Journal of the Korean
OR/MS Society
Vol. 22, No. 3, September 1997

An AI Approach with Tabu Search to Solve Multi-level Knapsack Problems: Using Cycle Detection, Short-term and Long-term Memory*

Ilsang Ko**

Abstract

An AI approach with tabu search is designed to solve multi-level knapsack problems. The approach performs intelligent actions with memories of historic data and learning effect. These actions are developed not only by observing the attributes of the optimal solution, the solution space, and its corresponding path to the optimal, but also by applying human intelligence, experience, and intuition with respect to the search strategies. The approach intensifies, or diversifies the search process appropriately in time and space. In order to create a good neighborhood structure, this approach uses two powerful choice rules that emphasize the impact of candidate variables on the current solution with respect to their profit contribution. "Pseudo moves," similar to "aspirations," support these choice rules during the evaluation process. For the purpose of visiting as many relevant points as possible, strategic oscillation between feasible and infeasible solutions around the boundary is applied. To avoid redundant moves, short-term (tabu-lists), intermediate-term (cycle-detection), and long-term (recording frequency and significant solutions for diversification) memories are used. Test results show that among the 45 generated problems (these problems pose significant or insurmountable challenges to exact methods) the approach produces the optimal solutions in 39 cases.

1. Introduction

Suppose we have dozens of objects that have different costs, weights, volumes, and profits, and a knapsack that has limits on total cost, total weight, and total volume in which to pack them. How

* This research was supported by Chosun University Research Fund in 1995.
** Chosun University, Kwangju, Korea, isko@ic21a.chosun.ac.kr

can we put these objects in the knapsack in order to maximize the overall sum of their individual profits? This question is a typical multi-level knapsack problem, which is known to be NP-complete [4, 10]. Previously developed algorithms have used certain combinations of the simplex method, the branch and bound method, binary search, the complement concept, heuristic selection rules, lagrange relaxation, duality, and surrogate constraints. They frequently use ordering of variables according to the unit profit (C_j/A_j), or some other sort of ratio.

A branch and bound algorithm that uses the simplex method and binary search produces an exact optimal solution, but it takes tremendous amounts of computing resources to find that solution [15]. This exact method often faces a combinational explosion. In contrast, heuristic algorithms find a good solution very quickly, but do not guarantee the optimality of the solution. Because of their economical computation time, the later algorithms are more practical for real problems. One of the most classical approaches to the multi-level knapsack problem is the gradient method [20]. Because it moves from infeasible solutions toward a feasible solution as the dual simplex method does, this method is also called "the dual effective gradient method." In contrast, "the primal effective gradient method" [21] sets all variables free with value zero, calculates all gradients of the suggested function in terms of the free variables, selects one variable, and sets its value to one in the current solution until the solution reaches infeasibility. The two approaches are quite simple with respect to their logic and computation time, and both produce good solutions that are very close to the optimal solutions.

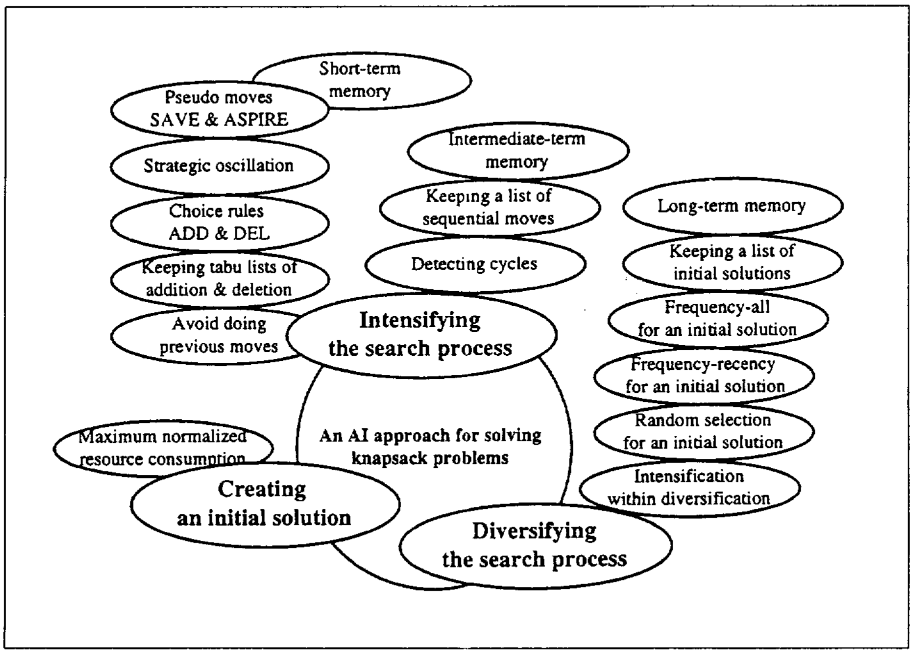

Short-term memory
Pseudo moves SAVE & ASPIRE
Strategic oscillation
Choice rules ADD & DEL
Keeping tabu lists of addition & deletion
Avoid doing previous moves
Intermediate-term memory
Keeping a list of sequential moves
Detecting cycles
Long-term memory
Keeping a list of initial solutions
Frequency-all for an initial solution
Frequency-recency for an initial solution
Random selection for an initial solution
Intensification within diversification
Intensifying the search process
An AI approach for solving knapsack problems
Maximum normalized resource consumption
Creating an initial solution
Diversifying the search process

Figure 1. Intelligent ations of the AI approach for solving knapsack problems

Consequently, they have been used to create an initial solution in some recent algorithms [13].

Some of the prior methods have their own weakness, because of their underlying assumptions, when working on a certain problem class. For example, in the reduction method [15], some of the variables are fixed to have zero or one as their values, and the remaining free variables are used to create two branches: one branch with one as its value, and the other branch with zero as its value. In an ordered variable list, some variables at both extremes have a tendency to have one (or zero in the other extreme) as their value in the optimal solution. A similar concept of reduction method has recently been applied to reducing constraints in solving multi-level knapsack problems [5].

"Strongly determined variables" [8] are very similar to the reduction method. One weakness is that sometimes these methods cannot reach the optimal solution when a fixed variable has a wrong value, for example, one instead of zero or zero instead of one as its value. In other words, they can easily be trapped at local optima that may be far from the global optimum. This situation usually occurs in a strongly-correlated problem. In performance comparisons, researchers have classified knapsack problems into three categories: random, weakly-correlated, and strongly-correlated [1]. The correlation between the profit (C_j) and resource consumptions (A_{ij}) of each variable is the criterion for this classification. But no algorithm outperforms the others in all three categories of problems. In particular, most algorithms have difficulty reaching the optimal solution in a strongly-correlated problem. This is unfortunate since this type of problem represents the most realistic case.

Up to now, most heuristic algorithms lack a long-run planning. They assume a short-sighted plan and a constant decision environment. They are too static, and lack adoptability during the search process. They do not have any scheme to incorporate information about the dynamically changing space during the search process. Intensification and diversification, and their dynamic structural constitution provide a fresh approach compared to the current algorithms. Utilizing the short-term, intermediate-term, and long-term memories, and deriving a learning mechanism from them are most promising.

In this study, we developed an AI Approach with tabu search to solve multi-level knapsack problems. Figure I describes several concepts of intelligence that are utilized in this approach. The concepts of choice rules ADD and DEL, pseudo moves SAVE and ASPIRE, tabu lists, strategic oscillation, cycle detection, and the diversification process are explained later in their corresponding sections.

2. A Mathematical Definition of the Multi-level Knapsack Problem

A multi-level knapsack problem is the knapsack problem extended to have multiple constraints. It is defined as follows;

$$\begin{array}{ll} \text{MAX} & Z = CX' \\ \text{Subject to:} & AX' \ <= B' \\ & X_j = 0 \text{ or } 1 \text{ for all } j. \end{array}$$

$$\begin{array}{ll} \text{Where} & C = [C_1, C_2, C_3, \ldots, C_n] \\ & X = [X_1, X_2, X_3, \ldots, X_n] \\ & A = A_{11}, A_{12}, A_{13}, \ldots, A_{1n} \\ & \qquad \cdots \quad \cdots \\ & \qquad A_{m1}, A_{m2}, A_{m3}, \ldots, A_{mn} \\ & B = [B_1, B_2, B_3, \ldots, B_m] \\ & \text{All } C_j, A_{ij}, B_i\text{'s are positive.} \end{array}$$

This problem has been well-used to model project selection problems [18], capital budgeting [17], resource allocation problems, and so on. In this problem, multiple constraints, reflecting the various resource consumptions of each variable, make the solution algorithms more complicated than in a knapsack problem with only one constraint. Observations about the attributes of the optimal solution, its solution space, and its corresponding solution path are crucial to developing intelligent actions of an ideal approach. These attributes are described as follows;

1. The optimal solution has the largest objective value within the given resource capacities.
2. The optimal solution has resource consumptions that are feasible and very close to the boundary between feasible and infeasible solutions. But the optimal solution may not be the solution closest to the boundary.
3. The optimal solution has many neighbors. These neighbors are determined by choice rules for the next moves. Consequently, different choice rules create different neighborhood structures.
4. The optimal solution can be reached by many possible paths. These paths are determined by the solution space and choice rules. In other words, different solution spaces and different choice rules are expected to create different solution paths.
5. There may be several optimal solutions that have the same objective value, but may have

different resource consumptions. The number of variables in the optimal solutions can vary.

Because of the attributes 2 and 4, the integration of strategic oscillation and pseudo moves SAVE and ASPIRE makes it possible to find optimal or near optimal solutions in a few computation time.

3. Creating an Initial Feasible Solution

Creating a good initial solution is very important in order to reduce the amount of computation time needed to reach the optimal solution. The variables which produce high unit-profits in terms of their resource consumptions are attractive enough to be included in an initial solution. For example, in a knapsack problem, unit-profits (C_j / A_j) are employed to constitute an initial solution in several heuristic approaches [2, 14]. In a multi-level knapsack problem, unit-profits in terms of surrogates of resource consumptions of individual variables are also used to create an initial feasible solution [16].

In order to create an initial feasible solution, our approach uses the maximum resource consumption of variables normalized by profits and capacities [$\text{MAX} \{ A_{ij} / (C_j * B_i)\}$] (this value will be called the maximum normalized resource consumption). This value indicates the maximum relative resource consumption of each variable in order to create one unit of profit. The variables are arranged in increasing order based on the maximum normalized resource consumption ($\text{MIN} [\text{var} \mid \text{MAX} \{ A_{ij} / (C_j * B_i)\}]$). The variables on the ordered list are introduced one at a time into the solution until the introduction of the next variable would produce an infeasible solution. An example of creating an initial feasible solution is in Figure 2.

4. Intensification

An intensification strategy attempts to find a better solution in a restricted region. In a multi-level knapsack problem, this strategy can be performed intelligently by two choice rules ADD and DEL, strategic oscillation, tabu-lists, pseudo moves SAVE and ASPIRE, and cycle detection. These intelligent actions are activating appropriately in time and space according to the current space and historic information. The details of an intensification process is in Figure 3.

An Example

$$
\begin{array}{ll}
\textbf{MAX} & 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10} \\
\textbf{S.T.} & 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} <= 45 \\
 & 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} <= 50 \\
 & 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} <= 40 \\
 & \text{All } X_j = 0 \text{ or } 1
\end{array}
$$

Variables	x_1	x_2	x_3	x_4	x_5
A_{1j} / (B_1 *C_j)	15/(45*20) = 0.0166	16/(45*18) = 0.0197	12/(45*15) = 0.0177	12/(45*14) = 0.0190	10/(45*12) = 0.0185
A_{2j} / (B_2 *C_j)	22/(50*20) = 0.0220	21/(50*18) = 0.0233	16/(50*15) = 0.0213	14/(50*14) = 0.0200	15/(50*12) = 0.0250
A_{3j} / (B_3 *C_j)	18/(40*20) = 0.0225	20/(40*19) = 0.0277	15/(40*15) = 0.0250	10/(40*14) = 0.0178	9/(40*12) = 0.0187
MNRC^	0.0225	0.0277	0.0250	0.020	0.0250
Priority	2	7	4	1	5

Variables	x_6	x_7	x_8	x_9	x_{10}
A_{1j} / (B_1 *C_j)	10/(45*9) = 0.0246	8/(45*7) = 0.0253	5/(45*5) = 0.0222	4/(45*3) = 0.0296	3/(45*2) = 0.0333
A_{2j} / (B_2 *C_j)	7/(50*9) = 0.0155	5/(50*7) = 0.0142	2/(50*5) = 0.0080	4/(50*3) = 0.0266	4/(50*2) = 0.0400
A_{3j} / (B_3 *C_j)	8/(40*9) = 0.0222	2/(40*7) = 0.0253	6/(40*6) = 0.0300	2/(40*3) = 0.0166	5/(40*2) = 0.0625
MNRC^	0.0246	0.0253	0.0300	0.0296	0.0625
Priority	3	6	9	8	10

MNRC^ = Maximum Normalized Resource Consumption
The created initial solution = (x_4 x_1 x_6), The objective value = 43

Figure 2. Creating an initial solution

4.1 Choice rules ADD and DEL

Intelligent choice rules have an important role in creating a good neighborhood structure that may lead to the optimal solution. Some classical approaches for creating such a structure use unit-profits and surrogates of normalized resource consumptions (by capacities) of candidate variables. Glover's [6], Geoffrion's [5] and Petersen's [17] approaches are typical examples. Kochenberger, McCarl, and Wyman's [12] approach uses unit-profits of candidate variables and surrogates of their resource consumptions in terms of slack resources (resource consumptions are normalized by slack resources). These approaches usually catch the best candidate variable. But, in some cases, they can not select the best move because they do not consider the total profit and total resource consumptions of the current solution. The best candidate variable for the next move can be determined not only from the merit of the variable, but also from the relationship between the variable and the current solution.

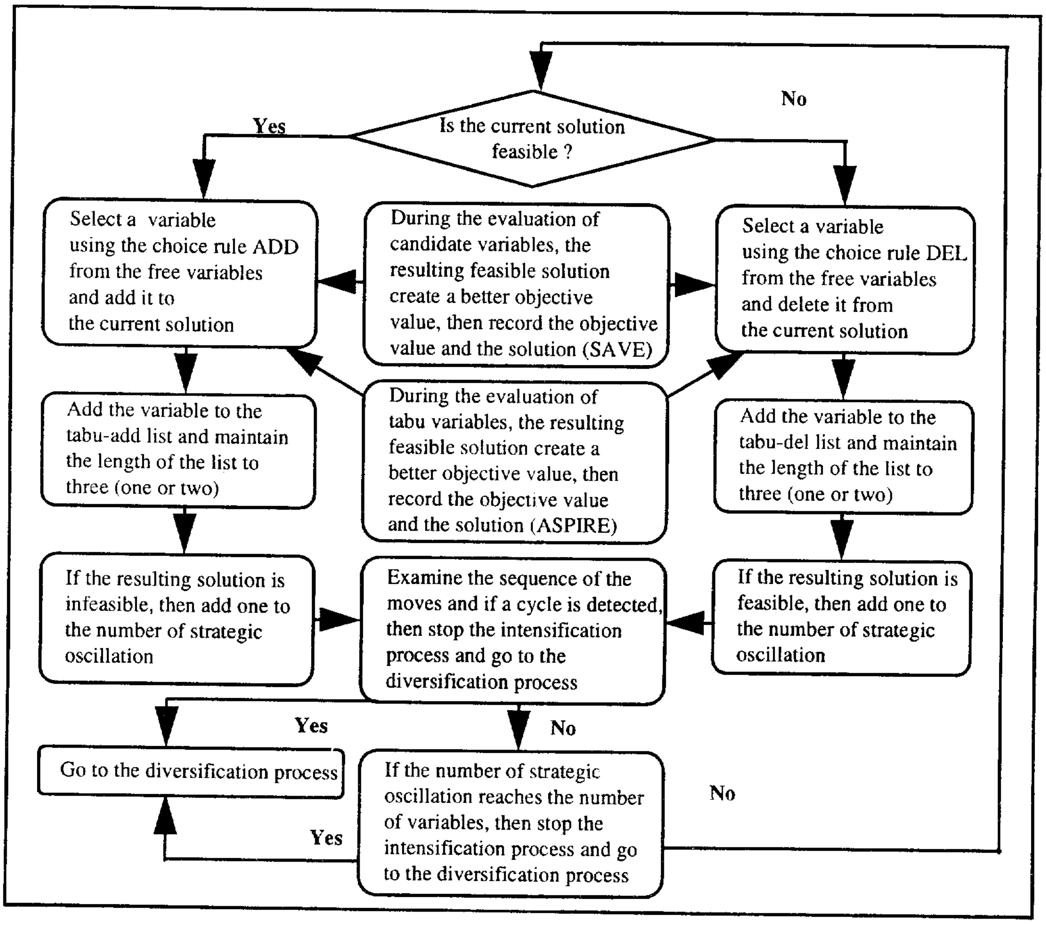
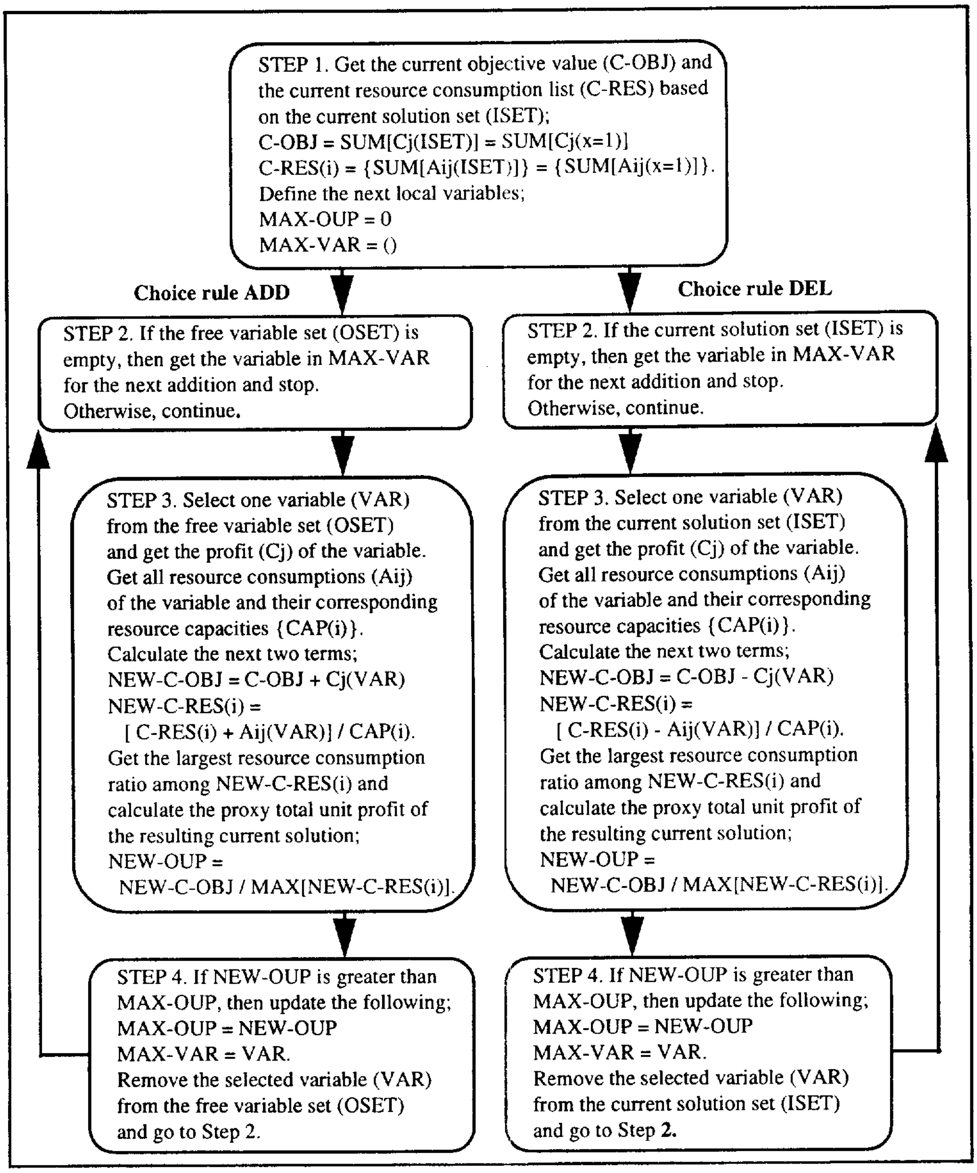

Is the current solution feasible ?
Yes
No
Select a variable using the choice rule ADD from the free variables and add it to the current solution
During the evaluation of candidate variables, the resulting feasible solution create a better objective value, then record the objective value and the solution (SAVE)
Select a variable using the choice rule DEL from the free variables and delete it from the current solution
Add the variable to the tabu-add list and maintain the length of the list to three (one or two)
During the evaluation of tabu variables, the resulting feasible solution create a better objective value, then record the objective value and the solution (ASPIRE)
Add the variable to the tabu-del list and maintain the length of the list to three (one or two)
If the resulting solution is infeasible, then add one to the number of strategic oscillation
Examine the sequence of the moves and if a cycle is detected, then stop the intensification process and go to the diversification process
If the resulting solution is feasible, then add one to the number of strategic oscillation
Yes
No
Go to the diversification process
If the number of strategic oscillation reaches the number of variables, then stop the intensification process and go to the diversification process
Yes
No

Figure 3. The intensification process

It is better to reflect the state of the current solution in the choice rules for adding or deleting a variable. In evaluating the next potential moves, measures are desirable to reflect at least some part of the change of the objective value [$a(CX' - CX'')$] and some part of the change of the current solution's resource consumption [$b(AX' - AX'')$] where X' is the next solution, X'' is the current solution, and a and b are greater than zero but less than one [6]. Two such measures are choice rule ADD for evaluating variables to be added, and choice rule DEL for evaluating variables to be deleted. Choice rule ADD calculates the ratio of the total profit to the maximum normalized resource consumption of the resulting current solution by adding a candidate variable. On the other hand, choice rule DEL selects a leaving variable that maximizes the utility of the total resource consumption of the resulting current solution by calculating its similar proxy total unit profit. The details of the two rules are given in Figure 4.



STEP 1. Get the current objective value (C-OBJ) and the current resource consumption list (C-RES) based on the current solution set (ISET);
C-OBJ = SUM[Cj(ISET)] = SUM[Cj(x=1)]
C-RES(i) = {SUM[Aij(ISET)]} = {SUM[Aij(x=1)]}.
Define the next local variables;
MAX-OUP = 0
MAX-VAR = ()

Choice rule ADD

STEP 2. If the free variable set (OSET) is empty, then get the variable in MAX-VAR for the next addition and stop.
Otherwise, continue.

STEP 3. Select one variable (VAR) from the free variable set (OSET) and get the profit (Cj) of the variable.
Get all resource consumptions (Aij) of the variable and their corresponding resource capacities {CAP(i)}.
Calculate the next two terms;
NEW-C-OBJ = C-OBJ + Cj(VAR)
NEW-C-RES(i) =
[C-RES(i) + Aij(VAR)] / CAP(i).
Get the largest resource consumption ratio among NEW-C-RES(i) and calculate the proxy total unit profit of the resulting current solution;
NEW-OUP =
NEW-C-OBJ / MAX[NEW-C-RES(i)].

STEP 4. If NEW-OUP is greater than MAX-OUP, then update the following;
MAX-OUP = NEW-OUP
MAX-VAR = VAR.
Remove the selected variable (VAR) from the free variable set (OSET) and go to Step 2.

Choice rule DEL

STEP 2. If the current solution set (ISET) is empty, then get the variable in MAX-VAR for the next addition and stop.
Otherwise, continue.

STEP 3. Select one variable (VAR) from the current solution set (ISET) and get the profit (Cj) of the variable.
Get all resource consumptions (Aij) of the variable and their corresponding resource capacities {CAP(i)}.
Calculate the next two terms;
NEW-C-OBJ = C-OBJ - Cj(VAR)
NEW-C-RES(i) =
[C-RES(i) - Aij(VAR)] / CAP(i).
Get the largest resource consumption ratio among NEW-C-RES(i) and calculate the proxy total unit profit of the resulting current solution;
NEW-OUP =
NEW-C-OBJ / MAX[NEW-C-RES(i)].

STEP 4. If NEW-OUP is greater than MAX-OUP, then update the following;
MAX-OUP = NEW-OUP
MAX-VAR = VAR.
Remove the selected variable (VAR) from the current solution set (ISET) and go to Step **2.**

Figure 4. The choice rule ADD and DEL

4.2 Strategic oscillation with feasibility

Strategic oscillation around the boundary between feasible and infeasible solutions is a powerful scheme that allows the search to visit as many relevant points as possible. The general assumption is that the optimal solution is a point that is very close to the boundary, but may not be the closest. If the current solution is feasible, the process adds variables, one by one, until the solution violates its

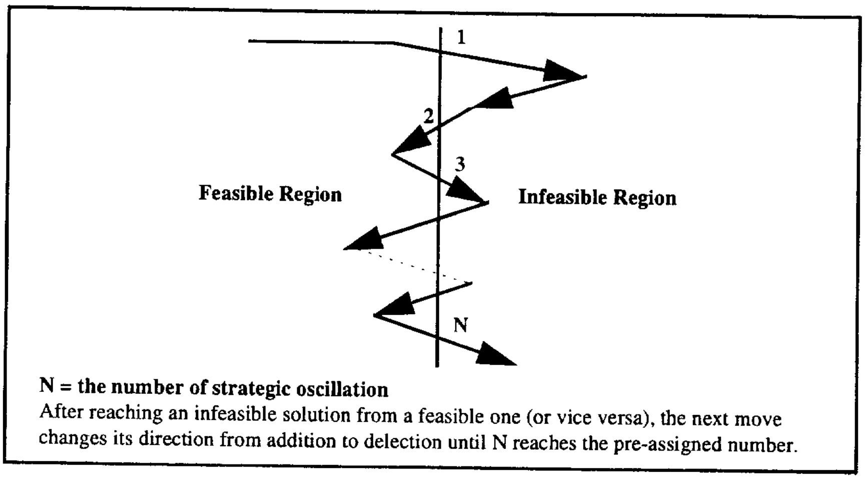

1
2
3
N
Feasible Region
Infeasible Region
N = the number of strategic oscillation
After reaching an infeasible solution from a feasible one (or vice versa), the next move changes its direction from addition to delection until N reaches the pre-assigned number.

Figure 5. Strategic Oscillation

feasibility, attempting to move close to the boundary. If the current solution is infeasible, the process deletes variables sequentially until the current solution becomes feasible, and visits a relevant point very close to the boundary that may be optimal. Figure 5 shows an example of strategic oscillation.

4.3 The length of tabu-lists

Tabu-lists (tabu-add-list and tabu-del-list) hold a few variables that have recently been added to or deleted from the recent solutions. They force the effect of those changes on the current solution and its objective value to endure for the next several iterations. Tabu-lists are called "the short-term memory," and make the previous changes last for the next several moves.

A careful configuration of strategic oscillation and tabu lists often provides a substantial improvement in the quality of the solution. For example, if we add one variable and delete another variable three times sequentially, and if the length of the tabu-list for those variables is three, this has the same effect as adding three variables together and deleting three other variables at the same time. In addition, the effect of adding two variables and deleting one variable or adding one variable and deleting two variables is achieved simultaneously. As a result, strategic oscillation around the boundary between feasible and infeasible solutions is a variant of exchanging a bundle of variables.

4.4 Pseudo moves SAVE and ASPIRE

Pseudo moves are intelligent tools which can be used during a search process. They compensate for the possible drawbacks of the choice rules in creating a effective neighborhood structure. Pseudo moves SAVE and ASPIRE are designed for such a purpose, and attempt to maximize the possibility of visiting the optimal solution. These two schemes can catch the optimal solution if the solution path goes near the optimal solution, even it does not visit the optimal exactly. They are very similar to "aspiration" as defined by Glover [8, 9].

A. Pseudo move SAVE

The purpose of pseudo move SAVE is to support choice rules ADD and DEL by making a side effect of recording a possible better feasible move which creates a better objective value than the best value ever found during the evaluation of candidate variables. More specifically, even though a candidate move is not the best candidate chosen by the choice rules ADD and DEL, if the candidate move can produce a better objective value that is also feasible, the pseudo move SAVE identifies this new best solution and records its move and the resulting objective value (as if this move had been an actual move during the search process). In some cases, this move is not the best candidate that should be chosen, but simply recording the move has an important contribution toward visiting the optimal solution.

B. Pseudo move ASPIRE

In some cases, the moves in the tabu lists (tabu-add-list and tabu-del-list) can produce a feasible move and a better objective value than the best value ever found. These moves have a "tabu" status, which means that they are protected from being added into or deleted from the current solution for the next several iterations. During each iteration, the variables in the tabu lists can also be evaluated to check this possibility, and if one of them is appropriate, that move is simply recorded with its new best objective value. This kind of move is another "pseudo move" and can also play a critical role in visiting the optimal solution. Consequently, if a variable in the tabu lists produces a feasible solution which has a better objective value than the best value ever found, pseudo move ASPIRE detects this possibility and records the solution.

4.5 Stopping rules for intensification

A. A pre-assigned number of strategic oscillation

The most simple stopping rule for an intensification process is a pre-assigned number of strategic oscillations, which is heuristically set to the number of variables in a problem. This pre-assigned number is an approximate combination of all possible different moves with all different variables. If the number of strategic oscillation becomes greater than the pre-assigned number, then the current intensification process stops.

B. Cycle detection

In the case of using deterministic choice rules, detecting a cycle provides a more intelligent stopping rule for an intensification process than the pre-assigned number of strategic oscillations. During the search process, if the same solutions are visited again with a cycle, there is no possibility of visiting a better solution. Detecting a cycle provides an effective stopping rule.

A cycle can be defined as "visiting repeatedly the same solution with the same objective value after a fixed number of iterations." In order to detect a cycle, we need to maintain at least two lists about the same information: one is the entire information (for example, the entire sequence of moves or solutions) and the other is a significant part (for example, recent 5 moves or recent 5 solutions) of the entire information. This part should change at each iteration in order to reflect recent information. The accuracy of cycle detection is totally dependent on the quality and quantity of this part. For example, maintaining recent 20 solutions can detect a cycle more accurately than maintaining only recent 5 solutions.

Incomplete information can be used to detect a cycle. For example, recording two lists, one with all moves in a sequence and the other with the recent 5 or 7 moves in a sequence, is usually enough to successfully detect a cycle in uncorrelated knapsack problems. When adding or deleting a variable,

An Example: Adding a variable to the current feasible solution

Candidate Variables	x_3	x_5	x_7	x_2	x_9	x_8	x_{10}
NEW-OBJ	43+15 = 58	43+12 = 55	43+7 = 50	43+18 = 61	43+3 = 46	43+5 = 48	43+2 = 45
C-RES$_1$	37+12 = 49	37+10 = 47	37+8 = 45	37+16 = 53	37+4 = 41	37+5 = 42	37+3 = 40
C-RES$_2$	43+16 = 59	43+15 = 58	43+5 = 48	43+21 = 64	43+4 = 47	43+2 = 45	43+4 = 47
C-RES$_3$	36+15 = 51	36+9 = 45	36+2 = 38	36+20 = 56	36+2 = 38	36+6 = 42	36+5 = 41
Feasibility	I	I	F	I	F	I	I
C-RES$_1$/CAP$_1$	1.0888	1.0444	1	1.1777	0.9111	0.9333	0.8888
C-RES$_2$/CAP$_2$	1.18	1.16	0.96	1.28	0.94	0.9	0.94
C-RES$_3$/CAP$_3$	1.275	1.125	0.95	1.4	0.95	1.05	1.025
MNRC	1.275	1.16	1	1.4	0.95	1.05	1.025
OUP	45.4901	47.4137	50	43.5714	48.4210	45.7142	43.9024
Next Move			*				

NEW-OBJ = the new objective value, C-RES = the current resource consumption, I = infeasible, F = feasible, CAP = the capacity, MNRC = the maximum normalized resource consumption
OUP = the overall unit profit (NEW-OBJ / MNRC)

Figure 6. The first iteration of the intensification process

we can compare two lists to check whether the latter list becomes a subset of the former list. If the latter list is a subset of the former list, then there is a high probability that a cycle exists.

For example, Let List-A and List-B be the following:

List-A = (X_1 X_3 X_4 X_2 X_5 X_6 X_7 X_1 X_3 X_2 X_5 X_6 X_7 X_1)

List-B = (X_2 X_5 X_6 X_7 X_1).

In this example, List-A is a list of all moves in an intensification process, and the variables represent added or deleted variables in a sequence. List-B is the most recent 5 moves in a sequence. In each iteration, the chosen variable is added to the end of the two lists. In List-B, the length of the list is maintained as 5 by deleting the first variable if the length becomes 6 by the addition of the chosen variable. In order to detect a cycle, we can simply check whether or not any five sequential moves in LIST-A is the same with LIST-B. The current List-B is clearly a subset of List-A, except for the last subset with 5 moves (X_2 X_5 X_6 X_7 X_1). A cycle is detected. In some cases, it may not be a cycle, and additional information such as the resulting objective function values can be recorded simultaneously for a better detection. Figure 6 shows an example of the first iteration of an intensification process to solve the same problem in Figure 2.

5. Diversification

A diversification strategy is used to encourage the search procedure to move to less-explored regions of the solution space. The basic scheme for diversification is to frequently restart the search process in a new region in order to find a near-optimal or optimal solution. For this purpose, a long term memory device (like frequency counts) can be used to find a more promising region where the optimal solution may exist. In a multi-level knapsack problem, a frequency count of each variable (the number of times the variable appears in a solution) can be recorded, and this information can be used effectively during the following intensification and diversification processes.

First, "intensification within diversification" is described as a simple learning process. This memory records recent significant moves such as improving solutions. Based on this information, an inference engine with If-Then rules guides the search process to make decisions on the promising solution space to find the optimal. If there exists no improving solution, then in the following main diversification process, random selection and recent frequency of the variables are used to create initial solutions to continue to search.

5.1 Intensification within diversification

During the search process, valuable information can be gathered, and used later for better solutions. One such intelligent scheme is "long-term memory" [9]. This long-term memory can contribute to intensifying or diversifying the search process. It is very important to determine what information should be recorded and how it can be utilized for future diversification. Avoiding redundant moves is a good criteria from which to start. For this purpose, some of the significant solutions and their related paths are recorded. For example, we can record every initial feasible solution when restarting the search in a new region for diversification. If recording all initial solutions is too expensive, then alternatively some of significant initial solutions can be recorded in order to reduce the computational burden. The record of these initial solutions can be used to protect redundant moves and redundant solution paths between diversification processes.

Simple record-keeping with frequency of the recent three best improving solutions (FREQUENCY-3BEST), frequency of all improving solutions (FREQUENCY-IMPROVING), and frequency of all the current feasible and infeasible solutions (FREQUENCY-ALL) can be intelligently implemented for diversification and "intensification within diversification." In order to record three best improving solutions, the global variable "IMPROVING-ISET" is created. At the end of each diversification process, if IMPROVING-ISET is still empty, FREQUENCY-3BEST does not need to be updated. This information handling process is explained by the two processes; one is the information-acquiring process and the other is the information-utilizing process.

A. Information-acquiring process

a). Updating FREQUENCY-3BEST

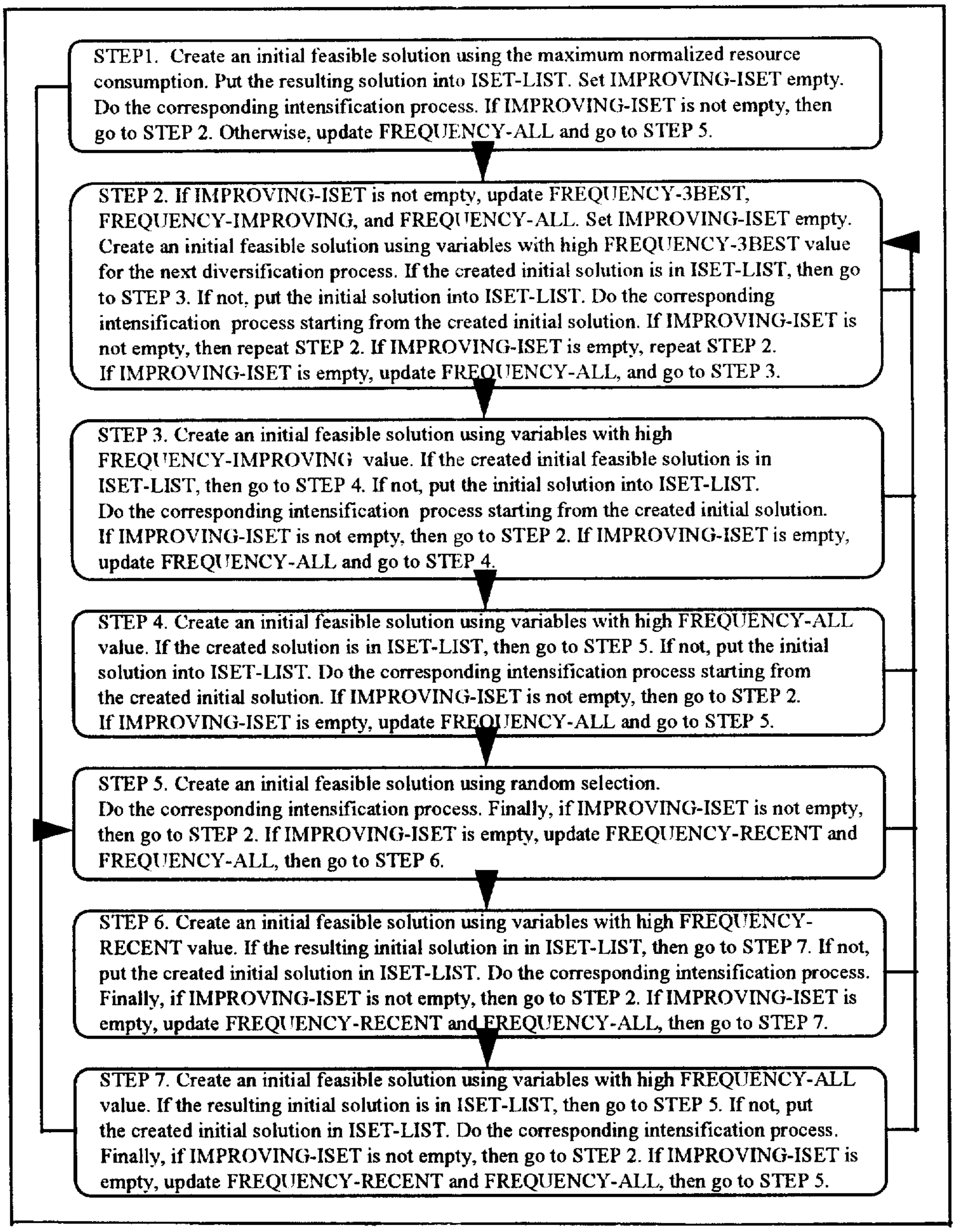
During each diversification process, record, at maximum, the 3 most recent best improving solutions in IMPROVING-ISET, and update FREQUENCY-3BEST in all variables in these solutions by counting their appearance at the end of the diversification process.

b). Updating FREQUENCY-IMPROVING

If any improving solutions occur during a diversification process, then update immediately FREQUENCY-IMPROVING by adding 1 to the existing value of the FREQUENCY-IMPROVING of variables in those solutions.

c). Updating FREQUENCY-ALL

Each time when adding a variable in the current solution, record the current iteration for that variable. Each time when deleting a variable, update FREQUENCY-ALL of that variable by counting their tenures in the current solution using the difference between their exiting iterations (the current iteration) and their iterations of entering the current solution. At the end of the diversification process, record the current solution (ISET). Update FREQUENCY-ALL of the



STEP1. Create an initial feasible solution using the maximum normalized resource consumption. Put the resulting solution into ISET-LIST. Set IMPROVING-ISET empty. Do the corresponding intensification process. If IMPROVING-ISET is not empty, then go to STEP 2. Otherwise, update FREQUENCY-ALL and go to STEP 5.

STEP 2. If IMPROVING-ISET is not empty, update FREQUENCY-3BEST, FREQUENCY-IMPROVING, and FREQUENCY-ALL. Set IMPROVING-ISET empty. Create an initial feasible solution using variables with high FREQUENCY-3BEST value for the next diversification process. If the created initial solution is in ISET-LIST, then go to STEP 3. If not, put the initial solution into ISET-LIST. Do the corresponding intensification process starting from the created initial solution. If IMPROVING-ISET is not empty, then repeat STEP 2. If IMPROVING-ISET is empty, repeat STEP 2. If IMPROVING-ISET is empty, update FREQUENCY-ALL, and go to STEP 3.

STEP 3. Create an initial feasible solution using variables with high FREQUENCY-IMPROVING value. If the created initial feasible solution is in ISET-LIST, then go to STEP 4. If not, put the initial solution into ISET-LIST. Do the corresponding intensification process starting from the created initial solution. If IMPROVING-ISET is not empty, then go to STEP 2. If IMPROVING-ISET is empty, update FREQUENCY-ALL and go to STEP 4.

STEP 4. Create an initial feasible solution using variables with high FREQUENCY-ALL value. If the created solution is in ISET-LIST, then go to STEP 5. If not, put the initial solution into ISET-LIST. Do the corresponding intensification process starting from the created initial solution. If IMPROVING-ISET is not empty, then go to STEP 2. If IMPROVING-ISET is empty, update FREQUENCY-ALL and go to STEP 5.

STEP 5. Create an initial feasible solution using random selection. Do the corresponding intensification process. Finally, if IMPROVING-ISET is not empty, then go to STEP 2. If IMPROVING-ISET is empty, update FREQUENCY-RECENT and FREQUENCY-ALL, then go to STEP 6.

STEP 6. Create an initial feasible solution using variables with high FREQUENCY-RECENT value. If the resulting initial solution in in ISET-LIST, then go to STEP 7. If not, put the created initial solution in ISET-LIST. Do the corresponding intensification process. Finally, if IMPROVING-ISET is not empty, then go to STEP 2. If IMPROVING-ISET is empty, update FREQUENCY-RECENT and FREQUENCY-ALL, then go to STEP 7.

STEP 7. Create an initial feasible solution using variables with high FREQUENCY-ALL value. If the resulting initial solution is in ISET-LIST, then go to STEP 5. If not, put the created initial solution in ISET-LIST. Do the corresponding intensification process. Finally, if IMPROVING-ISET is not empty, then go to STEP 2. If IMPROVING-ISET is empty, update FREQUENCY-RECENT and FREQUENCY-ALL, then go to STEP 5.

Figure 7. The diversification process

variables in this solution by counting their tenures (the difference between the final iteration of the intensification process and each variable's entering iteration).

B. Information-utilizing process

The updated frequencies are used to create initial solutions for the next several diversification processes in a sequence. In creating an initial solution for each diversification process with the updated information, human intelligence can easily be represented with If-Then rules. These rules can create a partial learning effect that allows the search procedure to find better solutions around the best value recently found. Step 2, Step 3, and Step 4 in Figure 7 show a cascading execution of these rules.

During each step, information-acquiring process is performed simultaneously, and FREQUENCY-3BEST, FREQUENCY-IMPROVING, and FREQUENCY-ALL are updated. Steps, 2, 3, and 4 have important roles in locating good solutions. These steps are diversification processes, especially devoted to intensifying the search process within a very restricted area. Consequently, these steps are called a process of "intensification within diversification" using high frequency information. During this process, if no improving solution is found, then we continue to diversify the search process using random selection in STEP 5 until the current iteration number exceeds a preset limit.

5.2 The main diversification process

We define FREQUENCY-RECENT as the frequency of the variables in the current solution in each diversification process. During every diversification process, we can collect FREQUENCY-RECENT, and simply accumulate it to FREQUENCY-ALL. More specifically, in every diversification process, FREQUENCY-RECENT is set to zero at the beginning, accumulated during the process, and finally, added to FREQUENCY-ALL at the end of the process.

Variables that have occurred with higher frequency have a higher priority for constructing an initial solution for the corresponding intensification process. In order to obtain unbiased frequency information, random selection (every other diversification process) is used to create an initial solution for a diversification process. Then, for the next two diversification process, the updated information contained in FREQUENCY-RECENT and FREQUENCY-ALL is used to create its initial solution. The three construction processes alternate every other time. Using these three alternating diversification processes provides intelligent learning not only in the short run, but also in the long run.

To determine whether the created initial solution is exactly the same as an initial solution in the list (ISET-LIST), "matching" can be performed [19]. The way to perform "matching" is, without reordering, to record the initial solutions as they are. When determining whether the created initial solution is the same as one in the list, we can, first, check whether the two initial solutions have the same number of variables. If they do, we check whether every member of the variables in the created initial solution is also a member of the initial solution in the list. If any of the variables are different, it is determined that the two initial solutions are not the same.

For example, let us assume two initial solutions previously created are in the initial solution list

(ISET-LIST), and a new initial solution is created by using FREQUENCY-ALL data. They are described as follows:

ISET-LIST = ((X3 X10 X4 X2) (X10 X5 X2))

A new initial solution = (X10 X2 X5)

The newly created initial solution is already a member of ISET-LIST, and its corresponding intensification will certainly create the same redundant solution path if we use deterministic choice rules ADD and DEL with deterministic tabu-list lengths. In this example, the two initial solutions, (X10 X5 X2), and (X10 X2 X5), are equivalent even though they have different orders.

In general, STEP 5 is a kind of information-acquiring (or information-generating) process, and STEP 6 and STEP 7 are another type of information-utilizing process. These three processes are oscillating strategically during the diversification processes. Only after a diversification process with random selection, do we use high frequencies to create an initial solution for the next diversification.

6. Generating Problems

Several articles in the literature classify knapsack problems into four categories: uncorrelated, weakly-correlated, strongly-correlated, and value-independent, based on the relationships between the profits (C_j) and resource consumptions (A_{ij}). For multi-level knapsack problems, the formulae of Balas and Zemel [1] are modified slightly and extended. For the computer experiments to test the suggested approaches in this study, five types of problems are generated with the following formulae:

(1) Uncorrelated	$1 <= C_j <= 1000$
	$1 <= A_{ij} <= 1000$
(2) Weakly-correlated	$1 <= C_j <= 1000$
If Cj <= 500,	$1 <= A_{ij} <= C_j + 500$
otherwise,	$Cj + 500 <= A_{ij} <= C_j + 500$
(3) Moderately-correlated	$1 <= C_j <= 1000$
If Cj <= 300,	$1 <= A_{ij} <= C_j + 300$
otherwise,	$Cj + 300 <= A_{ij} <= C_j + 300$
(4) Strongly-correlated	$1 <= C_j <= 1000$
If Cj <= 100,	$1 <= A_{ij} <= C_j + 100$
otherwise,	$C_j + 100 <= A_{ij} <= C_j + 100$
(5) Extremely-correlated	$1 <= C_j <= 1000$

If $C_j <= 10$, $1 <= A_{ij} <= C_j + 10$

otherwise, $C_j + 10 <= A_{ij} <= C_j + 10$

These formulae first choose C_j from an uniform random distribution and next create A_{ij} according to the type of problems. For the convenience of creating multiple A_{ij}'s for multi-level knapsack problems, this sequence is the opposite of Balas and Zemel [1], which first create A_j and next C_j.

In order to add more variety to the problem set, we introduce some deviations to the resource capacities. Since A_{ij}'s range from 1 to 1000, the sum of A_{ij}s for a single resource becomes approximately the product of 500 and the number of variables (500 * N). One third to one half of this product value is randomly assigned to the resource capacities. This appears to produce challenging problems. Furthermore, large deviations between resource capacities is unrealistic, and is avoided. Under the above considerations, resource capacities of each problem are determined by using the following formulae:

FN = the number of variables in each problem

NUM_1 = (integer (FN / 4))

NUM_2 = (integer (FN / 6))

NUM_3 = NUM_1 - NUM_2

(Random NUM_3) = a uniform random integer between 0 to (NUM_3 - 1)

Resource capacity (i) = [1000 * (NUM_1 + (random NUM_3) + 1)]

As these formulae show, the resource capacities of a problem have a distribution from (1000 * NUM1) to (1000 * NUM_2). For the problems actually created, the 20 variable problems have a range of 4000 to 5000, the 40 variable problems have a range of 8000 to 10000, and the 60 variable problems have a range of 11000 to 15000. 45 problems are created: 9 uncorrelated, 9 weakly-correlated, 9 moderately-correlated, 9 strongly-correlated, and 9 extremely-correlated. Each set of 9 problems consists of three 10-constraint / 20-variable problems, three 20-constraint / 40-variable problems, and three 30-constraint / 60-variable problems. In order to differentiate these problems, two letters are used to reflect their correlation, four digits are used to indicate the number of variables and the number of constraints in each problem, and the final one digit is used as an index to differentiate the same sized problems. For example, UC10201 means the uncorrelated problem with 10 constraints, 20 variables, and number 1 problem of this type. Additionally, WC is used for weakly-correlated, MC for moderately-correlated, SC for strongly-correlated, and EC for extremely-correlated.

7. Test Results

An Experiment is performed to test the AI approach with tabu search on the 45-generated problems (9 uncorrelated, 9 weakly-correlated, 9 moderately-correlated, 9 strongly-correlated, and 9 extremely-correlated). The approach uses high frequency of the variables and random selection alternatively for diversification. The total number of iterations is determined approximately by the product of the numbers of diversification trials and strategic oscillations. This value is the square of the number of variables like N * N. But, from our findings in this experiment, the number of total iterations should be at least 2/3 * N * N because the number of strategic oscillations (N) creates slightly more than N moves. This means returning to the feasible region may sometimes involve adding or deleting more than one variable. Finally, the total number of iterations are limited to 1000 for the 20 variable problems, 2000 for the 40 variable problems, and 4000 for the 60 variable problems.

The real computation time of the suggested approaches can be represented as the following:

O (1/2 * N * N * C * D) where N is the number of variables in a problem.

In this order function, 1/2 * N indicates the number of candidate variables to be evaluated for each move, the next N is the number of moves resulting from strategic oscillation in an intensification process, C is the number of constraints in the problem, and D represents the number of diversification processes, which contain their corresponding intensification processes.

The results of the test are described in Table 1. 39 optimal solutions are visited: all cases in uncorrelated and weakly-correlated problems, 8 cases in moderately-correlated and strongly-correlated problems, and 5 cases in extremely-correlated problems. The approach produces 8626 instead of the optimal solution (8630) in one moderately-correlated problem (MC20403). In one strongly-correlated problem (SC20401), the approach produces 8567, for which the optimal solution is 8577. In four extremely-correlated problems (EC20402, EC20403, EC30602 and EC30603), the approach produces 8024, 8025, 11036 and 11031 instead of the optimal solutions such as 8027, 8026, 11037 and 11032. The results are compared with each other in terms of problem sizes, types, and iterations needed to reach the optimal, and are described in Table 2. The table clearly shows that the more correlated and the larger the problem is, the more iterations it needs to reach the optimal.

8. Conclusions and Future Research

The basic premise of this study comes from the belief that if we have some valuable information about a problem such as the attributes of the optimal solution, its solution space and its solution paths, and if we can perform some intelligent actions, appropriately in time and space, we can hopefully find the optimal or near optimal solution within a small number of trials. The AI approach tries to do its best to find the optimal solution by utilizing intelligent actions appropriately in time and space. These intelligent actions are imitated from human intelligence. During the past experiment, we conclude that the basic idea is successfully accomplished. The AI approach that uses three as tabu-list length is the best one. In 60 variable / 30 constraint problems, LINDO usually takes several hundred thousand iterations to obtain the optimal or near-optimal solution (approximately about 10 hours in a pentium PC). But our AI approach usually takes one or two thousand iterations. In comparing computation time, our approach consumes less than one tenth of the computation time of LINDO.

In this study, a simple learning effect is designed so that it may be achieved from the information about improving solutions. Another more AI comes from random selection and frequency information, and helps to guide the search to more regions of the solution space. On the other hand, information acquiring and utilizing processes are very expensive because of their time consuming nature. They should be appropriately designed with effective data structures and if possible, their use should be minimized. In addition, cycle detection is very intelligent but expensive, and it should be used carefully. Several intelligent actions such as cycle detection, pseudo moves, matching, strategic oscillation, diversification within intensification, and intensification within diversification can easily be used for other difficult problems.

This study provides a good example of utilizing human heuristic intelligence to solve a multi-level knapsack problem, that is NP-complete. The AI approach, which uses the oscillation of the information-acquiring processes and the information-utilizing process, provides a wonderful framework for a learning effect. Collecting unbiased information, and utilizing that information with learning in solving combinatorially difficult problems are the most promising for future research. The results of this study show the superiority of the approach. Long-run testing in the future is needed to confirm this superiority. Research Opportunities also exist in the area of probabilistic approaches in relation to Simulated Annealing and Genetic Algorithms. The suggested intelligent actions can be used for solving the Generalized Assignment Problem, the Traveling Salesman Problem or any pure 0-1 integer problems including Job-shop Scheduling.

Table 1. Test Results

Tabu-list-lengths				1		2		3	
Problem	Opt.	U-bound	Cor.	best	itr (nim ns na)	best	itr (nim ns na)	best	itr (nim ns na)
UC20101	5227	5457.249	0.21954	5227*	76 (3, 1, 0)	5227*	66 (4, 1, 0)	5227*	37 (3, 2, 0)
UC20102	5946	6314.669	0.15504	5946*	23 (4, 0, 0)	5946*	7 (4, 0, 1)	5946*	7 (4, 0, 2)
UC20103	6285	6570.273	0.21058	6285*	53 (5, 0, 0)	6285*	13 (5, 0, 0)	6285*	21 (5, 0, 0)
UC40201	11801	12149.430	0.12667	11801*	1237 (9, 0, 0)	11801*	1444 (11, 2, 3)	11785	365 (9, 0, 1)
UC40202	11916	12115.281	0.13344	11916*	2 (3, 0, 0)	11916*	2 (3, 0, 0)	11916*	2 (3, 0, 0)
UC40203	11275	11518.479	0.16097	11268	1 (2, 0, 0)	11275*	425 (3, 0, 0)	11275*	140 (3,0,1)
UC60301	17766	18110.964	0.08925	17766*	73 (7, 2, 0)	17766*	1402 (10, 1, 0	17766*	691 (8,1,0)
UC60302	18800	19070.982	0.10760	18800*	56 (5, 0, 0)	18800*	20 (5, 0, 1)	18800*	40 (6, 1, 0)
UC60303	18646	18896.923	0.10796	18646*	358 (7, 1, 0)	18646*	30 (7, 1, 0)	18646*	1457 (14, 1, 2)
WC20101	4499	4964.515	0.56324	4499*	257 (4, 1, 0)	4499*	162 (5, 2, 0)	4499*	92 (5, 2, 0)
WC20102	3839	4165.678	0.60884	3839*	372 (7, 1, 0)	3839*	385 (8, 2, 1)	3839*	593 (7, 3, 3)
WC20103	4527	4774.148	0.68886	4527*	15 (7, 3, 0)	4527*	55 (7, 4, 0)	4527*	74 (6, 3, 1)
WC40201	8438	8749.125	0.63734	8438*	303 (14, 7, 0)	8438*	154 (15, 6, 1)	8438*	1030 (13, 3, 2)
WC40202	8787	9081.054	0.56653	8787*	644 (9, 4, 0)	8787*	268 (5, 2, 0)	8787*	1461 (7, 1, 0)
WC40203	8262	8543.669	0.60213	8262*	300 (5, 0, 0)	8262*	109 (5, 0, 0)	8262*	414 (5, 0, 1)
WC60301	11473	11738.018	0.63062	11473*	1533 (23, 8, 0)	11473*	3876 (17,7,2)	11473*	418 (17, 6, 5)
WC60302	11377	11618.511	0.67980	11360	1926 (18, 12, 0)	11377*	679 (20,10,1)	11377*	1691 (19, 7, 3)
WC60303	11626	11889.077	0.58207	11626*	1276 (20, 9, 0)	11626*	92 (15, 3, 3)	11626*	3279 (15, 4, 3)
MC20101	4012	4177.624	0.85000	4012*	68 (7, 4, 0)	4012*	47 (7, 2, 2)	4012*	69 (4, 1, 0)
MC20102	4115	4298.980	0.90800	4115*	873 (11, 7, 0)	4082	200 (10, 6, 0)	4115*	31 (10, 4, 1)
MC20103	4397	4508.292	0.85100	4397*	1 (2, 0, 0)	4397*	1 (2, 0, 0)	4397*	1 (2, 0, 0)
MC40201	8173	8326.062	0.85660	8173*	265 (13, 4, 0)	8173*	973 (13, 4, 1)	8173*	342 (14, 3, 1)
MC40202	8384	8557.174	0.84627	8384*	109 (14, 5, 0)	8384*	234 (17, 5, 0)	8384*	659 (14, 5, 2)
MC40203	8630	8807.713	0.86047	8626	55 (11, 4, 0)	8626	733 (12, 3, 1)	8626	1824 (12, 3, 2)
MC60301	11962	12122.913	0.85470	11962*	2738 (14, 5, 0)	11962*	2700 (19, 7, 3)	11962*	642 (12, 4, 1)
MC60302	12030	12208.771	0.85754	12019	247 (16, 3, 0)	12029	1238 (21, 3, 0)	12030*	304 (13, 2, 1)
MC60303	12141	12324.132	0.86597	12141*	741 (16, 9, 0)	12141*	2349 (18, 8, 0)	12141*	649 (18, 8, 1)
SC20101	3992	4050.174	0.98177	3892	170 (5, 3, 0)	3992*	115 (4, 2, 0)	3992*	179 (6, 1, 0)
SC20102	3895	3969.677	0.96206	3895*	15 (5, 2, 0)	3895*	15 (6, 2, 1)	3895*	79 (7, 2, 1)
SC20103	4026	4097.951	0.97748	4026*	220 (8, 3, 0)	4026*	126 (6, 2, 0)	4026*	256 (7, 1, 1)
SC40201	8577	8635.621	0.98490	8568	20 (10, 8, 0)	8567	20 (10, 7, 1)	8567	20 (10, 7, 1)
SC40202	8148	8195.965	0.98041	8131	542 (10, 8, 0)	8148*	353 (11, 9, 0)	8148*	1254 (12, 8, 4)
SC40203	8281	8329.444	0.98387	8281*	207 (11, 5, 0)	8281*	24 (12, 5, 1)	8281*	24 (13, 6, 2)
SC60301	11378	11444.120	0.98103	11366	1416 (19, 9, 0)	11371	3487 (18, 9, 2)	11378*	597 (17, 11, 2)
SC60302	11364	11417.530	0.97895	11360	2110 (19, 11, 0)	11347	1882 (18, 13, 0)	11364*	332 (16, 8, 2)
SC60303	11322	11363.727	0.98042	11309	145 (18, 8, 0)	11295	1832 (22, 10, 1)	11322*	3253 (20, 8, 2)
EC20101	4002	4010.561	0.99980	3999	923 (8, 5, 0)	3998	128 (8, 5, 1)	4002*	405 (8, 4, 2)
EC20102	4008	4021.287	0.99982	4008*	207 (9, 6, 0)	4008*	146 (6, 3, 1)	4008*	53 (5, 2, 1)
EC20103	4008	4013.704	0.99977	4006	177 (6, 4, 0)	4006	517 (8, 5, 1)	4008*	156 (8, 2, 2)
EC40201	8024	8030.749	0.99975	8023	74 (9, 7, 0)	8022	208 (9, 7, 0)	8024*	238 (10, 7, 1)
EC40202	8027	8035.268	0.99978	8024	12 (10, 3, 0)	8024	12 (10, 3, 0)	8024	12 (10, 3, 0)
EC40203	8026	8027.635	0.99986	8025	73 (9, 7, 0)	8025	286 (10, 6, 2)	8025	11 (7, 3, 1)
EC60301	11042	11048.040	0.99982	11042*	3854 (14, 8, 0)	11040	1015 (15, 8, 0)	11041	1199 (13, 7, 0)
EC60302	11037	11041.260	0.99982	11036	1317 (10, 8, 0)	11036	1183 (11, 9, 0)	11036	1286 (10, 7, 1)
EC60303	11032	11038.370	0.99978	11031	344 (14, 11, 0)	11031	3140 (18, 12, 3)	11030	2839 (16, 11, 2)

Opt.= the optimal solution, **U-bound** = Upper bound, **Cor.** = Correlation, **best** = the best solution found, * = the optimal solution found, **itr** = the number of iteration, **nim** = the number of improving moves, **ns** = the number of SAVE, **na** = the number of ASPIRE, UC = Uncorrelated, WC = Weakly correlated, MC = Moderately correlated, SC = Strongly correlated, EC = Extremely correlated

Table 2. The average iteration and time for the best solution among the various problem type

Problem type and size	Avg. iterations	Avg. seconds*
UC1020s	34	4
UC2040s	402	90
UC3060s	459	172
WC1020s	223	27
WC2040s	520	117
WC3060s	1641	615
MC1020s	143	17
MC2040s	577	130
MC3060s	1290	484
SC1020s	131	16
SC2040s	274	62
SC3060s	1673	627
EC1020s	301	36
EC2040s	103	23
EC3060s	1804	676
	* seconds in a pentium 75 MHz PC	

References

[1] Balas, E. and E. Zemel, "An Algorithm for Large Zero-One Knapsack Problems," Operations Research, Vol. 28, No. 5, September-October (1980), 1131-1154.

[2] Balas, E. and C. H. Martin, "Pivot and Complement - A Heuristic for 0-1 Programming," Management Science, Vol. 26, No. 1, January (1980), 86-96.

[3] Freville, A. and G. Plateau, "An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem," Proceedings of the conference "Viewpoints on Optimization" Grimentz, September 1990, Switzerland.

[4] Garey, M. and D. Johnson, ***Computers and Intractability: A Guide to the theory of NP-Completeness***, Freeman, San Francisco, 1979.

[5] Geoffrion, A. M., "An Improved Implicit Enumeration Approach for Integer Programming," Operations Research, Vol. 17, No. 3 (1969), 437-454.

[6] Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem,"

Operations Research, Vol. 13, No. 3 (1965), 879-919.

[7] Glover, F., "Heuristics for Integer Programming using Surrogate Constraints," Decision Sciences, Vol. 8 (1977), 156-166.

[8] Glover, F., "Artificial Intelligence, Heuristic Frameworks and Tabu Search," Managerial and Decision Economics, Vol. 11 (1990a), 365-375.

[9] Glover, F., "Tabu Search: A Tutorial," Center for Applied Artificial Intelligence, University of Colorado, February 1990b.

[10] Karp, R., "Reducibility Among Combinatorial Problems," in ***Complexity of Computer Computations***, Miller, R. E. and J. W. Thatcher (eds), Plenum Press, New York, 1972, 85-103.

[11] Kelly, J. P., B. L. Golden, and A. A. Assad, "Large-scale Controlled Rounding using Tabu Search with Strategic Oscillation," Annals of Operations Research, 1993.

[12] Kochenberger, G. A., B. A. McCarl, and F. P. Wyman, "A Heuristic for General Integer Programming," Decision Sciences, Vol. 5, No. 1 (1974), 36-44.

[13] Lee, J. S. and M. Guignard, "An Approximate Algorithm for Multidimensional Zero-One Knapsack Problems-A Parametric Approach," Management Science, Vol. 34, No. 3 (1988), 402-410.

[14] Magazine, M. J. and O. Oguz, "A Heuristic Algorithm for the Multidimensional Zero-One Knapsack Problem," European Journal of Operational Research, Vol. 16 (1984), 319-326.

[15] Martello, S. and P. Toth, "A New Algorithm for the 0-1 Knapsack Problem," Management Science, Vol. 34, No. 5 (1988), 633-644.

[16] Nemhauser, G. L., and L. A. Wolsey, Integer and combinatorial Optimization, A Wiley-Interscience Publication, 1988

[17] Petersen, C. C., "A Capital Budgeting Heuristic Algorithm Using Exchange Operation," AIIE Transactions, Volume 6, No. 2, June (1974), 143-150.

[18] Petersen, C. C., "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects," Management Science, Vol. 13, No. 9, May (1967), 736-750.

[19] Rich, E., ***Artificial Intelligence***, McGraw-Hill, Inc. 1983

[20] Senju, S. and Y. Toyoda, "An Approach to Linear Programming with 0-1 Variables," Management Science, Vol. 15, 1968, 196-207.

[21] Toyoda, Y., "A simplified Algorithm for Obtaining Approximate Solutions to 0-1 Programming Problems," Management Science, Vol. 21, No. 12 (1975), 1417-1427.