Neural metaheuristics for the multidimensional knapsack problem

Jason Deane

Department of Decision and Information Sciences

Warrington College of Business Administration

University of Florida, Gainesville, FL 32611-7169, USA

jdeane@ufl.edu

and

Anurag Agarwal*

Department of Decision and Information Sciences

Warrington College of Business Administration

University of Florida, Gainesville, FL 32611-7169, USA

aagarwal@ufl.edu

*Corresponding author

Neural metaheuristics for the multidimensional knapsack problem

Abstract

The multidimensional knapsack problem (MKP) belongs to a very important class of integer optimization problems. In this study, we propose, develop and test metaheuristics based on the neural-networks paradigm for solving the MKP. We show how domain-specific knowledge can be incorporated within the neural-network framework for solving this NP-Hard problem. We provide a mathematical formulation of the algorithm, outline the steps involved and empirically test our approach on standard benchmark problems in the literature. The results are extremely competitive with those by other heuristics and metaheuristics in the literature.

Key Words: Neural Networks, Integer optimization, Multidimensional knapsack.

1. Introduction

The multidimensional knapsack problem (MKP) has been well researched since the 1950s and continues to receive a considerable amount of attention because of its widespread real-word applicability. Many real-world problems including: the capital budgeting problem [1], the cutting stock problem [2], the cargo loading problem [3], the set covering problem [4], the distributed processor and database allocation problem [5] and the Stigler diet problem [6] can all be formulated as an MKP. Additionally, the MKP often appears as a sub problem in a variety of other larger problems. As a result of this widespread applicability, any incremental solution improvement or new, potentially promising, solution technique could prove to be extremely beneficial to many industries.

The MKP is known in the literature by many other names including: the multiple knapsack problems, the m-dimensional knapsack problem, the multi-constraint knapsack, and the 0-1 multidimensional knapsack problem, among others. We've chosen to call the problem the multidimensional knapsack problem or MKP in this paper. In this problem, we are given a set $T = \{1.....n\}$ items and a set $R = \{1....m\}$ resources. Each item $i \in T$ has associated with it a value c_i and a vector of non-negative constraint coefficients A_{ij}, which represent the amount of resource j consumed by item i. In addition, each resource $j \in R$ has a capacity b_j. The objective is to identify the subset of items which, when added to the knapsack, maximizes the total knapsack value without violating any of the resource constraints.

Formally, the problem can be stated as:

$$\text{Max} \quad \sum_{i=1}^{n} c_i x_i$$

$$\text{st:} \quad \sum_{i=1}^{n} A_{ij} x_i \leq b_j, \quad j = 1,\ldots,m$$

$$x_i \in \{0,1\}, \qquad i = 1,\ldots,n$$

Like most optimization problems that belong to the class of NP-hard problems [7], both exact and approximate solution techniques have been developed for the MKP. Existing exact solution techniques are primarily based on the branch-and-bound method [3], [8] and [9] and the dynamic programming method [2], [10], [11] and [12]. These techniques work well on small problems, but due to their inherent time complexity and the curse of dimensionality, they are not very practical for application to larger problem instances.

In an attempt to solve larger problem instances, many heuristic approaches have been developed. Three early heuristics by Senju and Toyoda [13], Kochenberger et al. [14] and Hillier [15] relied primarily on greedy methods which add or remove items from the knapsack based on a pseudo-utility ratio. Senju and Toyoda developed a dual-gradient method which begins with an infeasible solution that includes all objects in the knapsack. This method then follows a gradient path dropping one item at a time based on its value of the pseudo-utility, G_i, until feasibility is achieved.

$$G_i = c_i / \sum_{j=1}^{m} CS_j (A_{ij} b_j), \quad i = 1, \ldots, n \quad (CS_j = constraint\ surplus)$$

The item with the smallest G_i is dropped. In contrast, Kochenberger et al. developed a primal-gradient method which begins with an empty knapsack and follows the path of steepest gradient, adding one item at a time based on its value of the pseudo-utility, G_i, until no additional items can be added feasibly.

$$G_i = c_i / \sum_{j=1}^{m} (A_{ij} / RC_j), \quad i = 1, \ldots, n \quad (RC_j = residual\ capacity\ of\ resource\ j)$$

The item with the largest G_i is added. Additional heuristic methods include: combined dual gradient and greedy method [16], combined dual gradient and lagrangean relaxation method [17], surrogate-duality method [18], primal based parametric approach [19], surrogate-relaxation technique [20], [21] and [22]. For additional details about these techniques, see [23].

These heuristics produce good solutions very quickly; therefore, they can be applied successfully to very large problems, but they often leave significant gaps from the optimal solutions. In an attempt to improve the solution quality of the basic heuristics, several meta-heuristic approaches, which utilize iterative non-deterministic local-search techniques such as: Tabu Search [24], [25], [26], and [27], and Genetic Algorithms [28], [29], [30] and [31] have been developed. These metaheuristics are able to find improved solutions by searching a larger portion of the solution space; however, depending on the number of iterations required, they may be very time consuming and are often quite cumbersome and complicated to implement.

Approximate dynamic programming approach was applied to the MKP in [32]. MetaRaPs approach is used in [33].

In this paper, we apply a neural-networks based approach called Augmented Neural Networks (AugNN). AugNN is a metaheuristic approach which takes advantage of both the greedy-heuristic approach and the iterative local-search approach. AugNN utilizes a proven base heuristic to exploit the problem and domain-specific structure and then iteratively searches the local neighborhood in a somewhat random yet guided manner in an effort to improve upon the initial solution. The procedure was initially applied to the task-scheduling problem in [34]. In this approach, the MKP is formulated as a neural network of input, hidden and output nodes. Like in a traditional neural network, weights are associated with the links between the nodes. Unlike traditional neural networks, however, the input, output and activation functions are not continuous and differentiable, due to the discrete nature of the problem. The Input, output and activation functions are designed to (i) enforce the capacity constraints and (ii) apply a particular base heuristic. The chosen base heuristic provides the algorithm with a starting solution and a neighborhood in the feasible search space. After at most *n* iterations, or an epoch, a feasible solution is generated. The relative weights are modified after each epoch in an attempt to produce improved neighborhood solutions in future epochs. If improvements are found from epoch to epoch, the weights are reinforced. In addition, if an improved solution is not identified within a predetermined number of epochs, the algorithm has the ability to backtrack.

We applied the AugNN approach in conjunction with two base heuristics, Senju-Toyada's dual-gradient heuristic, henceforth ST heuristic and the Kochenberger-McCarl-Wyman's primal-gradient heuristic, henceforth KMW heuristic. Because of its learning characteristics and its ability to leverage the relative problem structure, AugNN tends to find good solutions faster than traditional meta-heuristic approaches such as Tabu Search, Simulated Annealing and Genetic Algorithms. In addition, because it doesn't involve any relaxations or LP solutions, it is a relatively simple technique in terms of computational complexity and ease of implementation.

The rest of the paper is organized as follows. In section 2, we present the AugNN formulation for the MKP. We explain the neural-network architecture for the MKP and describe all the functions and learning strategies required to implement the technique and explain the dynamic-learning strategy and the motivation behind it. In section 3, we discuss our empirical setup and provide our computational results and in section 4, we discuss our conclusions and suggestions for potential research extensions.

2. AugNN Formulation

Figure 1 shows how an MKP is framed as a NN. Node *I* acts as the input layer, nodes $T_1,\dots,T_n$ act as hidden layer nodes and node *K* acts as the final layer. One pass or iteration from the input to the output layer assigns one item to (in case of KMW heuristic) or removes one item from (in case of ST heuristic) the knapsack. One epoch consists of as many iterations as are needed to reach a feasible solution. If the obtained solution is a repeat of one of the previously obtained solutions, the entire epoch is discarded. There are weights associated with the links from the item layer to the knapsack layer. Once a unique feasible solution has been obtained, the weights are modified and a new epoch is initiated. The weight modification allows for the generation of a neighboring solution in the search space. It serves as a perturbation mechanism.

INSERT Figure 1 here

2.1 Notation used

n	: Number of Items
m	: Number of Resources
T	: Set of items = $\{1,2,\dots,n\}$
R	: Set of resources = $\{1,2,\dots,m\}$
T_i	: i^{th} item node in the item layer, $i \in T$
A_{ij}	: Requirement of resource j by item i, $i \in T$, $j \in R$
b_j	: Capacity of the j^{th} resource, $j \in R$
SUAI	: Set of unassigned items.
SURI	: Set of un-removed items.
SAI	: Set of assigned items
RF	: Reinforcement factor
BF	: Backtracking factor

α	: Learning coefficient
c_i	: Value of item i, $i \in T$
k	: Epoch number
t	: Current assignment iteration [0,n]
I	: Initial Dummy node
K	: Knapsack Node

<u>Following are all functions of assignment iteration *t*:</u>

$IFI(t)$	: Input function of the Initial node
$IFT_i(t)$	: Input function of Item nodes T_i, $i \in T$
$IFK_i(t)$	: Input function of Knapsack node from Item nodes T_i, $i \in T$
$OFI(t)$	: Output function of Initial node
$OFTK_i(t)$	: Output function of the Item node T_i to Knapsack node, $i \in T$
$OFKT_i(t)$	: Output function of Knapsack node to Item node T_i, $i \in T$
$OFKI(t)$	: Output function of Knapsack node to the Initial node I
$\theta I(t)$	: Activation function of the Initial node
$\theta T_i(t)$	: Activation function of Item node T_i, $i \in T$
$\theta K(t)$	: Activation function of the Knapsack node
$assign_i(t)$	: Item i assigned to the Knapsack, $i \in T$
$remove_i(t)$	: Item i removed from the Knapsack, $i \in T$
$RC_j(t)$	: Residual Capacity for j^{th} resource
$ERU_j(t)$	: Amount of the j^{th} resource used in excess of its capacity.
CS_j	: Constraint Surplus
$G_i(t)$	: Utility function used in the heuristics.

<u>Following are functions of epoch *k*:</u>

$\omega_i(k)$	: Weight on links from the item nodes to the knapsack node, $i \in T$
$\varepsilon(k)$	: Error or difference between solution and upper bound in epoch k
KV	: Knapsack Value

2.2 Preliminary Steps:

1. Calculate the Upper Bound (i.e. the max Knapsack Value) by solving the LP Relaxation problem.
2. Weights $\omega_i(0)$ are initialized at 1.00.

2.3 AugNN Functions

We present here, the input, transfer or activation and output functions of each layer of nodes, starting with the initial node, followed by the item nodes, and then the Knapsack node.

2.3.1 Initial Node

$t = 0$ and $k = 1$ to begin with.

Input function

$$IFI(0) = 1$$

$$IFI(t) = OFK(t), \text{ for } t > 0$$

The initial node receives a signal of 1 which sets off the first iteration of the first epoch. Thereafter, it receives an input signal from the Knapsack node, based upon which it determines whether to stop, continue with the same epoch, or to start a new epoch.

Activation State

The state of the initial node is defined by *t,* the assignment number*,* and *k*, the epoch number.

$$\theta I(t): \quad \begin{cases} t = t+1,\ k = k & \text{if } IFI(t) = 1 \\ t = 1,\ k = k+1 & \text{if } IFI(t) = 2 \\ t = 0,\ k = 0 & \text{if } IFI(t) = 0 \end{cases}$$

At the beginning, *t* is initialized to 0 and *k* is initialized to 1. An *IFI* of 1 indicates a new assignment iteration for the same epoch. In that case, *t* is incremented by one, while *k* remains the same. At the end of an epoch, signified by *IFI* of 2, *k* is incremented by 1 while *t* is reset to 1. At the end of the problem, i.e. when *IFI* is 0, both *t* and *k* are set to 0.

Output function

$\forall\ i \in T,$

$$OFI(t) = \begin{cases} 1, & if\ t > 0 \\ 0, & otherwise \end{cases}$$

Whenever $t > 0$, we must attempt to either add another item to the Knapsack for the KMW heuristic or remove another item from the Knapsack for the ST heuristic, so the initial node sends an output signal of 1 to the item nodes, signaling that if they have not yet been chosen (for addition or removal), it is time to be reconsidered.

2.3.2 Item Layer

Input function

$IFT_i(t) = OFI(t),\ \forall i \in T$

Activation function

$\forall\ i \in T,$

$\theta T_i(0)=1$

For $t > 0,$

$$\theta T_i(t)= \begin{cases} 0 & if\ \left[\theta T_i(t\text{-}1) = 0\right] \vee \left[(\theta T_i(t-1) = 1) \wedge OFKT_i(t\text{-}1) = 1\right] \\ 1 & if\ \left[\theta T_i(t\text{-}1) = 1 \wedge OFKT_i(t\text{-}1) = 0\right] \vee (IFI(t) = 2) \end{cases}$$

Explanation for the KMW Heuristic

State 1 above implies that item node T_i has not been assigned to the knapsack yet and therefore is still competing. State 0 implies that the item has been assigned to the knapsack and is therefore no longer in competition. Initially (i.e. at t=0) the state of all item nodes is initialized to 1. When the item is assigned (signified by $OFKT_i(t) = 1$), its state changes to 0 and stays that way for the rest of the current epoch. The activation state changes back to 1 when a new epoch starts (i.e. when *IFI* is 2).

Explanation for the ST Heuristic

State 1 above implies that item node T_i has not been removed from the knapsack yet. State 0 implies that the item has been removed from the knapsack. Initially (i.e. at t=0) the state of all item nodes is initialized to 1, i.e. all items are included in the knapsack. When the item has been removed (signified by $OFKT_i(t) = 1$), its state changes to 0 and stays that way for the rest of the current epoch. The state changes back to 1 when a new epoch starts (i.e. when *IFI* is 2).

Output function

$$\forall\ i \in T,$$

$$OFTK_i(t) = \theta T_i(t) * c_i$$

Explanation for the KMW Heuristic

OFTK is 0 if the item is already assigned (due to $\theta T_i(t) = 0$), and therefore this item is not to be considered for assignment. Otherwise *OFTK* sends a weighted value to the Knapsack layer indicating that the respective item is still competing and therefore should be considered for assignment.

Explanation for the ST Heuristic

OFTK is 0 if the item is already been removed (due to $\theta T_i(t) = 0$), and therefore this item is not to be considered for removal. Otherwise *OFTK* sends a weighted value to the Knapsack layer indicating that the respective item is still competing and therefore should be considered for removal.

2.3.3 Knapsack Node

Input function

$$IFK_i(t) = OFTK_i(t) * \omega_i(k)$$

Activation Function (KMW)

The activation function is a vector (RC_j, $assign_i$, KV)

$$KV = \sum_{\forall i \in SAI} c_i$$

The knapsack value is equal to the summation of the values for all of the items to which it is assigned.

$\forall i \in T,$

$$G_i(t) = IFK_i(t) / [\sum_{j=1}^{m} (A_{ij} / RC_j(t))]$$

Initially, $RC_j(t) = b_j, \forall j \in R$

<u>*Attempted assignment of an item to the Knapsack*</u>

$$assign_i(t) = \begin{cases} 1 & \text{if [(i = argmax[G}_\text{i}\text{(t)])} \wedge (RC_j(t) >= A_{ij}, \forall j \in R) \\ 0 & \textit{otherwise} \end{cases}$$

If $assign_i(t) = 1$, $SAI = SAI + i$

An attempt is made to add the item with the largest G_i to the knapsack. If the solution feasibility is not compromised, i.e. none of the resource constraints are violated, the item is added, the knapsack value and the residual resource capacities are updated, and the next iteration is initiated.

The updated residual capacity of each resource j is equal to the current residual capacity of resource j minus the amount of resource j that is required by item i.

$RC_j(t) = RC_j(t) - A_{ij}$ where i = argmax $[G_i(t)]$, $\forall j \in R$

Activation Function (ST)

The activation function is a vector (CS_j, $remove_i$, KV)

$\forall j \in R,$

$$CS_j = \max[0, \sum_{i=1}^{n} A_{ij} / b) - 1]$$

$\forall i \in T,$

$$G_i(t) = IFK_i(t) / \sum_{j=1}^{m} CS_j (A_{ij} / b_j), \forall i \in T$$

Initially,

$$ERU_j = (\sum_{i=1}^{n} A_{ij}) - b_j\ , \forall j \in R$$

Attempted removal of an item from the Knapsack

$$remove_i(t) = \begin{cases} 1 & \text{if } [\ (\mathrm{i} = \operatorname{argmin}[\mathrm{G_i}(\mathrm{t})]) \wedge (ERU_j > 0,\ for\ any\ j) \\ 0 & otherwise \end{cases}$$

If $remove_i(t) = 1$, $SRI = SRI + i$

An attempt is made to remove the item with the smallest G_i from the knapsack. If solution feasibility was not achieved in the last iteration, i.e. at least one of the resource constraints is still violated, the item is removed, the knapsack value and the residual resource capacities are updated, and the next iteration is initiated.

The updated excess amount used for each resource *j* is equal to the current excess amount used of resource *j* minus the amount of resource *j* that was required by item *i*.

$ERU_j = ERU_j - A_{ij,}$, $\forall j \in R$ where *i* is the index for Min $G_i(t)$

Output function (KMW)

$$OFKI(t) = \begin{cases} 2 & if\ \forall i \in SUAI, assign_i(t) = 0 \\ 1 & if\ assign_i(t) = 1\ for\ at\ least\ one\ i \in SUAI \\ 0 & if\ k = k_{\max} \vee\ KV = UB \vee SUAI = Null \end{cases}$$

If no items can be added to the Knapsack, and the solution obtained is unique, the knapsack node sends a signal ($OFKI(t)$=2) to the initial node indicating that it should begin a new epoch. If an item was added to the knapsack, the knapsack node sends a signal ($OFKI(t)$=1) to the initial node indicating that it should attempt to add another item by beginning a new iteration. And lastly, if the program has; (a) reached the last iteration (k = $\mathrm{k_{max}}$), (b) achieved the maximum possible value (*KV*=UB) or (c) added all of the items to the knapsack (*SUI* = Null), then the knapsack node sends a signal ($OFKI(t)$=0) to the initial node which indicates that the program is to be terminated.

$$OFK_iT(t) = \begin{cases} 1 & if\ assign_i(t) = 1 \\ 0 & otherwise \end{cases}$$

If an item is chosen to be added to the knapsack, the knapsack node sends a signal *($OFKT_i(t)=1$) to the item node; otherwise, it sends $OFKT_i(t)=0$.*

Output function (ST)

$$OFKI(t) = \begin{cases} 2 & if\ \forall j \in R, ERU_j < 0 \\ 1 & if\ ERU_j > 0\ for\ at\ least\ one\ j \in R \\ 0 & if\ k = k_{\max} \end{cases}$$

$$OFK_iT(t) = \begin{cases} 1 & if\ remove_i = 1 \\ 0 & otherwise \end{cases}$$

2.4 Learning Strategy

2.4.1 Traditional/Static Learning Strategy

The method by which we modify the augmented neural network weights is determined by the learning strategy. The learning strategy consists of the weight modification formula and any additional methods which are chosen. We employ the following learning strategy:

a. *Weight modification formula*

$$\omega_i(k+1) = \omega_i(k) + \alpha * c_i * \varepsilon(k), \quad \forall i \in T$$

where $\varepsilon(k) = UB - KV$

b. *Additional methods*

In addition, we employ reinforcement and backtracking mechanisms to improve the solution quality.

Our strategy is predicated on the theory that if the error in an epoch is too high, the order in which the items are to be chosen during the following epoch should be changed more than if the error was less.

2.4.2 Dynamic Learning Strategy

As mentioned in the introduction, the AugNN procedure utilizes the output of a base heuristic as its starting point. We often refer to the area surrounding this starting point as a 'neighborhood' within the search space. After obtaining this initial solution, the AugNN, by adjusting the relative weights, attempts to strategically search within this 'local neighborhood' in an effort to locate an improved solution. In prior AugNN research, the common practice has been to maintain a constant learning rate, α, within the weight modification formula. In theory, the size of α directly determines the amount of data perturbation and therefore the size of the target search space. Instead of utilizing a constant learning rate, we tested the potential of employing a parametric adaptation approach to controlling the size of α throughout the process. The strategic changes in the size of the learning rate occurred as a result of a set of user defined parameters as detailed in section 3.2.

The dynamic learning strategy is based on the theory that, as a result of the simplistic nature of many greedy based heuristics, it's very possible that our starting solution has placed us in a good community of neighborhoods, but not necessarily in the best 'neighborhood' within that community. Therefore, once we're relatively confident that we've located the local optima within the initial 'local neighborhood', it is in our best interest to strategically expand our search space in an effort to locate promising adjacent 'neighborhoods'.

The dynamic learning strategy is implemented in two stages.

Stage 1 – Strategic Search Space Expansion

In stage one; by placing an upper limit on the number of times that the system can backtrack at a specific learning rate, we attempt to avoid being constrained to a local neighborhood which has been thoroughly searched. If the upper limit on the number of backtracks is reached, the learning rate is iteratively increased by a user defined amount and the number of backtracks is reset to 0, theoretically expanding the target search space to include adjacent neighborhoods. This process continues until stage 1 is complete.

Stage 2 – Refocusing of the Search

In stage 2, which includes a predetermined number of epochs; we attempt to refocus the search to the best neighborhood by backtracking to the best set of identified weights and resetting the learning rate to its initial size. In this stage, we make an effort to ensure that we have completed a thorough search of the most promising 'neighborhood' which was identified during stage 1.

3. Empirical Testing and Results

In this section, we describe the experimental setup and provide computational results for the AugNN procedure using both the traditional and the dynamic learning strategies. The AugNN metaheuristic was coded in Visual Basic 6.0 and run on a Pentium-IV 2.8 GHz machine. We present comparisons of the AugNN results with those of other commonly referenced heuristics.

3.1 Problem Set

We test the AugNN procedure against the MKP dataset provided by Chu and Beasley (1998). We found it to be the most thorough and challenging of the publicly available datasets. The number of constraints, *m*, within the dataset is set to 5, 10 or 30, and the number of variables, *n*, is set to 100, 250 or 500. 30 problems are provided for each *m*-*n* combination. The right-hand side (b_i's) of the constraint coefficients were determined using: $b_j = \beta \sum_{j=1}^{n} A_{ij}$. Where, β, the constraint tightness ratio, is set to 0.25 for the first 10 problems of each set, 0.50 for the second 10 problems of each set, and 0.75 for the last 10 problems of each set. These problem sets are available at - *http://mscmga.ms.ic.ac.uk/jeb/orlib/mknapinfo.html*.

3.2 Algorithmic Parameters

Three parameters, viz., the RF (Reinforcement factor), BF (Backtracking factor) and α (Learning coefficient) determine the path the search process takes. With trial and error, using a small subset of problems, we determined the approximate range of values of RF, BF and α that gave good results. For example too high a value of α (0.1) did not give good results because it made the search too erratic. Too small of an α (0.00001), on the other hand, made the search too slow and did not give good results fast. We found that an α of 0.001 gave good results in reasonable time. Similarly, too small or too large RF and BF values also did not produce good results. An RF of 3 and a BF of 10 were found to be the best. We then fixed these parameter values and tested the algorithm on the entire set of problems.

3.3 Performance comparison criteria

The optimal solution values for most of the problems are unknown; therefore, we've adopted the evaluation procedure followed by Chu and Beasley (1998) and others in the literature. The solution quality is measured as the percentage of gap between the best solution found and an upper bound represented by the optimal LP relaxation value. The AugNN procedure was run once with KMW base heuristic and once with ST base heuristic. Each static run consisted of 100 unique epochs and 19 restarts. Each dynamic run consisted of 100 unique epochs with 9 restarts.

3.4 Performance of the AugNN Approach with two Base Heuristics

As previously stated, the power of the AugNN approach comes from its ability to leverage the domain-specific problem structure, by utilizing a proven base heuristic to obtain a good starting solution, and subsequently implementing an efficient local-search approach to explore the surrounding solution space for better solutions.

The Koch column in Table 1 summarizes for each set of problem size, the average gap using the KMW heuristic alone. The AugNN-K column gives the same gap for the AugNN metaheuristic with the KMW base heuristic. The relative percentage improvements in the gaps are shown in the next column. The next three columns do the same for the ST heuristic. Incorporation of the AugNN approach provided a considerable reduction in gap which is in excess of 50% for both heuristics.

INSERT Table 1 here

3.5 Performance comparison of the static and dynamic learning strategies

For the two dynamic learning experiments, stage 1 consisted of 92 unique epochs with varying learning rate and stage 2 consisted of 8 more unique epochs with a fixed smaller learning rate. A backtrack occurred if a solution improvement was not achieved in 10 consecutive iterations. An upper limit of 4 was placed on the number of backtracks that could occur at a specific learning rate during stage 1. Each time this upper limit was violated, the learning rate was increased by a factor of 3.

INSERT Table 2 here

The results are presented in Table 2. Incorporation of the dynamic approach provided approximately a 3% further reduction in gap.

3.6 Performance comparison of AugNN with other heuristics

Table 3 compares the results of AugNN to the results by Chu and Beasley[30], Moraga, DePuy and Whitehouse [33], Haul and Voss [31], Pirkul [18], Volgenant and Zoon [12] and Bertsimas and Demir [32]. These approaches are denoted GA-CB, MKP-M, GA-HV, MKP-P, MKP-VZ and ADP-B respectively.

INSERT Table 3 here

AugNN dominates all of the other techniques except the genetic algorithm approach by Chu and Beasley in terms of percent gap over the entire data set. Bertsimas and Demir provide an approach which also seems to be very good for the two sets of problems that they report the results for. One run of AugNN on 270 problems took on an average 8500 seconds or 31.5 seconds per problem. Of course larger problems took longer than the smaller ones. Most prior studies have not reported processing times. Chu and Beasley reported an average time of 1266 seconds per problem.

4. Summary and conclusions

We proposed, developed and tested a neural-network based metaheuristic called the augmented neural networks approach for the multidimensional knapsack problem. This is the first time a neural metaheuristic has been tried for this problem. The AugNN meta-heuristic, which combines a heuristic approach with a learning approach, performed favorably in terms of both solution time and quality on a well-known set of difficult benchmark instances. Relative to the other techniques, our technique is computationally very simple yet provides very good results. As a result of its relative simplicity and positive performance, we feel that many real world applications could potentially benefit from the application of the technique.

One obvious extension would be to explore the utilization of more complicated base heuristics. Although we would anticipate the time complexity of the overall process to increase, we would also expect the solution performance to improve as well. In addition, we explored the potential of implementing a dynamic learning strategy within the AugNN solution framework and found it to be extremely promising. Based on our experimental results we feel that future research which utilizes the Augmented Neural Network optimization technique may benefit from exploring a similar learning strategy.

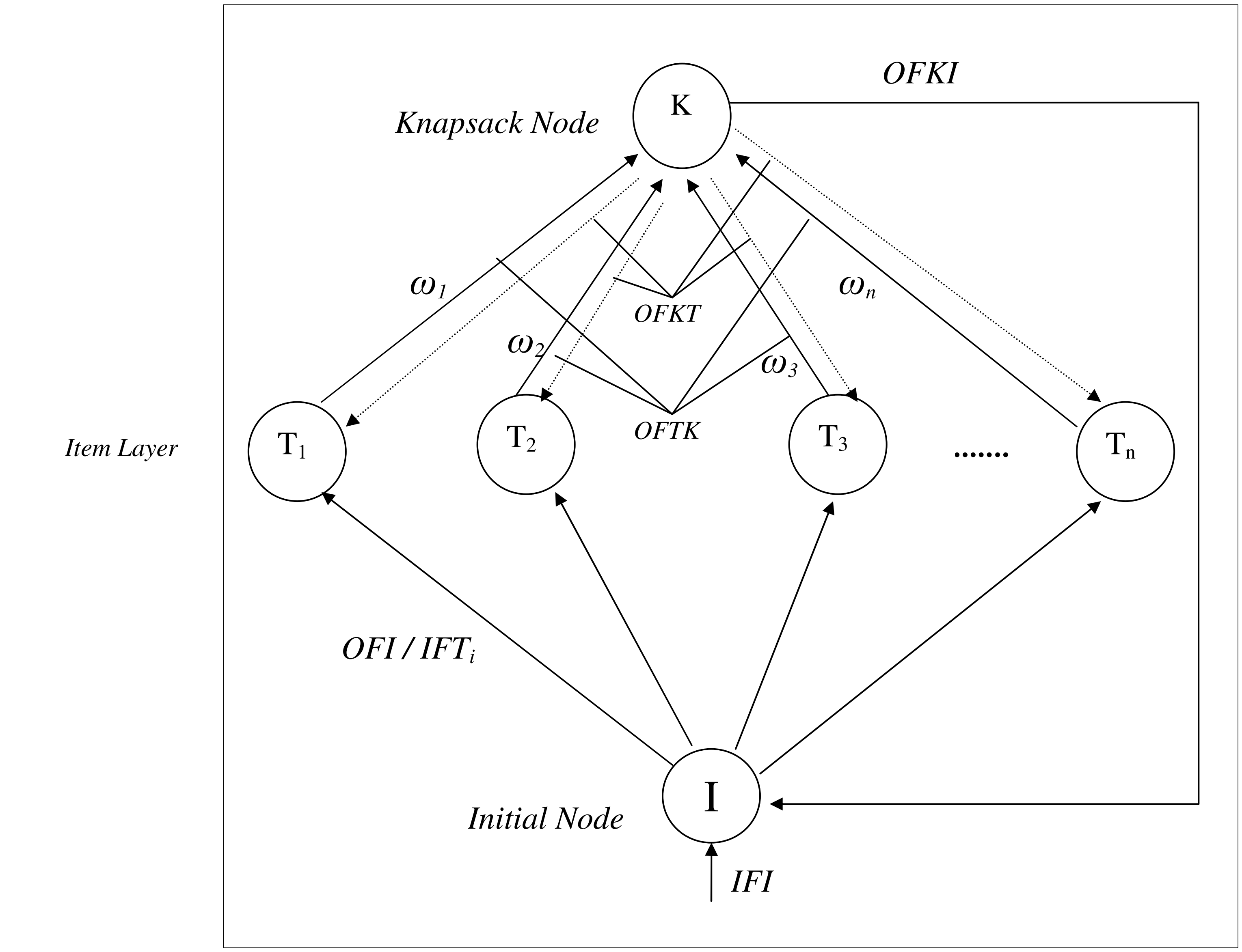

Figure 1. Graphical Representation of the AugNN for the MKP

Table 1 - Performance improvements realized by incorporation of AugNN with the chosen base heuristic

				Average% Gap			
m	**n**	**Koch**	**AugNN-K**	**% Impr**	**Senju**	**AugNN-S**	**% Impr**
5	100	1.43	0.65	54.65%	1.82	0.67	63.12%
5	250	0.55	0.21	62.05%	0.60	0.23	61.88%
5	500	0.24	0.09	61.97%	0.29	0.11	61.63%
10	100	2.79	1.22	56.27%	2.76	1.16	57.97%
10	250	1.13	0.50	55.75%	0.94	0.46	50.89%
10	500	0.44	0.22	50.38%	0.46	0.20	56.52%
30	100	4.37	2.28	47.79%	3.65	2.01	44.93%
30	250	2.08	1.23	40.77%	1.69	0.97	42.49%
30	500	1.13	0.74	34.71%	0.96	0.62	35.64%
Average		1.57	0.79	51.59%	1.46	0.71	51.15%

Table 2 - Performance comparison of the AugNN static and dynamic learning strategies

		AugNN-K		AugNN-S		Best of Both	
Problem		***Static***	***Dynamic***	***Static***	***Dynamic***	***Static***	***Dynamic***
m	n			Avg% Gap			
5	100	0.67	*0.65*	0.68	*0.67*	0.63	*0.61*
5	250	0.21	*0.21*	0.24	*0.23*	0.21	*0.20*
5	500	0.09	*0.09*	0.11	*0.11*	0.09	*0.09*
10	100	1.29	*1.22*	1.18	*1.16*	1.16	*1.13*
10	250	0.51	*0.50*	0.48	*0.46*	0.46	*0.45*
10	500	0.22	*0.22*	0.20	*0.20*	0.19	*0.19*
30	100	2.31	*2.28*	2.05	*2.01*	2.01	*1.98*
30	250	1.25	*1.23*	0.99	*0.97*	0.99	*0.97*
30	500	0.74	*0.74*	0.63	*0.62*	0.63	*0.62*
Average		*0.81*	*0.79*	0.73	*0.71*	*0.71*	*0.69*

Table 3 - Performance comparison of AugNN with other solution techniques – Solution Quality

Problem		*Average% Gap*						
m	**n**	**AugNN**	**GA-CB**	**MKP-M**	**GA-HV**	**MKP-P**	**MKP-VZ**	**ADP-B**
5	100	0.61	0.59	0.60	0.72	0.95	7.63	N/A
5	250	0.20	0.14	0.17	0.36	0.31	4.61	N/A
5	500	0.09	0.05	0.09	0.34	0.12	3.02	N/A
10	100	1.13	0.94	1.17	1.26	2.12	10.65	N/A
10	250	0.45	0.30	0.45	0.74	0.66	6.74	N/A
10	500	0.19	0.14	0.20	0.64	0.29	4.99	N/A
30	100	1.98	1.69	2.23	2.14	4.85	11.11	N/A
30	250	0.97	0.68	1.38	1.36	2.02	7.81	0.97
30	500	0.62	0.35	0.82	1.20	1.03	6.28	0.52
Average		0.69	0.54	0.79	0.97	1.37	6.98	0.75

REFERENCES

1. H. Weingartner, “Capital Budgeting of Interrelated Projects: Survey and Synthesis,” *Management Science,* vol. 12, pp. 485-516, 1966.
2. P. Gilmore and R. Gomery, “The Theory and Computation of Knapsack Functions,” *Operations Research,* vol. 14, pp. 1045-1074, 1966.
3. W. Shih, “A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem,” *Journal of Operational Research Society*, vol. 30, pp. 369-378, 1979.
4. G. Bitran and A. Hax, “Disaggregation and Resource Allocation using Convex Knapsack Problems with Bounded Variables,” *Management Science,* vol. 27, pp. 431-441, 1981.
5. B. Gavish and H. Pirkul, “Allocation of Databases and Processors in a Distributed Computing System,” *Management of Distributed Data Processing* pp. 215-231, 1982.
6. L. Lancaster, “The History of the Application of Mathematical Programming to Menu Planning,” *European Journal of Operational Research,* vol. 57, pp. 339-347, 1992.
7. M. Garey and D. Johnson, “Computers and Intractability. A Guide to the Theory of NP-Completeness,” A Series of Books in the Mathematical Sciences, ed. V. Klee., New York: W.H. Freeman and Company, 1979.
8. S. Martello and P. Toth, “A Branch and Bound Algorithm for the Zero-One Multiple Knapsack Problem,” *Discrete Applied Mathematics,* vol. 3, pp. 275-288, 1981.
9. B. Gavish and H. Pirkul “Zero-One Integer Programs with Few Constraints – Efficient Branch and Bound Algorithms,” *European Journal of Operational Research,* vol. 22, pp. 35-43, 1985.
10. H. Weingartner and D. Ness “Methods for the Solution of the Multidimensional 0/1 Knapsack Problem,” *Operations Research,* vol. 15, pp. 83-103, 1967.
11. G. Nemhauser and Z. Ullmann “Discrete Dynamic Programming and Capital Allocation,” *Management Science,* vol. 15, pp. 494-505, 1969.
12. A. Volgenant and J. Zoon “An Improved Heuristic for Multidimensional 0-1 Knapsack Problems,” *Journal of Operational Research Societ,y* vol. 41, pp. 963-970, 1990.
13. S. Senju and Y. Toyoda “An Approach to Linear Programming with 0-1 Variables,” *Management Science,* vol. 15, pp. B196-B207, 1968.
14. G. Kochenberger, B. McCarl and F. Wyman “A Heuristic for General Integer Programming,” *Decision Sciences,* vol. 5, pp. 36-44, 1974.
15. F. Hillier “Efficient Heuristic Procedures for Integer Linear Programming with and Interior,” *Operations Research,* vol. 17, pp. 600-637, 1969.
16. R. Loulou and E. Michaelides “New Greedy-Like Heuristics for the Multidimensional 0-1 Knapsack Problem,” *Operations Research,* vol. 27, pp. 1101-1114, 1979.
17. M. Magazine and O. Oguz “A Heuristic Algorithm for the Multidimensional Zero-One Knapsack Problem,” *European Journal of Operational Research,* vol. 16, pp. 319-326, 1984.
18. H. Pirkul “A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem,” *Naval Research Logistics,* vol. 34 pp. 161-172, 1987.
19. J. Lee and M. Guignard “An Approximate Algorithm for Multidimensional Zero-One Knapsack Problems – a Parametric Approach,” *Management Science,* vol. 34, pp. 402-410, 1988.

20. A. Freville and G. Plateau "Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems," *European Journal of Operational Research*, vol. 24, pp. 206-215, 1986.

21. A. Freville and G. Plateau "An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem," *Discrete Applied Mathematics*, vol. 49, pp. 189-212, 1994.

22. A. Freville and G. Plateau "The 0-1 Bidimensional Knapsack Problem: Toward and Efficient High-Level Primitive Tool," *Journal of Heuristics,* vol. 2, pp. 147-167, 1997.

23. E. Lin "A Bibliographical Survey on Some Well-Known Non-Standard Knapsack Problems," *Information Systems and Operations Research*, vol. 36, pp. 274-317, 1998.

24. F. Dammeyer and S. Voss "Dynamic Tabu List Management Using Reverse Elimination Method," *Annals of Operations Research,* vol. 41, pp. 31-46, 1993.

25. R. Aboudi and K. Jornsten "Tabu Search for General Zero-One Integer Programs Using the Pivot and Complement Heuristic," *ORSA Journal on Computing,* vol. 6, pp. 82-93, 1994.

26. F. Glover and G. Kochenberger "Critical Event Tabu Search for Multidimensional Knapsack Problems," In: Osman I.H. and Kelly J.P. (eds.). Meta-heuristics: Theory and Applications. 1996, Kluwer Academic Publishers: pp. 407-427.

27. S. Hanafi and A. Freville "Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem," In: Osman I.H. and Kelly J.P. (eds.). Meta-heuristics: Theory and Applications. 1997, Kluwer Academic Publishers: pp. 449-465.

28. S. Khuri, T. Back T and J. Heitkotter "The Zero/One Multiple Knapsack Problem and Genetic Algorithms," *Proceedings of the 1994 ACM Symposium on Applied Computing*, ACM Press 1994; pp. 188-193.

29. G. Smith, N. Steele and R. Albrecht "Artificial Neural Nets and Genetic Algorithms 3," New York: Springer-Verlag Wien, 1998. pp. 250-254.

30. P. Chu and J. Beasley "A Genetic Algorithm for the Multidimensional Knapsack Problem," *Journal of Heuristics*, vol. 4, pp. 63-86, 1998.

31. C. Haul and S. Voss "Using surrogate constraints in genetic algorithms for solving multidimensional knapsack problems," In D. L. Woodruff, *Advances in computational and stochastic optimization, logic programming, and heuristic search. Interfaces in computer science and operation research* 1997; Dordecht: Kluwer Academic Publishers: pp. 235–251.

32. D. Bertsimas and R. Demir "An Approximate Dynamic Programming Approach to Multidimensional Knapsack Problem," *Management Science*, vol. 48(4), pp. 550-565, 2002.

33. R. Moraga, G. DePuy and G. Whitehouse "Meta-RaPs approach for the 0-1 Multidimensional Knapsack Problem," *Computers and Industrial Engineering*, vol. 48, pp. 83-96, 2005.

34. A. Agarwal A., H. Pirkul and V. Jacob "Augmented Neural Networks for Task Scheduling," *European Journal of Operational Research*, vol. 151(3), pp. 481-502, 2003.

LIST OF FIGURES AND TABLE CAPTIONS