Solving the 0-1 Multidimensional Knapsack Problem using Tabu Search and Visualization

Trung Tuan Dung[1], Steven Halim[2], Wan Wee Chong[3], Lau Hoong Chuin[4]

[1]SRP Student, Anglo Chinese Junior College
[2]1st year Graduate, National University of Singapore
[3]School of Computing, National University of Singapore
[4]School of Computing, National University of Singapore

Abstract

In many industrial and business sectors, solving complex optimization problems is the key to increasing productivity and product quality. Often times, it is computationally intractable to search for the optimal solution out of a very large number of possible solutions. Heuristic techniques for solving optimization problems offer a computationally efficient way to find near-optimal solutions. Meta-heuristics (such as Tabu Search and Genetic Algorithms) is a top-level general algorithmic strategy that guides heuristics search. In this project, we apply an existing Meta-heuristics Development Framework developed by a team of NUS students to solve a notorious optimisation problem, at the same time study the positive effects of visualization when used in conjunction with the framework. The resulting system was benchmarked against the input instances in the Operations Research Library, which has the results for the best algorithms so far. The experimental results obtained are within 0.54% and 1.89% from the best-published results on average and in the worst case respectively.

1. Introduction

1. 1. Combinatorial Optimization Problems

Finding optimal solution(s) for combinatorial optimization problems in many areas such as business, engineering and science still poses a real challenge, despite the impact of recent advances in mathematical programming and computer technology. It is because many such problems belong to the class of NP-hard problems. In complexity theory, no polynomial time algorithm is known to find optimal solution(s) for these problems; or in other words, no algorithm is expected to solve them exactly within a reasonable time limit. Examples of such problems are: Traveling Salesman Problem, Quadratic Assignment Problem, Vehicle Routing Problem, etc [Garey and Johnson, 1979]. In this research project, the problem of concern is the Multidimensional 0-1 Knapsack Problem (MKP), which is an NP-hard problem.

There exist exact methods to solve NP-hard problems, such as Branch & Bound (B&B) algorithm. However, even though B&B is effective in pruning the search space, its strength depends on the bounding formula, which varies from problem to problem. In general, B&B is ineffective for larger instances of the problems, especially if these problems must be solved in real-time.

1. 2. Meta-Heuristics

Instead of solving these problems exactly, a user is often satisfied if he can obtain good solutions within a reasonable time. The term “good” here refers to solutions that are close (if not equal) to the optimal solution. This relaxation leads many researchers to develop various approximation and meta-heuristics algorithms.

This field of meta-heuristics has been evolving quickly in recent years with various techniques such as *Simulated Annealing, Tabu Search, Genetic Algorithms, Scatter Search, Greedy Randomized Adaptive Search Procedure, Variable Neighborhood Search, Ant Systems,* and their hybrids. They are currently among the most efficient and robust optimization strategies to find high-quality solutions for many real-life optimization problems. A very large number of successful applications of meta-heuristics are reported in the literature, books, journals and conference proceedings ([Blum and Roli, 2003], [Ribeiro and Hansen, 2002]).

1. 3. Tabu Search

Our project involves the use of the tabu search engine in the Meta-heuristics Development Framework (MDF) [Lau *et al.*, 2004], which was developed by a team of NUS students, two of whom have guided me throughout the project.

With MDF, algorithm developers:

- simply **translate** a specific problem (MKP in this case) to MDF interfaces,
- are freed from the **mundane** task of implementing meta-heuristic algorithms,
- can focus on **search strategies** to improve search quality.

The primary project objective is to apply the Tabu Search Engine in the MDF in conjunction with a visualizer component to solving MKP.

1. 4. Problem Definition

The **Multidimensional 0-1 Knapsack problem (MKP)** is a classified *NP-hard* optimization problem. It can be defined formally as:

Input:

- A set of N objects **O** = $\{o_1,...o_N\}$ and their profit values $\{p_1,...p_N\}$.
- Capacities of a knapsack **K =** $\{b_1,...b_M\}$.

Output:

- A subset **O'** of objects in **O** to be stored in the knapsack such that the total profit of the selected objects is **maximized,** while satisfying the knapsack capacities.

The special case of the MKP with m = 1 is the classical knapsack problem (*KP01*). Note that although KP01 is not strongly NP-hard because there exist polynomial approximation algorithms for it, MKP, on the other hand, is strongly NP-hard [Garey and Johnson, 1979].

MKP can be used to define many practical problems such as *capital budgeting* where project j has profit p_j and consume (r_{ij}) units of resource i. The goal is to determine a subset of the n projects such that the total profit is maximized and all resource constraints are satisfied. Other important applications include *cargo loading* [Shih, 1979], *cutting stock problems*, and *processor allocation in distributed systems* [Gavish and Pirkul, 1982].

2. Materials and Methods

2. 1. Tabu Search Framework (TSF) in MDF applied to MKP

The MDF was written in C++ and contains a set of interfaces to epitomize the routine tabu search procedures. There are key interfaces that must be implemented for each specific problems. In Object-Oriented terms, MDF uses abstraction and inheritance as the primary mechanism to build adaptable components or interfaces. In this project I make use of the Tabu Search Framework (TSF), which is one of the engines available in MDF. The architecture of MDF is shown in Figure 1.

Apart from the User-defined Interfaces, the interfaces that I implemented include the Solution, Objective Function, Neighborhood Generator, Move, and the Control Mechanism, all of which are problem-specific, at the same time following the framework standards. Details are omitted in this report.

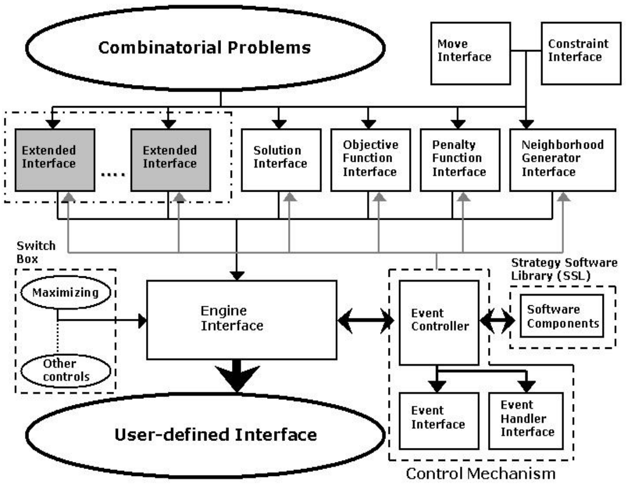

Figure 1: Architecture of MDF

2. 2. Visualization

2. 2. 1. A need for GUI

This project studies the potential of interactive human guides to enhance the quality of a meta-heuristic search algorithm. By combining the power of both worlds: the sheer speed of computing and the impressive human visualization and reasoning capabilities, we hope to achieve results which the computer cannot perform alone. Research in this field has recently shown to be promising [Klau, Lesh, and Mitzenmacher, 2002].

Clearly, in order for human to guide the search (i.e., by altering or adjusting some parameters), he must know what is going on in the search using some form of visualization. Text-based visualization will never suffice for this task, thus a graphical user interface (GUI) was built in the MDF. GUI can also be used for parameter tuning, because as the number of meta-heuristics supported by MDF grows, the overall number of parameters will increase and these parameters may be hard to tune, since they may be conflicting with one another.

Human participation in searching for optimal solutions can be effective when it comes to visual perception. Man has ability that state-of-the-art Artificial Intelligence cannot offer, such as follows:

- Human can see a graphical output and can deduce the underlying information: (E.g. by looking at a picture, one can tell at once the number of circles while complicated algorithm is required for a computer program to perform such a task.)
- Human has common sense.
- Human can make good guesses.

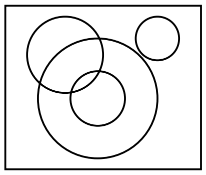
Figure 2: How many circles are there?

A `OpenGL` GUI component has been implemented by the team of NUS students. I extended this generic visualization component and applied it to help the Tabu Search avoid cycling, detect local optimality, and jump to another search space.

2. 2. 2. GUI Functionalities

Users can guide the search by keying in certain keyboard shortcuts. Some of the functionalities to guide the search are listed as follows:

1. **Search speed:** Slowing down the speed would sometimes allow better observation of the search for any repeated pattern. When the user expects the result to improve he should leave the speed as highest. Higher speed also means higher CPU usage.

(The following features have been added in this project)

2. **Save Solution:** Save the state of the solution which produce the best result so far, for the purpose of verification as well as resumption of the search from this point at some other time.
3. **Delete Solution:** The system automatically loads the solution from a file (if the file is found) when it starts, thus deleting solution would mean the next time the system will start with a blank solution.
4. **Empty Tabu List:** When the search is found to be caught in local optimality, emptying the tabu list can help and guide the search to other solution space.
5. **Backtrack:** The user can track the search back to the point where it found the best solution so far.
6. **Visualization On/Off:** For impatient users, turning Visualization off would increase the search speed because rendering graph actually slows down CPU time.
7. **Random discard:** This feature is perculiar to MKP. The system reads in a number *n* and removes *n* number of objects that are currently in the knapsack. This helps move away from the local optimum and explore another search space. This is called ‘diversification’ and it is an essential step in an efficient tabu search.

2. 2. 3. 3D Search Space

MDF is integrated with a generic 3-D search space visualization model. It is intended to visualize the search process and to allow user to observe and guide the search engine. Its graphic representation is based on characteristics found in (almost) all combinatorial optimization problems and meta-heuristics. They are:

a) **Time or number of iteration unit (X-axis)**.
Meta-heuristics are iteration-based algorithms. This dimension allows us to visualize the current solution in each iteration.

b) **Objective value (Y-axis)**
All optimization problems have objective values. We can distinguish which solution is better just by comparing their objective value. However, the first two metrics alone are not sufficient to determine the search space, since there may be many solutions with similar objective value but are very different. Thus we need the third metric below.

c) **Hamming Distance between solutions (Z-axis)**.
I use this metric mainly to gauge the position of current solution in search space compared to the current best solution. This metric is derived from the observation that the search is caught in local optimality when it continuously produces solutions which are similar to one another. However, this third metric is problem-specific and depends on solution representation. For example: in MKP, where each solution is represented by a bit string, the Hamming distance between solutions is then the number of bit difference between two strings.

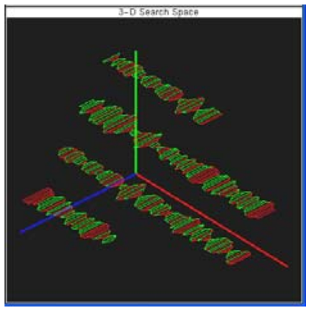

Figure 3: The 3D Search Space

The continuous graph displayed as the search progresses has been split into different levels, as shown in figure 3. This allows the display of a longer range of solutions whose attributes (objective functions and hamming distance from the best solution so far). Furthermore, human eyes decipher the patterns of the graph better if they are more or less fixed in front of them instead of a continuous plot, as it was built previously. We can imagine that the graph has been *folded* into different levels thereby allowing viewers to compare a portion of graph with another more easily. Thus the X-axis can be regarded as being split to different sections with each corresponding to a different level.

Another issue is the scale of graph. Depending on input sizes, the graph displays differently. Since this is a 3D graph, users should be able to choose the angle from which they can observe the change of solution better. Moreover, zooming in and out is also important to allow users to change the scale of values so that the gradient of change is more easily seen. The observation of the graph is thus enhanced with many keyboard keys available to change the view and scale of drawing. For example, Figure 2 shows a change in scale followed by changes in camera angle and zooming scale on the same graph (the search was paused to facilitate better comparison).

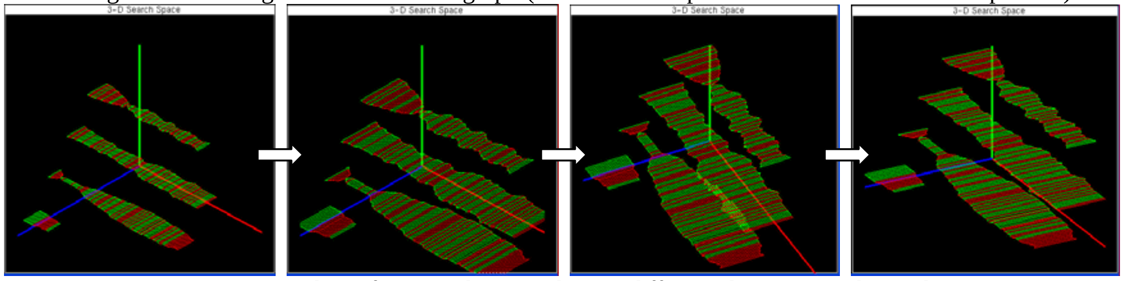

Figure 4: Screenshots of 3D Search Space, showing different adjustments to the graph view.

2. 2. 4. Making use of the 3D Search Space

The fundamental purpose of visualizing search is to allow the human to detect the phenomenon of cycling during search. This is useful since it takes much memory and implementation effort to let the system detect this by itself. By seeing the repetition of any pattern along the graph, the user can be sure that a cycle has occurred and therefore alter the parameters accordingly to get the search out of the cycle, so that there is a possibility of getting a better solution. An example is shown in Figure 3.

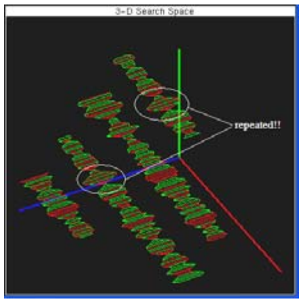

Figure 5: Screenshots edited to indicate repetition

2. 2. 5. Problem-Specific Visualization

Some combinatorial optimization problems have natural visualizations such that the ‘goodness’ of the solutions can be distinguished easily.For example, in the Traveling Salesman Problem (TSP), one can easily compare the ‘goodness’ of TSP solution by one quick look at the visualization. In the example aside (Figure 6), one can quickly verify that the tour on the right is better (shorter) since the one on the left contains a crossing, which is not a characteristic of a good tour.

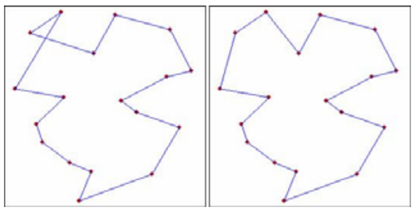

Figure 6: Two TSP tours of the same set of cities

This ideal situation does not apply for some other problems. In MKP, I have not found a representation of the solution as intuitive so far. A good representation means that user can tell whether the solution can be improved, or better still, if he can see *which* is the better solution from the visualization, as in the TSP case. I have chosen to show how much of each capacity of the knapsack that has been occupied. It is useful to show whether objects are taken in a way that maximize the constraints allowed, or there is still a lot of space to make use of in a certain capacity - which is likely to be improvable. A good solution tends not to have a constraint (capacity in this problem) that is far from being filled. Figure 5 shows a not very good solution that have the 2nd capacity (the 2nd bar from the left) fully occupied while the 3rd and 7th ones are far from being occupied. The display is useful in showing changes in the occupation of each capacity when some of the objects are discarded from the knapsack during the guidance of the search (see Random discard in GUI Functionalities above). The display also helps show the correctness of the solution. If any constraint is violated during the running of the search (due to some bugs, for example), it can be detected from the graph.

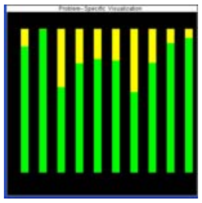

Figure 7: Representation of a solution

3. Results

We benchmarked our proposed system against the input instances available in Operations Research library[1]. Results are shown in Tables 1 and 2.

For small-sized problems, best results were found within a very short time. These are already proven to be the optimal values. Our system obtains the answer very quickly especially if the visualization feature is turned off. For relatively small-sized problems, visualization is not necessary.

270 of the test cases of sizes up to N = 500 and M = 30 are provided by [Chu and Beasley, 1998], all of which do not have proven optimal solutions (N and M are the number of objects and the number of capacities or constraints, see Problem Definition above). There are 30 test cases for each problem size. As it is too time consuming to carry out all the test cases, I have tested on some of the instances of varied size. All the results obtained by running the search for a maximum of 5000 iterations with the user guiding the search. It is possible that the better the results can be obtained by allowing the search to run for more iterations.

It should be noted that the average execution time to get the best known results as presented in [Chu and Beasley, 1998] are between 6 minutes (for the problem size N = 100, M = 5) and 60 minutes (for some problems of size N = 500, M = 30). As this is a human-guided search, it is not necessary to get the timing correct to seconds. In comparison, our results are obtained within a short computation time. They are also very close to the best-published results.

4. Discussion and Conclusion

The results shown were obtained by ourselves and are less likely to be reproduced easily by other users when they first use the system since it takes some familiarity with the Visualizer. However, it is generally intuitive and user-friendly, and as soon as the user is comfortable with the GUI he can be very apt in guiding the search. As our results are close to the best published ones (0.54% on average), we have demonstrated that MDF when used in conjunction with the Visualizer component is a promising combinatorial search strategy.

The user should know the mechanism of tabu search quite well in other to make good use of the system. The most basic concept is that in each iteration a best solution is picked among the generated solutions based on the previous solution without repeating those in the tabu list. Tabu search uses a tabu list where solutions or moves are stored for a designated number of iterations (depending on the Tabu tenure, which is can be changed on the fly). For this particular MKP, we have implemented a more special algorithm that uses two tabu tenures and we *tabu* the objects that have been added or removed after each iteration. The *add* tenure is the number of iterations an object will not be removed after it is put into the knapsack, and conversely, the *remove* tenure determines the number of iterations an object will not be added after it is removed from the knapsack. Therefore, larger tenures mean that a cycle is less likely to occur. However, smaller tenures allow the search to achieve local optimal results faster, until a cycle occurs. Some patterns that we applied in our search strategies and any user should use are to enlarge the tabu tenures once a cycle is detected, to adjust to smaller tabu tenures to improve the solution faster. Other strategies include the use of above-mentioned *Random discard* and *Empty tabu list* (see GUI functionalities) which allow more possible solutions generated and move the search out of a cycle.

The search runs much slower on input with size N = 500 compared to N = 100, even when M is larger for the latter. Table 2 also shows that the gap from the best published results is smaller when N = 100. Thus our search speed is more notably affected by the number of objects, not the number of constraints. In fact, for real life problems, the number of constraints normally does not get very large, but the number of elements to be considered might be beyond the range considered in these inputs. We believe results can be improved within the same amount of time if this system is run on a faster or parallel machine.

Acknowledgements

I would like to express my heartfelt gratitude to my mentor Dr Lau Hoong Chuin, for his invaluable guidance and patience throughout the course of research. I would like to thank Steven Halim and Wan Wee Chong for their help in algorithms and programming skills, without whom this research could have been impossible. Thanks also go to the Ministry of Education, the National University of Singapore, and my school Anglo-Chinese Junior College for giving me the opportunity to carry out this research, in which I had a precious learning experience.

[1] The source is available at http://www.brunel.ac.uk/depts/ma/research/jeb/orlib/files/

Problem size		Best known	Our results	Gap (0%)	CPU seconds
N	M				
6	10	3800	3800	0	1
15	10	4015	4015	0	5
20	10	6120	6120	0	5
28	10	12400	12400	0	60
38	5	10618	10618	0	60
50	5	16537	16537	0	60

Table 1: Results for small-sized problems.

Problem size			Best known	Our results	Gap (%)	CPU minutes
N	M	Name				
100	5	5.100-00	24381	24381	0	5
		5.100-01	24274	24274	0	5
		5.100-02	23551	23551	0	5
		5.100-03	23534	23511	0.09	5
		5.100-28	61520	61520	0	5
		5.100-29	59453	59453	0	5
250	5	5.250-28	155075	154790	0.15	8
		5.250-29	154662	154519	0.09	8
500	5	5.500-0	120130	118062	1.72	18
		5.500-28	302814	302089	0.24	18
		5.500-29	29904	298527	0.45	18
100	10	10.100-0	23064	22827	1.02	5
		10.100-29	60633	60633	0	5
250	10	10.250-0	59187	58142	1.76	9
		10.250-28	149155	148665	0.33	9
		10.250-29	149704	149486	0.14	9
500	10	10.500-0	117726	115499	1.89	20
		10.500-29	307014	304845	0.70	20
100	30	30.100-0	21946	21719	1.03	7
		30.100-29	60603	60465	0.22	7
250	30	30.100-0	56693	56061	1.11	10
		30.250-28	149568	149156	0.27	10
		30.250-29	149572	149050	0.34	10
500	30	30.500-00	115868	114113	1.51	23
		30.500-01	114667	113895	0.67	23
		30.500-29	300460	299494	0.32	23

***The experiments were conducted on a P4 1.80 GHz PC with 256MB RAM.**

Table 2: Results for large-sized problems.

References

1. Lau, H. C, Lim, M. K, Wan, W. C, and Halim, S. A Development Framework for Rapid Meta-heuristics Hybridization. COMPSAC Conference, Hong Kong, 2004.
2. Halim, S. Extending Metaheuristic Development Framework for solving Combinatorial Optimization Problem. Honours Years Project Report, 2004
3. Lau, H. C, Wan, W. C, and Jia, X. A "Generic Object-Oriented Tabu Search Framework", Proc. 5th Metaheuristics *International Conference*, Kyoto, Japan, 2003. Also presented at EURO/INFORMS Joint International Meeting, Istanbul, Turkey.
4. Hanafi S. and Fréville A. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106:659–675, 1998.
5. Garey, M and Johnson, D. Computers and Intractability - A Guide to the Theory of NP-completeness. Readings, Freeman, 1979
6. Blum, C. and Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Survey, 35:3, pp. 268-308, 2003.
7. Ribeiro, C.C, and Hansen, P. Essays and Surveys in Metaheuristics. Readings, Kluwer Academic Publishers. 2002
8. Klau, G.W, Lesh, N, Marks, J, Mitzenmacher, M. Human-Guided Tabu Search. In proceedings of National Conference on Artificial Intelligence, Edmonton, 2002.
9. Chu P.C. and Beasley J.E. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristic, 4:63–86, 1998.
10. Gavish B. and Pirkul H. Allocation of data bases and processors in a distributed computing system. Management of Distributed Data Processing, 31:215–231, 1982.
11. Shih W. A branch & bound method for the multiconstraint zero-one knapsack problem. Journal of the Operational Research Society, 30:369–378, 1979.

12 Hertz, A, Taillard. E, and de Werra, D. A tutorial on Tabu Search. In proceedings of Giornate di Lavoro AIRO'95, 1992.