Ann Oper Res (2009) 168: 101–117
DOI 10.1007/s10479-008-0364-8

Theoretical insights into the augmented-neural-network approach for combinatorial optimization

Anurag Agarwal

Published online: 12 June 2008

Abstract The augmented-neural-network (AugNN) approach has been applied lately to some NP-Hard combinatorial problems, such as task scheduling, open-shop scheduling and resource-constraint project scheduling. In this approach the problem of search in the solution-space is transformed to a search in a weight-matrix space, much like in a neural-network approach. Some weight adjustment strategies are then used to converge to a good set of weights for a locally optimal solution. While empirical results have demonstrated the effectiveness of the AugNN approach vis-à-vis a few other metaheuristics, little theoretical insights exist which justify this approach and explain the effectiveness thereof. This paper provides some theoretical insights and justification for the AugNN approach through some basic theorems and also describes the algorithm and the formulation with the help of examples.

Keywords Combinatorial optimization · Neural networks · Metaheuristics · Local search

1 Introduction

A number of intelligent metaheuristic approaches have emerged lately for solving various NP-Hard combinatorial problems. In just the past few years, iterated greedy algorithm (Ruiz and Stutzle 2006), scatter search (Martí et al. 2006; Pacheco 2005), electromagnetism-like algorithms (Birbil and Fang 2003; Debels et al. 2006), Augmented Neural Networks (Agarwal et al. 2003), GRASP (Pitsoulis and Resende 2001) and RAMP (Rego 2005) have been proposed and applied to various combinatorial optimization problems. Other metaheuristics such as variable neighborhood search (Ribeiro and Souza 2002), memetic algorithms (Cheng and Gen 1997), ant colony optimization (Dorigo et al. 1999), iterated local search (Stutzle 1998), were proposed in the nineties. Tabu search (Glover 1989), genetic algorithms

A. Agarwal (✉)
Department of Information Systems and Operations Management,
Warrington College of Business Administration, University of Florida,
Gainesville, FL 32611-7169, USA
e-mail: aagarwal@ufl.edu

(Goldberg 1989) and simulated annealing (Kirkpatrick et al. 1983) have established themselves as metaheuristics for some time now. Refinement strategies such as path-relinking have also been explored within the established metaheuristic frameworks (Díaz and Fernández 2006; Reeves and Yamada 1998).

At the most fundamental level, there are significant commonalities amongst all metaheuristics. They all, essentially, attempt to iteratively improve upon a set of feasible solutions using appropriate local moves or perturbation. The nature of moves or perturbation varies from one technique to another. For a given metaheuristic, we find in the literature, empirical evidence of their effectiveness in solving certain types of problems. Little theoretical insights, however, are known for the effectiveness of metaheuristics (Ruiz and Stutzle 2006; Kolisch and Hartmann 2006). In many cases, significant randomness is built in the way the perturbation takes place. If perturbations are completely random then a metaheuristic is akin to random search or a black-box approach. Wolpert and Macready (1997) argue that no black-box algorithm can dominate another over the set of all problems for a given class of problems and that the performance of any black-box algorithm is no better than a random search. In order for an algorithm to dominate another, some mechanism to mitigate randomness must, therefore, be built into the search mechanism. In tabu search, for example, randomness is alleviated through the use of memory lists. Attempts have also been made to incorporate domain/problem specific knowledge as a means to mitigate randomness in search. For example, in genetic algorithms, repair operators that utilize domain-specific knowledge to impose feasibility are designed to bias the search towards good solutions instead of random solutions (Chu and Beasley 1998). The use of heuristics to find initial solutions is also considered a step towards applying domain and problem-specific knowledge, thus reducing randomness.

This paper focuses on one such metaheuristic—augmented-neural-networks (AugNN), in which incorporating domain-specific knowledge is integral to the approach. The approach is a hybrid of neural-networks and the heuristic approach. In this approach, a weight matrix of appropriate size is assumed and the problem of search in the solution space is transformed to a problem of search for the best weight matrix in the weight space. The effectiveness of this approach has been empirically demonstrated in recent years on the task-scheduling problem (Agarwal et al. 2006, 2003), the open-shop scheduling problem (Colak and Agarwal 2005) and the resource-constrained project scheduling problem (Colak et al. 2006). However, little theoretical justification for the approach has been provided. In this paper we provide some theoretical insights into the AugNN approach. We first explain the approach with the help of two generic examples and then discuss the theoretical issues and develop theorems and their corollaries to provide the theoretical justification for the AugNN approach as a metaheuristic.

2 The AugNN approach explained

As pointed out before, the AugNN approach is a hybrid of the neural-network approach and the heuristic approach. It is well known that heuristics usually generate reasonably good approximate solutions fast to most NP-Hard problems. In the AugNN approach, a heuristic solution is used as a proxy for a good neighborhood of solutions. Iterative local search around the heuristic solution attempts to find a locally optimal point. In this approach, a given problem is framed as a neural network of non-linear functions (input, transfer and output) in such a way that (i) the constraints of the problem and (ii) a heuristic are embedded in this network of functions. Domain and problem specific knowledge are thus embedded

in the network. These non-linear functions include weights just like in a traditional neural network. A problem's parameters can be fed to the network as input and the network of functions produces as output the solution and the objective function value of the problem. We add memory to each processing element that holds the problem parameters. An example of an AugNN representation as a neural network is described in Sect. 5. The first pass or iteration produces the heuristic solution, which of course depends on the chosen heuristic, while subsequent iterations produce new solutions. New solutions are a result of perturbation of weights. The weight-adjustment strategy is designed so as to bias the search towards optimal/near-optimal local solution. In this and the next two sections, we explain and discuss the issues related to the role of weights in determining the solutions.

When applying metaheuristics, in general, one or more initial feasible solutions are first obtained either randomly or with the help of some good heuristics; improvements to these initial solutions are then sought iteratively through some perturbation mechanism. For example, in genetic algorithms, the initial population of chromosomes represents some initial solutions. Subsequent populations contain solutions obtained by perturbing the initial solutions using various crossover, mutation and repair operators. Some crossover schemes tend to perform better than others. Those that can capitalize on some domain or problem-specific knowledge would tend to outperform those more random in nature.

In the AugNN approach, the initial solution, as pointed out before, is found using a good heuristic. A heuristic generally uses some problem-specific parameter, hereafter heuristic parameter, such as the constraint coefficients or the objective function coefficients or some combination thereof, to prioritize the decision variables or to prioritize something that helps determine the decision variables. The perturbation mechanism, in the AugNN approach, utilizes a weight matrix. The heuristic parameter is weighted with the weight matrix and the same heuristic is then applied to the weighted heuristic parameter to generate a new solution. The weight matrix is modified at the end of each iteration and the heuristic applied again, to obtain new solutions in each iteration.

For example, consider the following one-dimensional knapsack problem:

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

where N represents the set of all items and $|N| = n$; c_i is the value of item i; a_i is the size of item i and b is the capacity of the knapsack. Suppose we use the heuristic HVSR in which—the item with the "Highest Value-to-Size-Ratio (c_i/a_i)" is placed next in the knapsack if it can fit in the knapsack without violating the capacity constraint. In the AugNN approach, we introduce a weight vector w. For this example, since the heuristic parameter is a vector (c_i/a_i) of n elements, w will also be a vector of n elements. Had the heuristic parameter been a matrix, we would need a weight matrix. Initially all elements of w are assumed to be 1, and $w_i.(c_i/a_i)$ is computed and the heuristic HVSR is applied once, giving us an initial solution. Let us call the list of items sorted in non-increasing order based on the ratio $w_i.(c_i/a_i)$, a permutation of items. According to the HVSR heuristic, items from the top of this permutation are placed in the knapsack until the knapsack is full. Clearly, if this permutation had been different, especially at the top, the solution would likely be different. See the numerical example in the Appendix. To find the next solution, we modify each of the weight vector elements as follows: If Rand() < 0.5 then $w_i = w_i + (\text{Rand}.()\alpha.e.w_i)$,

else $w_i = w_i - (\text{Rand}().\alpha.e.w_i)$, where α is the search coefficient, e is the error term, i.e. the deviation of the current solution to a lower (or upper bound) solution for a minimization (maximization) problem. We then update the elements of weighted parameter vector $w_i.(c_i/a_i)$ and apply the same HVRS heuristic again. If the new permutation of items (based on updated $w_i.(c_i/a_i)$) is different from the one based on the previous $w_i.(c_i/a_i)$, then a different solution is likely to be obtained. Weights are modified again and the same heuristic applied using updated weighted ratio $w_i.(c_i/a_i)$ producing yet another solution. Likewise, several iterations can be executed and the best solution saved.

In this approach, as can be observed from the above example, the search for the best solution has been transformed into the search for the best weight vector, for a particular heuristic. Suppose that we had used a different heuristic called HV in which the item with the "Highest Value" (c_i) is placed next as long as the knapsack capacity constraint is not violated. For the initial solution, this heuristic would give a different permutation of items, compared to the HVSR heuristic and consequently a different initial solution. Subsequent permutations will be obtained using the sorting order based on updated $w_i.c_i$ vector. The AugNN approach will then try to find the best weight vector, which in conjunction with the HV heuristic, gives the best solution. Note that a perturbation in the weight vector gives rise to a perturbation in the permutation of items which in turn produces a new solution. If the perturbation (in the weights, and consequently on the permutation) is small, the new solution can be considered a solution in the local neighborhood of the old solution. The weight vector, thus provides a mechanism for perturbation. The advantage of using the weight vector (in conjunction with a heuristic) as a means of perturbation is that using this mechanism, a feasible solution is always guaranteed because heuristics are designed to always generate feasible solutions.

Let us consider a different problem, the Traveling Salesman Problem (TSP):

$$
\begin{aligned}
\text{Min} \quad & \sum_{i\in N}\sum_{j\in N} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{i\in N} x_{ij} = 1 \\
& \sum_{j\in N} x_{ij} = 1 \\
& \sum_{i\in K}\sum_{j\in K} x_{ij} \le |K| - 1 \quad \text{for all } K \subset N
\end{aligned}
$$

where N is the set of all city nodes; $|N| = n$; c_{ij} is the cost (or distance) to travel from node i to node j; x_{ij} represents the edge from node i to node j; x_{ij} is 1 if the edge is on the tour and 0 otherwise. Many heuristics have been proposed in the literature for finding approximate solution to this problem. For purposes of explaining the AugNN approach, let us consider a simple heuristic called NNN, or "Nearest-Neighbor Next" in which the closest unvisited city is visited next till the tour is complete. To apply the AugNN approach, we introduce a weight matrix w. Since the heuristic parameter is the matrix c_{ij} of size $n \times n$, the weight matrix will also be of size $n \times n$. Initially, w is set as an identity matrix and $w_{ij}.c_{ij}$ is computed. The heuristic NNN is applied using $w_{ij}.c_{ij}$ to obtain the initial solution. We can call a TSP solution a permutation of city nodes. Subsequent solutions are obtained by perturbing the w matrix and updating the weighted distances between nodes ($w_{ij}.c_{ij}$) and applying the same NNN heuristic. The application of the NNN heuristic will presumably give a different permutation of city nodes each time the weight matrix is perturbed. Once again, the problem of finding the best tour has been transformed to the problem of searching for the best set of

weights which in conjunction with a given heuristic will attempt to produce the optimal solution.

It should be observed that heuristics, in general, use some problem-specific parameter(s) to establish a priority order (or permutation) of some aspect of the problem, which guides the solution procedure. For example, in the knapsack problem, the problem-specific parameters were c_i or (c_i/a_i) and the items themselves were prioritized. In the TSP, the NNN heuristic gave us a permutation of city nodes using the problem-specific parameter of c_{ij}. In the bin-packing problem, for example, items to be added to bins are prioritized based on the size of the item, therefore the item size becomes the problem-specific parameter and items themselves are permuted.

We will now try to generalize the AugNN procedure. Suppose for a given problem and for a given heuristic σ, a certain problem-specific parameter Υ is used to establish the priority order of whatever it is that guides the solution procedure. For example, for the knapsack problem if we are using the HV heuristic, σ will be the HV heuristic, Υ will be the vector of elements c_i and permutation is of the items themselves. For the HVSR heuristic, σ will be the HVSR heuristic, Υ will be the vector of elements c_i/a_i, and again, permutation will be of the items. For the TSP, for the NNN heuristic, σ is NNN, Υ is the matrix of elements c_{ij} and permutation is of city nodes. The general AugNN approach is as follows:

1. Choose an appropriate heuristic σ and identify the heuristic parameter(s) Υ.
2. Establish an identity weight matrix of size commensurate with Υ.
3. Compute weighted parameters $(w.\Upsilon)$.
4. *Repeat*
   4.1 Generate a solution using the heuristic σ using the weighted parameters $(w.\Upsilon)$.
   4.2 Capture the best solution so far.
   4.3 Modify the weights using a weight adjustment strategy.
   4.4 Update weighted input parameters $w.\Upsilon$.
   Until termination criteria met.
5. Display the best solution so far.

We raise the following theoretical questions—Is this approach of transforming the problem of search in the solution space to a problem of search in the weight space justified or not? If it is justified, what search strategies can be employed, what is the computational complexity of the approach, what advantages, if any, does this approach have over other approaches which focus on search in the solution space? In the next section we provide some theoretical justification for this approach and later discuss the subsequent issues.

3 Theoretical justification

In this section we first describe the notation and discuss the types of problems and types of heuristics that we are interested in addressing. We then develop theorems and corollaries and make observations which will justify this approach and help us design the best search strategy.

3.1 Notation and groundwork

In this paper, we are interested in integer-linear programming problems where the search space for the decision variables is combinatorial in nature. Examples include the traveling salesman problem, the knapsack problem, the bin-packing problem, various scheduling problems etc. An optimization problem ρ of this type can be expressed as $\rho : \Theta = f(\chi)$

s.t. ψ where Θ is the objective function, a function of χ; ψ represent a set of constraints. Non-linear or non-integer linear programming problems are not considered. Assume that the optimal solution to this problem is Θ_{opt}.

Let a deterministic heuristic σ solve ρ to an approximate solution $\Theta_{\sigma.}$. Here we are interested in heuristics which make an attempt to find the best possible solution. Heuristics which intentionally terminate before any constraint is violated are not considered. We are also assuming that these heuristics work by prioritizing some aspect of the problem using some problem-specific parameter(s).

Let n be the length of the problem. For example if ρ is the knapsack problem n would represent the number of items being considered for the knapsack, χ would be the vector of values of items. The constraint set ψ would be the various capacity constraints. For the TSP, n would be the number of city nodes, χ would be the matrix of distance (or cost) between the city nodes and ψ would be that all nodes must be visited at least once. The heuristic σ works with some problem-specific parameter(s) Υ. Using Υ, σ produces a permutation $\zeta(\Upsilon_\sigma)$ of some aspect of the problem. For example, for the knapsack problem, for the HV heuristic Υ is the c_i vector and $\zeta(\Upsilon_\sigma)$ is a permutation of items with non-increasing values of c_i.

We now propose and discuss a theorem which establishes the relationship between the weight matrix w, the optimal solution Θ_{opt} of the problem ρ and the heuristic σ.

3.2 Theorems

For the purpose of Theorem 1, without loss of generality, we will assume that Υ and w are matrices of single row (or column)—in other words, vectors. This assumption is made strictly for the ease of exposition of the proof and in no way affects the generality of the theorem when Υ and w are matrices of higher dimensions. We also assume that all elements of Υ are positive.

Theorem 1 *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, there exists a weight vector $w \in \Re^m$ such that the application of the heuristic σ using the weighted parameter vector $(w.\Upsilon)$ can produce the optimal solution Θ_{opt} of ρ.*

Proof of Theorem 1 Note that the application of the heuristic σ generates a permutation (or priority order) of some aspect of the problem which we will call the permutation of interest, which guides the solution. This ordering is based on some problem-specific parameter. We are assuming that one such permutation of interest results in the optimal solution. This assumption is valid because if one knew the optimal solution, a permutation corresponding to that solution can be generated. For example, if the optimal solution to a given knapsack problem is known then one permutation of interest that gives the optimal solution will be the list of all items in the knapsack (in any order) followed by the list of all items not in the knapsack (in any order).

Let $\zeta(\Upsilon_\sigma)$ represent the permutation of interest resulting from the use of the heuristic σ based on the problem-specific parameter Υ. Let the permutation $\zeta(\Upsilon_{opt})$ be the one that leads to a potentially optimal solution Θ_{opt} to the problem ρ. Let $\zeta(\Upsilon_{arb})$ be any arbitrary permutation of interest. Let $\zeta(\Upsilon_\sigma^w)$ be the permutation of interest generated using the heuristic σ using weighted parameter. If it can be shown that there exists a weight vector $w \in \Re^m$, such that the permutation of interest $\zeta(\Upsilon_\sigma^w)$ is identical to any arbitrary permutation $\zeta(\Upsilon_{arb})$

of ρ, then it can be argued that $\zeta(\Upsilon_{arb})$ could be $\zeta(\Upsilon_{opt})$ and that therefore a weight vector $w \in \Re^m$ exists for generating $\zeta(\Upsilon_{opt})$, thus completing the proof.

Let $\zeta(\Upsilon_{arb}) = (\tau^1_{arb}, \tau^2_{arb}, \ldots, \tau^m_{arb})$.

Let the heuristic parameter values corresponding to $(\tau^1_{arb}, \tau^2_{arb}, \ldots, \tau^m_{arb})$ be $(x^1_{arb}, x^2_{arb}, \ldots, x^m_{arb})$.

Assume that the heuristic σ prioritizes items in non-increasing order of parameter values. So, if there is a $w = (w^1, w^2, \ldots, w^m)$ such that $w^1.x^1_{arb} > w^2.x^2_{arb} > \cdots > w^m.x^m_{arb}$, then σ will prioritize the items in the order $(\tau^1_{arb}, \tau^2_{arb}, \ldots, \tau^m_{arb})$. We show that a such weight vector w can be constructed as follow:

Let $w^m = 1$ and let $w^i = \frac{w^{i+1}.x^{i+1}_{arb}}{x^i_{arb}} + 1$ for $i = m-1, \ldots, 1$.

Such a weight vector will ensure that $w^1.x^1_{arb} > w^2.x^2_{arb} > \cdots > w^m.x^m_{arb}$.

This completes the proof. □

Note that, even though the theorem was stated and proven in terms of a vector of Υ and w, that if Υ and w were matrices, the spirit of the theorem would still hold. This theorem, thus, provides the justification for searching for the best set of weights in the weight space, because all potentially optimal points are reachable through this search mechanism. In the next theorem we prove that there exist infinite weight vectors (or matrices) that can yield the permutation that leads to the potentially optimal solution.

Theorem 2 *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, 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 the optimal solution Θ_{opt} of ρ and that $|\Omega| = \infty$.*

Proof of Theorem 2 In the proof of Theorem 1, substitute $w^i = \frac{w^{i+1}.x^{i+1}_{arb}}{x^i_{arb}} + 1$ with $w^i = \frac{w^{i+1}.x^{i+1}_{arb}}{x^i_{arb}} + a$, where a is an arbitrary positive number. Since there can be an infinite number of possible values of a, there are an infinite number of potential weight vectors w that, in conjunction with a heuristic will generate an optimal solution. This completes the proof. □

It was important to first establish the existence of at least one such weight vector to justify searching for the ideal weight vector as a means of finding the optimal solution to the original problem. Theorem 2 suggests that the search is not just limited to one optimal weight vector but one of an infinite number of such vectors that will give us the optimal solution to the problem. Clearly, the likelihood of searching for one from an infinite number of such vectors is far better than of searching one of only one such vector. Theorem 2, thus, provides the encouragement to work with the weight search.

Corollary of Theorem 2 *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 $|\Omega| = \infty$.*

In plain English, this corollary states that for any solution to the original problem, there exist infinite number of weight-vectors that can produce the permutation necessary for ob-

taining that solution. The only difference between this corollary and Theorem 2 is that instead of proving that there exists a set of weights that gives the optimal solution, that there exists a set of weights for any solution of the problem. The proof for this corollary is actually contained in the proof for Theorem 1. Since in Theorem 1, we show that a weight vector exists such that $\zeta(\Upsilon_\sigma^w) = \zeta(\Upsilon_{arb})$. Since $\zeta(\Upsilon_{arb})$ is essentially any solution η, the corollary is proved.

4 Convergence and other issues

Having established theoretically that there exists a weight vector that in conjunction with a given heuristic can generate an optimal solution, the next question is can the iterative mechanism of the AugNN approach converge to such a weight vector. Given that there are an infinite number of weight vectors for every possible solution, including the optimal solutions, guaranteeing the search for the optimal weight vector in finite number of iterations would not be possible. The AugNN approach is a local search approach, which attempts to find an improved solution in the local neighborhood which is anchored by a given heuristic. Given that an optimal weight vector cannot theoretically be found in finite number of iterations, the next best thing to do is to suggest strategies on how to bias the search quickly towards a good weight vector. We now discuss some search strategies.

4.1 Search strategies

In Sect. 2, we had discussed a specific weight modification strategy—if Rand() < 0.5 then $w_i = w_i + (\text{Rand}().\alpha.w_i)$ else $w_i = w_i - (\text{Rand}().\alpha.w_i)$; where Rand() is a random number between 0 and 1 and α is a search coefficient. By controlling the magnitude of α, we can control the extent of weight modification. Several variations to this specific strategy are possible. We will first give a general strategy here—assuming k is the iteration number and e_k the error in iteration k, defined as the difference between the obtained solution value in iteration k and a lower bound estimate (for minimization problems) for the problem, we note that:

$$w_{k+1} = f(w_k, \alpha, e_k) \tag{1}$$

$$\Delta w = w_{k+1} - w_k. \tag{2}$$

$$\Delta w \propto \alpha \tag{3}$$

$$\Delta w \propto e \tag{4}$$

Equation (3) suggest that a higher value of α provides a higher perturbation. We develop a series of search strategies which revolve around the search coefficient α.

Strategy 1 (Annealing schedule) *A higher α during the initial stages of search and a gradually reducing α at later stages of search biases the search towards global optimal.*

While a small α is good for searching a local neighborhood, a large α helps in escaping a local valley in the search space, thus allowing the search for the global optimal. The underlying idea of larger initial moves followed by smaller moves is not new as it has been used in simulated annealing technique in the past (Kirkpatrick et al. 1983). With respect to α, the

annealing idea has been used in weight modification techniques in traditional neural networks as well (Mehrotra et al. 1997). We reiterate that such an annealing schedule is very beneficial in AugNN search for biasing the search towards the global optimal.

How small can the small value of α be and how high can the high value of α be? If it is too small, the weights may not be affected enough to make a dent in the permutation $\zeta(\Upsilon_\sigma^w)$. To better understand the issue of magnitude of α, we give a corollary of Theorem 2 and then make two observations, which will guide us in selecting the best magnitudes of α.

Existence of infinite number of weight vectors for each solution (Corollary) poses another problem. How to avoid the weight vectors that will produce the same solution again and again, for otherwise, we will be wasting a lot of CPU cycles? From this corollary, follows the following observation:

Observation 1 *If the vector difference between two weight vectors is infinitesimally small, the chosen heuristic σ in conjunction with these two weight vectors will generate the same permutation of interest, thereby giving the same solution to the problem ρ.*

The Corollary and Observation 1 have several implications for selecting the appropriate value of α. Observation 1 suggests that we should be careful in setting the rate of change of w so that we are not generating the same solution too frequently because it is possible for many sets of w to produce the same solution. It is difficult to make any theoretical suggestions about the minimum magnitude of α because a lot depends on the specific problem and the relative magnitudes of the heuristic parameters. Some experimentation is necessary to fix the lower bound for α to ensure that duplicate solutions are not generated too often. So the next question is how high of a value can α assume. Observation 2 will help us in this regard.

Observation 2 *The value of α should be such that the magnitude of Δw is of a lower order of magnitude than the values of the elements of w.*

If the order of magnitude of the value of Δw is equal or higher than the value of the elements of w, the weighted parameter value would be too distorted and the next iterative solution would not be in the same neighborhood as the current iteration's solution. In theory, of course, any order of magnitude of Δw could lead to the optimal value of the original problem, but it is less likely for large Δw to converge to any local optimal solution.

Strategy 2 (Dynamic annealing schedule) *Go through several cycles of the annealing schedule of Strategy 1.*

Cycles of high α followed by gradually decreasing α are repeated a number of times to bias the search towards a global optimal. The timing as to when to repeat a cycle can be triggered dynamically by watching the behavior of obtained solutions in recent iterations. If the obtained solution is not varying, it is time to start a new cycle to explore a new neighborhood. This dynamism includes backtracking of weights, which will be explained next.

Supplemental strategies

We now discuss some strategies which supplement Strategies 1 and 2.

Backtracking Sometimes the weights have a tendency to lose track if left unleashed. The result could be a series of poor solutions with no signs of reaching a good neighborhood. The search starts resembling more like a blind search in the dark. Such situations can be avoided by keeping track of the obtained solutions. If a series of poor solutions is obtained, the weights can be restored to the current best or the second best set of weights. Of course application of this strategy requires that we maintain a few good sets of weights and also, that we keep track of the last few solutions and determine if they are quite far from the best solution obtained so far. The disadvantage of backtracking is that we may be over searching the same neighborhoods explored earlier. An alternative to backtracking would be use of multiple heuristics explained in the next sub-section.

Use of multiple heuristics We assume that a heuristic solution is closer to the optimal solution than a random solution and therefore, use of a heuristic solution as the starting solution is itself a strategy for biasing the search towards the bottom of a valley (hill) in the search space for a minimization (maximization) problem. Use of different heuristics essentially allows the search access to multiple good valleys (hills) thereby increasing the probability of hitting upon the globally best valley (hill). We note that any metaheuristic search technique can employ a heuristic solution as the initial solution. Use of multiple heuristics as a strategy to bias a metaheuristic search towards near-optimal solutions should not be considered as something unique to the AugNN approach as it can be used with any metaheuristic approach. In AugNN, the implementation of this approach is facilitated by the fact that the heuristics are built into the functions of the network.

Combining deterministic and stochastic strategy In some optimization problems researchers have devised some deterministic local-search strategies. For example for the traveling salesman problem we have the 2-OPT or 3-OPT strategies (Lin and Kernighan 1973) and for PERT-type scheduling problems such as task scheduling or project scheduling we have the double justification strategies (Valls et al. 2005). If the computational complexity of these deterministic local-search strategies does not exceed that of the heuristic, then use of these search strategies will bias the search towards optimal or near-optimal solutions, without adding to the computational complexity. If the complexity exceeds that of the base heuristic then such strategies may be applied sporadically or some approximations of the local-search strategies may be used. For example the exhaustive 3-OPT routine has $O(n^3)$ complexity which makes it difficult to apply but $O(n)$ approximations of 3-OPT exist (Helsgaun 2000) which makes use of this strategy very favorable. This strategy is geared more towards biasing the search towards local optima. Again, the idea of applying a deterministic local-search strategy is not something new to the AugNN approach; any metaheuristic can benefit from this strategy.

Using the search strategies outlined in this section, the search can be biased towards local and global optima.

4.2 Computational complexity

We observe that in the AugNN approach, obtaining each new solution requires the application of the heuristic. So if the computational complexity of a particular heuristic is say $O(n^m)$, then the complexity for the AugNN approach would be $k.O(n^m)$, where k is the number of iterations. Compared to perturbation mechanisms of other metaheuristics, AugNN's perturbation mechanism has a higher computational complexity. For example in genetic algorithms, a perturbation consists of rearranging the genes of two parent chromosomes to produce a child chromosome. Such perturbations can be achieved in $O(n)$ time,

irrespective of the complexity of a heuristic for that problem. In some genetic algorithms, one has to ensure the feasibility of the new solution and fix any the solution if it is infeasible. Ensuring feasibility may require extra computational effort, although the extra effort is likely to be less than the effort required to apply the heuristic each time. To justify the use of the AugNN approach, it must have some other advantages with respect to other metaheuristics to justify its use from a practical point of view.

4.3 Advantages of AugNN vs. other metaheuristic techniques

What advantages does the AugNN approach have over other metaheuristics? We have already noted that in terms of computational complexity, this approach actually has a disadvantage over other approaches because the mechanism to generate a new solution requires a complete run of the heuristic. So unless there are some overriding advantages, this approach may not be very practical. We point out three advantages. First, this approach offers a better mechanism for local search than other techniques. The second advantage is the simplicity and flexibility of search strategies. The third advantage is that feasibility is guaranteed for each new solution.

4.3.1 Local search advantage

Using the AugNN approach, by controlling the value of α, a local neighborhood of a given solution can be searched quite thoroughly. This is so because by slightly modifying the problem parameters and using the same heuristic, the new solution is still within a close neighborhood of the original solution. In GA, by contrast, even if one pair of genes is swapped, there is no guarantee that the resulting chromosome represents a solution in the same neighborhood. We illustrate this with the help of a TSP example of Fig. 1.

Let's suppose a 6-city TSP as shown in Fig. 1 is given. Let's say the NNN algorithm gives the solution in Fig. 1a and let's also assume that the tour of Fig. 1b is the optimal tour. If we use a genetic algorithm chromosome to represent the solution in Fig. 1a we get a chromosome like 1–2–5–6–3–4. In genetic algorithms, a single swap between two nodes may be regarded as a local-move. In making the swap there may be no consideration of the relative distances between the city nodes. So, a swap between nodes 2 and 6 is just as likely as a swap between 3 and 4, yet, as is clear from the picture, a swap of cities 2 and 6 gives

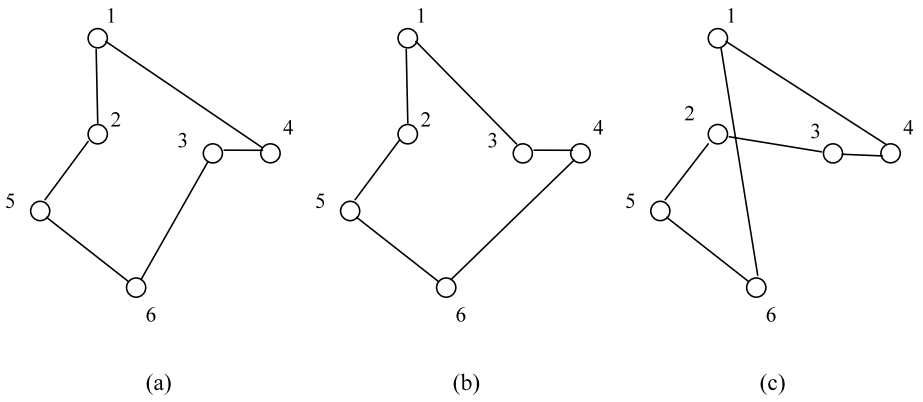

Fig. 1 TSP example

the tour in Fig. 1c, which is much worse than the tours of Figs. 1a or 1b. With the AugNN approach, with a slight modification of the distances between cities and the application of the same NNN heuristic, a move from Fig. 1a to 1b is much more likely than a move from Fig. 1a to 1c.

Due to this local search advantage, the AugNN approach wastes far fewer iterations generating poor solutions compared to other metaheuristics.

4.3.2 Simplicity and flexibility advantage

Since most of the search strategies outlined in Sect. 4.2 have to do with the magnitude of α, various search strategies can be implemented fairly simply and a variety of strategies can be tested without much coding effort. In contrast, in GA, for example, a new search strategy comes at the cost of coding a new crossover operator. This advantage also translates to the advantage of having to deal with only one parameter—the search coefficient.

4.3.3 Feasibility guarantee advantage

In other metaheuristics, each time a move or a perturbation is made, it remains to be determined whether the move resulted in a feasible solution. If an infeasible solution is obtained, then repair operators are required to bring the infeasible solution to the feasible region. In the AugNN approach there is no need for testing for feasibility or the need for a repair operator. This makes implementation that much easier. Some of the computational advantage that AugNN loses compared to other metaheuristics is regained due to this advantage.

5 Implementation

In Sect. 2 we pointed out that in the AugNN approach, a given problem is framed as a neural network of non-linear functions (input, transfer and output) in such a way that (i) the constraints of the problem and (ii) a heuristic are embedded in this network of functions. We explain the AugNN network representation with the help of a knapsack example. Say we want to solve an n-item knapsack problem. Let's suppose the heuristic we adopt is the HV heuristic. The network is shown in Fig. 2. It has an input and an output layer of neurons with one node each. The hidden layer has n nodes, one for each item. The links between the input node and each of the n hidden nodes have a weight associated with them, much like in a traditional neural network.

Table 1 shows the input, transfer and output functions and the memory items for the nodes of the neural network. The input to the hidden nodes is the weight vector element. The hidden node computes the weighted value $w_i c_i$ and the output of the hidden layer is that the node with the highest $w_i c_i$ fires a signal, while the other nodes do not fire a signal. In this way the HV heuristic is embedded in the functions of this network. The input of the output node is the signal from whichever hidden node fired. The output node computes the objective function value $\sum_{i \in N} c_i x_i$ and also determines if there is slack $b - \sum_{i \in N} a_i x_i$ and sends the same as output. It remembers which items are in the knapsack and what is the constraint capacity. In this way, the constraints of the problem are embedded in the network.

If the knapsack is full, and if the termination criteria is not met, then the best solution so far is captured, the weights are modified using some weight adjustment strategy and the cycle is repeated. As can be seen, the heuristic and the constraints are built into the network functions. So in each pass only a feasible solution is obtained.

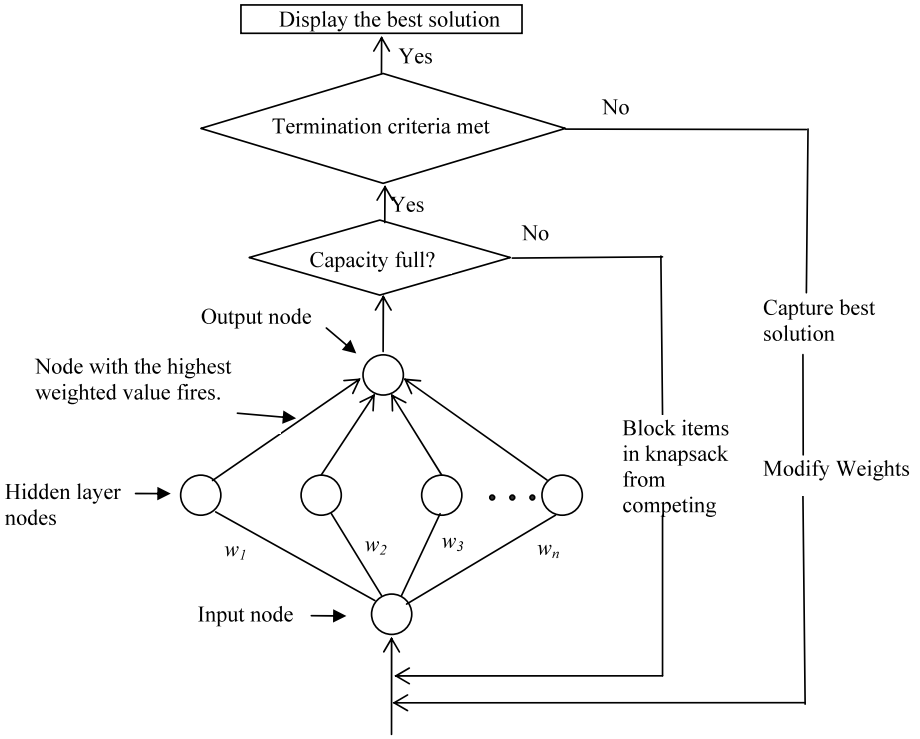

Fig. 2 An AugNN network for solving an n-item knapsack problem

Table 1 Input, transfer and output functions and memory for the AugNN nodes

	Input function	Transfer function	Output function	Memory
Input node	1	1	1	–
Hidden nodes	w_i	$w_i c_i$	1 if highest $w_i c_i$	c_i
			0 otherwise	
Output node	i	$\sum_{i\in N} c_i x_i$,	$\sum_{i\in N} c_i x_i$,	x_i, b, a_i
		$b - \sum_{i\in N} a_i x_i$	$b - \sum_{i\in N} a_i x_i$	

6 Summary and conclusions

In this paper we develop theoretical justification for the Augmented Neural Network approach as a metaheuristic for solving various optimization problems. The key element of this approach is the transformation of the search problem from the solution space of the given problem to that of a search in the weight space of a temporary weight matrix. We develop a theorem which establishes the existence of a weight matrix which gives an optimal solution to the given problem. The existence of such a weight matrix establishes the justification for the use of the AugNN approach. The paper then discusses various search

strategies to bias the search towards a good weight vector. Advantages of the approach are also discussed and finally an implementation of an AugNN for a knapsack problem is described.

The distinguishing feature of this approach is its ability to perform a local-move or a perturbation through a perturbation in the weight matrix and then applying a chosen heuristic for the given problem. This approach is still in its nascent stage of development and has so far been applied to scheduling problems. More research needs to be done on other types of optimization problems. Also, this approach holds promise as one of the partner techniques in a hybrid metaheuristic, especially a global search metaheuristic such as genetic algorithm. A combination of an effective global and an effective local search technique will of course be an effective search technique for combinatorial problems.

Appendix

A numerical example Let us consider a one-dimensional knapsack problem as follows:

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

Let $b = 15$ and let c_i and a_i be as given in Table 2.

The total number of possible subsets of items (feasible or infeasible) is ${}^5C_0 + {}^5C_1 + {}^5C_2 + {}^5C_3 + {}^5C_4 + {}^5C_5 = 1 + 5 + 10 + 10 + 5 + 1 = 32$. According to Table 3, 21 of these 32 possible subsets are feasible, i.e. subset of items that fit the knapsack capacity of 15. The best solution is the subset of items 1, 2, 5 with a value of 12.

How AugNN will solve this problem? σ : HVSR $\Upsilon : (c_i/a_i) = \{1.5, 0.625, 0.428, 0.8, 0.5\}$

Let α, the search coefficient be 0.2

Let the stopping criteria be stop after 4 iterations.

Table 4 shows the weight vector (w), the weighted problem parameter $(w.\Upsilon)$, the permutation of items $\zeta(\Upsilon_\sigma)$ based on the HVSR heuristic σ, and the solution based on those permutations.

In iteration 1, w is an identity vector. The product of $w.\Upsilon$ is basically (c_i/a_i). In iteration one, the application of the HVSR heuristic, gives a permutation of 1–4–2–5–3 based on the non-decreasing value of (c_i/a_i). This permutation gives the solution of 1, 4, 5 with a value of 11. In iteration 2, the weight vector has been modified, and updated $w.\Upsilon$ is computed.

Table 2 Values (c_i) and size (a_i) of items

Item #	Value (c_i)	Size (a_i)
1	6	4
2	5	8
3	3	7
4	4	5
5	1	2

Table 3 All possible solutions of the example problem of Table 2

#	Subset	Knapsack size	Feasible?	Obj. func. value
1	Empty	0	Feasible	
2	1	4	Feasible	6
3	2	8	Feasible	5
4	3	7	Feasible	3
5	4	5	Feasible	4
6	5	2	Feasible	1
7	1, 2	12	Feasible	11
8	1, 3	11	Feasible	9
9	1, 4	9	Feasible	10
10	1, 5	6	Feasible	7
11	2, 3	15	Feasible	8
12	2, 4	13	Feasible	9
13	2, 5	10	Feasible	6
14	3, 4	12	Feasible	7
15	3, 5	9	Feasible	4
16	4, 5	7	Feasible	5
17	1, 2, 3	19		
18	1, 2, 4	17		
19	1, 2, 5	14	Feasible	12
20	1, 3, 4	16		
21	1, 3, 5	13	Feasible	10
22	1, 4, 5	11	Feasible	11
23	2, 3, 4	20		
24	2, 3, 5	17		
25	2, 4, 5	15	Feasible	10
26	3, 4, 5	14	Feasible	8
27	1, 2, 3, 4	24		
28	1, 2, 3, 5	21		
29	1, 2, 4, 5	19		
30	1, 3, 4, 5	18		
31	2, 3, 4, 5	22		
32	1, 2, 3, 4, 5	26		

Based on the updated $w.\Upsilon$, the permutation is 1–3–4–5–2 which gives a solution of 1, 3, 5 with a value of 10. So the best solution so far is 11. In iteration 3, yet another perturbation of weights occurs, which gives a new weighted heuristic parameter vector $w.\Upsilon$**.** The new permutation gives the solution 1, 2, 5 with a value of 12. The next iteration also gives the solution of 1, 2, 5 although using a different permutation. We see that due to a perturbation in the weight vector, that $w.\Upsilon$ gets a perturbation, which perturbs the permutation $\zeta(\Upsilon_\sigma)$, which gives a new solution. It is clear that there is a permutation of items $\zeta(\Upsilon_\sigma)$ which results in one of the optimal solutions. According to Theorem 1 there exists a weight vector that gives the permutation that gives the optimal solution.

Table 4 Status of w, $(w.\Upsilon)$, $\zeta(\Upsilon_\sigma)$ and the solution for four iterations

Iteration	w	$w.\Upsilon$ or $w_{i.}c_i/a_i$	$\zeta(\Upsilon_\sigma)$	Solution from $\zeta(\Upsilon_\sigma)$	Best solution so far
1	{1,1,1,1,1}	{1.5, 0.625, 0.428, 0.8, 0.5}	1–4–2–5–3	Items: 1, 4, 5 Value = 11	11
2	{0.78, 0.8, 1.3, 0.68, 1.05}	{1.17, 0.5, 0.56, 0.54, 0.53}	1–3–4–5–2	Items: 1, 3, 5 Value = 10	11
3	{0.624, 0.85, 1.21; 0.58, 1.1}	{0.94, 0.53, 0.52, 0.46, 0.55}	1–5–2–3–4	Items: 1, 2, 5 Value = 12	12
4	{0.52, 0.89, 1.05, 0.6, 0.95}	{0.78, 0.56, 0.45, 0.48, 0.47}	1–2–4–5–3	Items: 1, 2, 5 Value = 12	12

References

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.

Cheng, R. W., & Gen, M. (1997). Parallel machine scheduling problems using memetic algorithms. *Computers & Industrial Engineering*, *33*(3–4), 761–764.

Chu, P., & Beasley, J. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, *4*(1), 63–86.

Colak, S., & Agarwal, A. (2005). Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. *Naval Research Logistics*, *52*(7), 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.), *Topics in modern scheduling* (pp. 297–318).

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

Díaz, J. A., & Fernández, E. (2006). Scatter search and Path Relinking for the capacitated p-Median problem. *European Journal of Operational Research*, *169*(2), 570–585.

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

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

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

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130.

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

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*(2), 498–516.

Martí, R., Laguna, M., & Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, *169*(2), 359–372.

Mehrotra, K., Mohan, C. K., & Ranka, S. (1997). *Elements of artificial neural networks*. Cambridge: MIT Press.

Pacheco, J. A. (2005). A scatter search approach for the minimum sum-of-squares clustering problem. *Computers & Operations Research*, *32*(5), 1325–1335.

Pitsoulis, L. S., & Resende, M. G. C. (2001). Greedy randomized adaptive search procedures. In P. M. Pardalos & M. G. C. Resende (Eds.), *Handbook of applied optimization*. London: Oxford University Press.

Reeves, C. R., & Yamada, T. (1998). Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, *6*(1), 45–60.

Rego, C. (2005). RAMP: A new metaheuristic framework for combinatorial optimization. In C. Rego & B. Alidaee (Eds.), *Metaheuristic optimization via memory and evolution: Tabu search and scatter search* (pp. 441–460). Dordrecht: Kluwer Academic.

Ribeiro, C. C., & Souza, M. C. (2002). Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics*, *118*(1–2), 43–54.

Ruiz, R., & Stutzle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop problem. *European Journal of Operational Research*, *177*(3), 2033–2049.

Stutzle, T. (1998). Applying iterated local search to the permutation flowshop problem. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt.

Valls, V., Ballestin, F., & Quintanilla, M. S. (2005). Justification and RCPSP: a technique that pays. *European Journal of Operational Research*, *165*(2), 375–386.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.