such a weight matrix is theoretically feasible is not obvious and may not be possible in all cases. The feasibility of determining the weight matrix is critical, for if it is not feasible, then the idea of interleaving GA with AugNN would also not be feasible. Similarly, a GA search goes from one generation to the next in a sequence. Generation n is derived from generation $n-1$. So we have a similar challenge in the reverse direction—is it possible to take an AugNN solution and translate it into a chromosome string so that GA search can incorporate good AugNN solutions as chromosomes in future generations? In the next section, we address these feasibility of hybridization of AugNN with GA.

3 Feasibility of hybridization

As discussed above, in order to successfully interleave GA and AugNN search, it should be feasible to determine a set of weights which in conjunction with a given heuristic is able to generate a given GA solution, and vice versa, it should be feasible to translate an AugNN solution into a GA chromosome string. In order to understand how the encodings between genetic algorithms and neural networks can be switched, it is first necessary to review and understand the encodings of each of these techniques.

3.1 AugNN encoding

In this section we describe the basics of the AugNN encoding scheme so that the switching procedure can be understood. Some notation is necessary to understand the encoding:

ρ: represents an optimization problem, expressed as $\rho : \Theta = f(\chi)$ s.t. ψ where Θ is the objective function, a function of χ. χ represents the problem parameters and ψ represents a set of constraints. We consider only linear integer programming problems because to be solvable by this method, a problem needs to satisfy three conditions (i) it should have a good priority-based heuristic since a good priority-based heuristic is integral to the AugNN approach, (ii) it should be solvable using GAs and (iii) the encoding scheme of the GA should allow switching to AugNN encoding and vice versa. Typically non-linear, non-integer problems will not satisfy all three conditions whereas linear integer programming problems will.

Θ_{opt}: represents the optimal solution of ρ.

σ: represents a deterministic heuristic that solves ρ to an approximate solution $\Theta_{\sigma,}$. It is assumed that σ attempts to find the best possible solution, i.e. if the solution space is convex, then σ will attempt to find a solution as close to the boundary solutions as possible. We also assume that these heuristics work by prioritizing some problem-specific parameter of the problem.

n: the length of the problem. For example for a scheduling problem ρ, n would be the number of activities. χ would be the set of a vector of values of processing times and amounts of resources needed. The constraint set ψ would be the precedence matrix and the constraint on the availability of resources. For a knapsack problem, n would be the number of objects, χ would be value of the objects and the resource needed for each object. ψ would be the capacity constraints for the resources.

Υ: represents the problem specific parameters. Using Υ, a given σ produces a permutation $\zeta(\Upsilon_\sigma)$ of some aspect of the problem. For example, for the scheduling problem, utilizing the heuristic "Longest Processing Time Next", Υ is the permutation vector in which activities are sorted in non-decreasing order of their processing times. For the knapsack problem, depending on the heuristic being used, Υ could be the vector of pseudo-utility ratios of the objects. Υ is assumed to be of length m. We also assume that all elements of Υ are positive.

$w \in \Re^m$: Weight vector of m real numbers. Without loss of generality, we can talk about a weight vector instead of a weight matrix.

η: Some feasible solution of ρ.

The encoding for an AugNN is implicitly contained in the set of σ, Υ and $w \in \Re^m$. The application of the heuristic σ to the vector $(w.\Upsilon)$ produces a solution. Agarwal (2009) proposes and proves the following theorem:

Theorem *Given an integer linear optimization problem, $\rho : \Theta = f(\chi)$ s.t. ψ and a deterministic heuristic σ which uses a problem-specific parameter vector Υ of length m to find the best possible solution of ρ, then for each solution η of ρ, there exists a set Ω of weight vectors $w \in \Re^m$ such that the application of the heuristic σ using the weighted parameter vector $(w.\Upsilon)$ can produce η of the problem ρ and that $|\Omega| = \infty$.*

This theorem basically says that given a solution η of a problem ρ, there exists an infinite number of weight vectors, w, which in conjunction with the given heuristic can produce the given solution. The existence of the required weight matrix is thus established by this theorem. For proof, see Agarwal (2009). In this encoding, σ and Υ are constant so the only thing that varies is the weight vector w. Although, based on this theorem, we know that the sought after weight vectors exist, whether or not it is possible to find such a weight vector represents a different challenge. One of the contributions of this paper is that we show that

The AugNN Algorithm

```
Initialize Weights
Repeat Until NumOfUniqueSolutions = RequiredNumOfUnique Solutions
    Run SinglePassHeuristic
    Check if the solution is unique
    If the solution is Unique then
        If solution is better than the best solution then save solution as the best solution
        NumOfUniqueSolutions ++
        Save this solution in a list of past solutions
    End If
    Modify the Weights
End Repeat
Display best solution
```

Fig. 3 The AugNN algorithm

for certain types of GA encoding schemes and certain types of heuristics, determining such a weight vector is feasible whereas for certain types of heuristics it is not feasible. In general, if the priority heuristic used in the AugNN approach uses a priority list that is determined a priori using some problem parameter and remains constant throughout the heuristic, then determining a weight vector is feasible. If the priority list either (i) changes during a single-pass of the heuristic or (ii) is created on the fly, i.e. not a priori, then determining such a weight matrix will not be feasible. We provide an algorithm for generating a w vector given a permutation of Υ that generates the given solution η, assuming that the solution is obtainable through the permutation of Υ. While this assumption is valid for most problems because most heuristics are based on a prioritization of some aspect of the problem, it may not work in some cases. For example in parallel scheduling algorithms, no initial permutation of any problem parameter is used. Also, for the knapsack problem, certain heuristics, such as the Kochenberger heuristic (Kochenberger et al. 1974), rely on a recalculation of the problem-specific parameter after each item is added to the knapsack. For such heuristics which do not use any preprocessed ordered vector of problem parameters that remains constant or static throughout the heuristic algorithm, it will not be feasible to directly obtain a suitable weight vector. However, in such situations, heuristic approximations can often be used to overcome the problem. They do so by using predetermined vectors of problem parameters which do not require the limiting iterative recalculations as mentioned above. Figure 3 outlines the basic AugNN algorithm.

3.2 GA encoding

In genetic algorithms, a chromosome string represents a solution either explicitly or implicitly, depending on the nature of the problem and the decision variables. An example of a chromosome string where the solution is explicit, as discussed earlier, is the encoding for the knapsack problem. We call this Encoding Type 1. An example of a problem which has a chromosome string that represents the solution implicitly is the scheduling problem. In implicit coding, the solution can be generated using only the information contained in the chromosome string. For example, the string might represent an ordering of activity assignments. We call this Encoding Type 2. The details of these two types of encoding are described next.

3.2.1 Encoding Type 1

In the first type of encoding, the chromosome string looks like (0 1 0 0 1 0 1 1). The elements of this chromosome represent the binary $0/1$ values of the decision variable. This type of problem is exemplified through a knapsack problem.

Example of a single-dimension Knapsack problem:

$$\begin{aligned} \text{Max} \quad & \sum_{i\in N} c_i x_i \\ \text{s.t.} \quad & \sum_{i\in N} a_i x_i \le b, \\ & x_i = 1, \quad \text{if item } i \text{ is selected, } 0 \text{ otherwise.} \end{aligned}$$

Here, c_i represents the value of the items, a_i is the size (or cost) x_i is the $0/1$ decision variable, N is the number of items, b is the size capacity (or cost budget). Note that in a multi-dimension knapsack problem, there will be several constraints, one for each of the dimensions.

Let us consider a specific five-item ($N = 5$) problem as follows:

Item	c_i	a_i	c_i/a_i
1	8	4	2
2	6	2	3
3	2	2	1
4	10	4	2.5
5	8	2	4

Suppose the knapsack capacity (b) is 8. The goal of the problem is to determine which items to place in the knapsack in order to maximize the value (c_i) of the knapsack without violating the knapsack capacity. Suppose we are interested in a solution in which items 1, 3 and 5 are selected for the knapsack and use the heuristic "Highest Value-to-Size Ratio next" (HVSR). The GA encoding for this solution is (1 0 1 0 1). The question is—can we produce a weight vector w which in conjunction with the HVSR heuristic will give the GA solution of (1 0 1 0 1).

Algorithm for switching from GA encoding Type 1 to AugNN encoding

Let's consider the example problem described above. The GA encoding for the solution of selected items 1, 3 and 5 is (1 0 1 0 1). The AugNN encoding is a weight vector $w =$

$(w_1, w_2, w_3, w_4, w_5)$ in conjunction with heuristic HVSR. Given the chromosome string of (1 0 1 0 1) how do we determine w? Following is a general procedure:

```
Repeat until all pairs of genes agree with the target order
{   Consider the next pair of genes
    Check if the weighted parameter value and the heuristic agree with the target order
    If not, adjust weights as follows:
    {   Let $w_a$ and $w_b$ represent the weights corresponding to the first gene and the target gene
        For permutations based on decreasing order of parameter
          Set $w_a = w_b * (param_b/param_a) - \alpha/param_a$
        For permutations based on increasing order of parameter,
          Set $w_a = w_b * (param_b/param_a) + \alpha/param_a$
    }
}
```

In the above pseudocode, α is an arbitrary positive real number. The value of α must be chosen carefully. Although any positive value will force the switch, a value too high might adversely affect the relative gene position with respect to other genes and therefore could delay convergence. For practical purposes, α could range between 0.05 to 0.5. We experimented with some values and determined that a value of 0.1 works quite well.

We will illustrate this general procedure with the help of the example problem.

Step 1. We start with a w vector of $(1, 1, 1, 1, 1)$ so that $w * (c_i/a_i)$ of $(2.0, 3.0, 1.0, 2.5, 4.0)$.

Step 2. Consider the pair of genes 1 and 2. According to the weighted parameter values, item 2 is preferred over item 1 which disagrees with the target order. So, we want

$$w_1 * 2 > w_2 * 3, \quad \text{or} \quad w_1 > w_2 * 3/2$$

since $w_2 = 1$, $w_1 > 3/2$, assuming $\alpha = 0.1$, set $w_1 = 1.6$, $w = (1.6, 1, 1, 1, 1)$ and $w * (c_i/a_i) = (3.2, 3.0, 1.0, 2.5, 4.0)$.

Step 3. Pairs $(1, 2)$, $(1, 3)$, $(1, 4)$ and $(1, 5)$ agree. Pair $(2, 3)$ does not agree. Item 2 will be selected over item 3 whereas we need to select item 3. So $w_3 > w_2 * (3/1)$; assuming $\alpha = 0.1$, $w_3 = 3.1$. New $w = (1.6, 1, 3.1, 1, 1)$ and new $w * (c_i/a_i) = (3.2, 3.0, 3.1, 2.5, 4.0)$.

The vectors in each step are summarized in Table 1.

Table 1 Summary of w and $w * (c_i/a_i)$ vectors for all steps

Item	(c_i/a_i)	Step 1		Step 2		Step 3	
		w	$w * (c_i/a_i)$	w	$w * (c_i/a_i)$	w	$w * (c_i/a_i)$
1	2	1	2	1.6	3.2	1.6	3.2
2	3	1	3	1	3	1	3
3	1	1	1	1	1	3.1	3.1
4	2.5	1	2.5	1	2.5	1	2.5
5	4	1	4	1	4	1	4

3.2.2 GA encoding Type 2

In the second encoding type, the chromosome looks something like $(1, 4, 6, 5, 9, 2, 3, 8, 7)$. This type of encoding represents an ordering of items. For example in a TSP, this will represent the order in which the cities are visited. Alternatively, in a scheduling problem, this encoding may represent the order in which the activities will be scheduled assuming that they are resource and precedence feasible.

Algorithm for switching from GA encoding Type 2 to AugNN encoding

The general algorithm to determine the weight vector for this type of encoding is as follows:

```
Repeat until for each gene, the position based on $w * param$ is the same as the target
position
{   If a gene is out of place,
    Let $w_a$ and $w_b$ represent the weights corresponding to the out of place gene and
    the target position gene
    If ($w_a.param_a > w_b.param_b$ and $position_a > position_b$) then
        If (heuristic based on non-increasing order of param) then
            Set $w_a = w_b * (param_b/param_a) - \alpha/param_a$
        Else
            Do nothing
    Else
    if ($w_a.param_a > w_b.param_b$ and $position_a < position_b$) then
        If (heuristic is based on non-decreasing order of param) then
            Set $w_a = w_b * (param_b/param_a) + \alpha/param_a$
        Else
            Do nothing
    Else
    If ($w_a.param_a < w_b.param_b$ and $position_a > position_b$) then
        If (heuristic based on non-decreasing order of param) then
            Set $w_a = w_b * (param_b/param_a) - \alpha/param_a$
        Else
            Do nothing
    Else
    If ($w_a.param_a < w_b.param_b$ and $position_a < position_b$) then
        If (heuristic based on non-increasing order of param) then
            Set $w_a = w_b * (param_b/param_a) + \alpha/param_a$
        Else
            Do nothing
    End If
}
```

Example Let us say we are scheduling activities in a CPM/PERT chart and we are using the heuristic of "minimum latest finish time first" (Min LFT). Suppose there are eight activities and the vector F of their latest finish times is $(0, 5, 9, 6, 10, 14, 12, 18)$. Assume a vector of weights $w = (1, 1, 1, 1, 1, 1, 1, 1)$. Assume that GA produces a string of $(1, 2, 3, 5, 4, 7, 6, 8)$. The ranking based on $w * F$ is $(1, 2, 4, 3, 5, 7, 6, 8)$. We notice that the gene at position 3 (activity 4) is out of place. The target position is position 5 (activity 5).

So, set $w_3 = w_5 * (param5/param3) + 0.1$, or $w_3 = 1 * (10/9) + 0.1 = 1.21$.

So the new $w = (1.0, 1.0, 1.21, 1.0, 1.0, 1.0, 1.0, 1.0)$ and the new $w * F = (0, 5, 10.9, 6, 10, 14, 12, 18)$. The new ordering based on $w * F$ is $(1, 2, 3, 5, 4, 7, 6, 8)$ which is the target string.

Algorithm for switching from AugNN encoding to GA encoding

Switching encoding schemes from AugNN to GA is a simpler process. For both the encoding types, we will use the same examples as above.

Encoding type 1 Suppose the weight vector $w = (1.5, 2.2, 1.8, 3.1, 0.4)$

We simply calculate the $w * (c_i/a_i)$ vector or (3.0, 6.6, 1.8, 3.25, 1.6) based on the HVSR heuristic, items 2 and 4 are selected so the chromosome string is (0 1 0 1 0).

The general algorithm is:

Compute the weighted parameter vector
Apply the chosen heuristic to determine the solution
Decode the solution to create the corresponding chromosome string.

Encoding type 2 Suppose the weight vector is $(1.5, 2.5, 0.3, 1.8, 3.2, 2.2, 2.8, 3)$ and F is $(0, 5, 9, 6, 10, 14, 12, 18)$. $w * F$ is $(0.0, 12.5, 2.7, 10.8, 32.0, 30.8, 33.6, 54)$ which gives a ranking of $(1, 3, 4, 2, 6, 5, 7, 8)$ which is also the GA encoding.

The general algorithm is:

Compute weighted parameter vector
Order the elements of the vector based on the chosen heuristic.

4 Empirical evidence

In order to test the effectiveness of the proposed approach, we focused on the multidimensional knapsack problem. Although the one-dimensional knapsack problem is not NP-Hard, the multidimensional problem is. This problem has received a lot of attention in the literature (Kochenberger et al. 1974; Senju and Toyoda 1968; Gavish and Pirkul 1985). Chu and Beasley (1998) developed a large set of benchmark problems and also developed an effective GA procedure to solve those problems. We coded (i) the GA search algorithm as described in Chu and Beasley (1998), (ii) an AugNN search algorithm (described in Fig. 3) and (iii) a NeuroGenetic search algorithm as described in this paper (Fig. 2). Our code was written in Visual Basic 6.0, running on Pentium IV machine with 1 GB memory. Chu and Beasley's data set is made up of nine subsets of problems of sizes 100×5, 100×10, 100×30, 250×5, 250×10, 250×30, 500×5, 500×10 and 500×30. A 500×5 problem has 500 items and 5 constraints. Each subset consists of 30 problems. On each subset, we ran two sets of experiments. In set one, a run was terminated once 5,000 unique solutions were found. In set two the run continued until 10,000 unique solutions were found. In each experiment, for the NeuroGenetic approach, we devoted 95 percent of the search iterations to the GA with the remaining 5 percent to AugNN. There were 5 interleavings. In each interleaving, 5 GA solutions were randomly selected from the current population and fed independently to the AugNN. Using each of these solutions as a starting point, the AugNN ran until it located 1% of solutions. If one of these solutions offered improvement over those located thus far, that solution was then added into the next GA generation.

Table 2 Results of AugNN, GA and NeuroGenetic on 270 benchmark problems, 5,000 unique solutions

Problem set	Percent Gap from LP-relaxed solution			Average time taken per problem in seconds		
	AugNN	GA	NeuroGenetic	AugNN	GA	NeuroGenetic
100 × 5	0.96	0.71	0.68	10.39	1.82	2.31
100 × 10	1.75	1.20	1.16	17.86	2.25	3.25
100 × 30	2.94	2.31	2.13	50.82	3.95	6.32
250 × 5	0.36	0.27	0.24	27.50	3.26	4.54
250 × 10	0.78	0.59	0.52	43.89	4.16	6.65
250 × 30	1.65	1.19	1.11	132.84	7.19	12.31
500 × 5	0.19	0.18	0.14	88.43	5.44	9.96
500 × 10	0.34	0.36	0.33	151.62	6.65	13.98
500 × 30	0.97	0.86	0.80	453.01	12.92	34.71
Average	1.10	0.85	0.80	108.48	5.29	10.45

Table 3 Results of AugNN, GA and NeuroGenetic on 270 benchmark problems, 10,000 unique solutions

Problem set	Percent Gap from LP-relaxed solution			Average time taken per problem in seconds		
	AugNN	GA	NeuroGenetic	AugNN	GA	NeuroGenetic
100 × 5	0.93	0.66	0.61	25.25	4.76	13.31
100 × 10	1.70	1.15	1.03	33.86	4.86	13.71
100 × 30	2.92	1.92	1.77	105.32	8.46	15.31
250 × 5	0.34	0.23	0.18	58.36	8.06	30.91
250 × 10	0.72	0.42	0.36	85.48	9.10	32.06
250 × 30	1.61	0.84	0.73	230.24	16.29	35.46
500 × 5	0.18	0.09	0.07	170.23	12.05	57.55
500 × 10	0.31	0.20	0.17	311.12	14.56	61.42
500 × 30	0.95	0.42	0.39	893.81	27.43	69.36
Average	1.08	0.66	0.59	212.63	11.73	36.56

Tables 2 and 3 summarize the results for the two sets of experiments. We observe that the AugNN was the slowest of the three approaches. The GA was the quickest technique and the NeuroGenetic approach was a little bit slower than GA but considerably faster than AugNN. This is not surprising given that 95% of NG iterations were GA iterations. In the knapsack literature, the quality of the solution is commonly measured as the percentage gap from the LP-relaxed solution. A lower gap is a sign of a better solution. For the first set of experiments (5,000 unique solutions) the AugNN, GA and NG approaches achieved average gaps of 1.10, 0.85 and 0.80 percent respectively. The average time taken per problem by the NG approach was 10.45 seconds compared to GA's 5.29 seconds. For each of the three subsets of problems, the NeuroGenetic approach showed improvement and dominated both the AugNN and the GA approaches. For the second set of experiments (10,000 unique solutions), AugNN averaged a gap of 1.08 percent, GA reduced the gap from 0.85 to 0.66 while NG reduced the gap from 0.80 to 0.59. Again the NG approach offered improved

Table 4 Test of significance of positive difference between NG and GA solutions for 5,000 unique solutions

Problem set	Wilcoxon-Signed-Ranks test z-value	p-value
100×5	-0.339	0.367
100×10	-1.008	0.157
100×30	-3.67	0.000121[a]
250×5	-2.27	0.012[b]
250×10	-4.77	0.00000091[a]
250×30	-4.73	0.0000011[a]
500×5	-4.56	0.000025[a]
500×10	-4.78	0.00000087[a]
500×30	-4.78	0.00000087[a]
Overall	-12.0756	7.1×10^{-34}

[a]significant at 1%

[b]significant at 5%

Table 5 Test of significance of positive difference between NG and GA 10,000 unique solutions

Problem set	Wilcoxon-Signed-Ranks test z-value	p-value
100×5	-3.98	0.0000345[a]
100×10	-4.22	0.0000124[a]
100×30	-3.70	0.000107[a]
250×5	-4.30	0.000008597[a]
250×10	-4.45	0.00000423[a]
250×30	-4.75	0.00000101[a]
500×5	-4.43	0.00000466[a]
500×10	-3.98	0.0000345[a]
500×30	-4.20	0.0000136[a]
Overall	-12.89	2.7×10^{-38}

[a]significant at 1%

results over the other techniques while requiring some extra processing time—36.56 seconds per problem vs. 11.73 seconds per problem for the GA.

We also performed statistical tests of significance to test if the NeuroGenetic approach provided better solutions than GA. We performed a one-tailed Wilcoxon Signed-Ranks test (Demsar 2006) on each of the nine sets of 30 problems and also on the entire 270 problems. Tables 4 and 5 provide the Wilcoxon Signed-Ranks test z-values and their corresponding p-values for each dataset for 5,000 and 10,000 unique solutions respectively. For the results of 5,000 unique solutions, on six of nine datasets, NeuroGenetic performed better than GA at the 1% significance level, on one dataset it performed better at the 5% significance level and on two datasets, the improvement was not significant at 10% level. For 10,000 unique solutions, on all nine datasets the NeuroGenetic approach performed significantly better than GA at the 1% level. For all 270 problems, the significance level was extremely high—the p-value being 7.1×10^{-34} for the 5,000 solutions and 2.73×10^{-38} for 10,000 solutions.

For many optimization problems, the best solutions are given by Genetic Algorithms. For the knapsack problem for example, the results using Genetic Algorithms (Chu and Beasley

1998) dominate all other results. For the resource-constrained project scheduling problems, Kolisch and Hartmann (2006) show that Genetic Algorithms dominate all other metaheuristic approaches. In this paper, we show that in certain situations a combination of genetic algorithms with AugNNs provides an even more powerful solution alternative than GAs alone.

5 Summary and conclusions

In an attempt to solve various discrete optimization problems that appear in a variety of disciplines, heuristic and metaheuristic approaches are commonly employed since exact solutions for larger problems are hard to find due to their NP-hard nature. One major challenge in using metaheuristics is that they often struggle in their development of a good balance between global search and local search. In this paper, we propose a hybrid approach called the NeuroGenetic approach in which the GA approach and the AugNN approach are interleaved, i.e., two approaches alternate the iterations and try to improve upon the previous best iteration. Since the GA and the AugNN approaches are, in general, strong in global search and local search respectively, the interleaved approach grants the advantages of both global and local search mechanisms hopefully helping to overcome this balancing problem that is seen with most other techniques. The transition from one approach to another, however, is not trivial, as it requires a switching (translation) of the encoding schemes used by the two approaches. For the interleaved approach to be feasible, this transition must be possible in both directions. In this paper, we describe the encoding schemes of both the GA and the AugNN approaches and explain with the help of examples, proposed algorithms for the translation of two types of encoding schemes. We also discuss some limitations of the NG technique, in terms of the potential difficulty of being able to switch from GA encoding to AugNN encoding, if the heuristic being used with the AugNN procedure does not rely on a static ordered vector of some problem parameter.

The proposed hybrid approach was tested on 270 benchmark multidimensional knapsack problems of size ranging from 100×5 to 500×30. Results on two different sets of experiments demonstrate that the NeuroGenetic approach is able to improve upon the solution quality of either the GA or the AugNN alone. And in doing so, the NeuroGenetic approach requires some extra processing time. Since this is a new metaheuristic, there are many opportunities for future work. An obvious opportunity is to see if the proposed approach is effective on other types of combinatorial optimization problems such as scheduling, the traveling salesman problem, the bin-packing problem etc. Also, in this paper, we discussed only two encoding schemes. Future research may focus on other types of encoding schemes or on discovering more effective weight-determination strategies.

References

Agarwal, A. (2009). Theoretical insights into the augmented-neural-network approach for combinatorial optimization. *Annals of Operations Research*, *168*, 101–117.

Agarwal, A., Jacob, V., & Pirkul, H. (2003). Augmented neural networks for task scheduling. *European Journal of Operational Research*, *151*(3), 481–502.

Agarwal, A., Jacob, V., & Pirkul, H. (2006). An improved augmented neural-networks approach for scheduling problems. *INFORMS Journal on Computing*, *18*(1), 119–128.

Birbil, S. I., & Fang, S. C. (2003). An electromagnetism-like mechanism for global optimization. *Journal of Global Optimization*, *25*(3), 263–282.

Cho, J. H., & Kim, Y. D. (1997). A simulated annealing algorithm for resource-constrained project scheduling problems. *Journal of the Operational Research Society*, *48*, 736–744.

Chu, P. C., & Beasley, J. (1998). A genetic algorithm for the multidimensional Knapsack problem. *Journal of Heuristics*, *4*, 63–86.

Colak, S., & Agarwal, A. (2005). Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. *Naval Research Logistics*, *52*, 631–644.

Colak, S., Agarwal, A., & Erenguc, S. S. (2006). Resource-constrained project scheduling problem: a hybrid neural approach. In J. Weglarz, J. Jozefowska (Eds.), *Perspectives in modern project scheduling* (pp. 297–318).

Debels, D., De Reyck, B., Leus, R., & Vanhoucke, M. (2006). A hybrid scatter search electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, *169*(2), 638–653.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*(1), 1–30.

Dorigo, M., Caro, G. D., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, *5*(2), 137–172.

Dueck, G. (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, *104*, 86–92.

Gavish, B., & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero-one Knapsack problems to optimality. *Mathematical Programming*, *31*, 78–105.

Gen, M., & Cheng, R. (1997). *Genetic algorithms & engineering design*. New York: Wiley.

Glover, F. (1989). Tabu search—part I. *ORSA Journal on Computing*, *1*(3), 190–206.

Glover, F., Kelly, J., & Laguna, M. (1995). Genetic algorithms and tabu search: hybrids for optimization. *Computers and Operations Research*, *22*(1), 111–134.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Dordrecht: Kluwer Academic.

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, *52*, 141–152.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Kochenberger, G., McCarl, B., & Wyman, F. (1974). A heuristic for general integer programming. *Decision Sciences*, *5*, 36–44.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: an update. *European Journal of Operational Research*, *174*(1), 23–37.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, *21*, 498–516.

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, *6*, 333–346.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, *41*, 421–451.

Senju, S., & Toyoda, T. (1968). An approach to linear programming with 0–1 variables. *Management Science*, *15*, B196–B207.

Tormos, P., & Lova, A. (2001). A competitive heuristic solution technique for resource constrained project scheduling. *Annals of Operations Research*, *102*, 65–81.