The effects of different neighborhood generation mechanisms on the performance of Tabu Search

FARHAD KOLAHAN [1], AHMAD TAVAKOLI [2]
[1] Department o f Mechanical Engineering, [2] Department of Management
Ferdowsi University of Mashhad
IRAN
kolahan@um.ac.ir

Abstract: - Tabu Search (TS) is a robust neighborhood search algorithm used to solve a wide range of combinatorial optimization problems. In this paper, the effects of six neighborhood generation and move selection mechanisms on the performance of TS are investigated. Among these strategies is a newly proposed dynamic neighborhood which is shown to be efficient in solving the problem under consideration. To compare the performance of these strategies, a set of constrained Traveling Salesman Problems (TSP) is solved using different neighborhoods. Computational results are then compared in terms of solution qualities and convergence speeds. The results show that Tabu Search performance is greatly affected by the neighborhood generation mechanism and move selection policy.

Key-Words: - Optimization algorithms, Tabu Search, Neighborhood generation, move strategies, TSP

1 Introduction

In recent years, with the advent of complex industrial systems, process and system optimization have become a major research area. Many organizations are faced with different challenges, such as resource constraints, severe competition, rapid market changes, and environmental limitations. Because of these issues many optimization problems arise in modern industries. Nevertheless, real sized optimization problems are usually large and complicated. Therefore, solving such problems by traditional deterministic methods is very time consuming and inefficient. On the other hand, simplification of such complex problems usually makes their results inaccurate and unsuitable for real life situations.

In response to such needs, heuristic algorithms such as Tabu Search, Simulated Annealing and Genetic Algorithm, with high calculation speed and good (not necessarily optimum) solution quality, seem to be good alternatives for optimization procedures. Among many advantages of these techniques, is their good adoption to wide range of optimization problems [1,2]. These methods are working within feasible solutions space and are not dependent on the problem structure. Thus, as long as the decision variables are discrete or they can be defined as discrete variables, complicated problems can be coded and solved by these methods. This is done by converting each solution into a numerical string of decision variables. However, even after adjusting the algorithm on problem structure, parameters selection has a great effect on run times and the quality of final solutions.

In some research works, the performances of some neighborhood search heuristics are compared using different standard benchmark problems. Nevertheless, such comparisons may not be fair and exact, since with proper search parameter settings most techniques may provide good answers. In other words, the performance of any algorithm is affected by the way that its parameters have been tuned and with proper parameter tuning, many techniques may work well on a given problem.

In this paper, however, a different approach is taken. Our objective is to evaluate the effects of various neighborhood generation mechanisms and move selections policies on the performance of Tabu Search (TS). To achieve this goal, first different neighborhood generation mechanisms and move selection policies in TS are presented. Along this line, we also propose a dynamic neighborhood mechanism that adjusts the size of neighborhood during the search. This new approach can adjust the size of neighborhood based on the search progress. Then, we employ a set of TSP problems-one of the most well known and widely used benchmark problems-to investigate the effects of different neighborhood generation and move selection policies. Finally, search results are compared and discussed in terms of computational speed and solutions qualities

2 The heuristic: Tabu Search

During the last two decades, with the advent of computer capabilities, many heuristic algorithms are proposed to solve large and complicated optimization problems [3]. Generally, most of these heuristics are inspired by the natural and physical phenomena. These techniques, to some extend, are the simulation of such phenomenon by mathematical functions.

Although, most of these methods do not guarantee optimum solutions, they try to find an optimum or near optimum solutions by searching the feasible solution space of the problem. That is why they are called "neighborhood search methods". Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS) are some well known such algorithms.

Tabu Search is a robust neighborhood algorithm. This method which was first proposed by Glover [4], is an iterative neighborhood search that step by step, searches feasible solution space to find the optimum or near optimum solution. In this method, search begins from a feasible solution and for each move, the neighborhood of current solution, is generated and evaluated. Then a new move is made to the best allowable (non-tabu) answer in the neighborhood. The stepwise transition from one solution to another allows the search to reach an optimal or a close-to-optimal solution after a number of moves. However, a single move, by itself, may not necessarily improve the current value of objective function. This distinguishes tabu search from other traditional techniques such as hill climbing that require each move to be an improving step. Throughout the search, the best solution found so far, *Cbest*, and its corresponding sequence, *Sbest*, will be recorded and updated. The important parameters in TS are as follows:

Starting Point: Search begins from a feasible solution such as *S*. A feasible solution is any set of values for decision variables that satisfies problem specifications and constraints.

Neighborhood: For a given solution *S*, the neighborhood *N(S)* is a set of feasible solution generated with the minimum changes in current solution. Pairwise interchange is the most popular mechanism for neighborhood generation. In this mechanism, each neighbor is generated by changing the position of two members in the current solution string.

Move: A move is the transition from the best solution, s^{**}, in the previous neighborhood to the best permissible solution, s^{*}, in the current neighborhood. The fitness of each solution is calculated and compared using the objective function of the problem. Such a move, however, may or may not be an improving one.

Tabu List: One of the important features of tabu search is its ability to avoid being trapped in local optima by constructing a list of tabu moves. Tabu list, *T_list*, includes a certain number (*T_size*) of previous moves which are not allowed at the current iteration. Once a move from s^{**} to s^{*} is made, s^{**} is stacked to the top of tabu list and the oldest member of the list is removed. Thus, returning back to this s^{**} is forbidden for the next *T_size* iterations. This can exclude, to some extent, those moves which lead to possible cycling. The size of tabu list can affect the search performance. Although a longer list may prevent cycling, it requires more scanning and may limit the search domain. The best tabu list size appears to be problem dependent.

Termination Criteria: The last element necessary for tabu search is termination criterion. In general, search can be stopped when a certain number of iterations, *Mmax*, is completed, after a pre-defined of computational time, *Tmax*, is reached, or when no improvement is obtained in a specific number of moves.

A full explanation of this technique and some of its applications can be found in [5,6,7,8]

3 Neighborhood generation and move selection mechanisms

Generally, parameters setting have a considerable impact on in heuristic algorithms performances. In many cases, it takes many trial runs to find suitable values for search parameters. This, in turn, makes optimization process a time consuming task. Fortunately, in comparison to the similar methods, TS is a robust algorithm with few parameters.

Neighborhood generation mechanism and move selection policy are the most important parameters that influence TS performance in terms of solution quality and computational time. Conventionally, a neighborhood, *N(S)*, is defined as a set of solutions that can be obtained by performing one transition in the current solution. In the related literature, pairwise interchange is the most widely used technique to make such a transition. In this method, a solution is generated by switching the members (cities in our case) in positions i and j. The complete pairwise interchanges of a J-city problem leads to $[N(s)] = J(J - 1)/2$ neighbors.

Extraction and reinsertion is another technique for transition. With this method, the neighborhood contains all solutions obtained by extracting the city

in position *i* and inserting it right after (or before) the city in position *j*. The neighborhood size for the extraction and reinsertion approach is *[N(s)] = J(J - 1)2* which is almost doubled as compared to pairwise interchange. Thus this mechanism appears to be more computationally demanding. Furthermore, as indicated by Adenso-Diaz [9], none of these two techniques seems to outperform the other in terms of solution quality for a given run time. Therefore, in this paper only the pairwise interchange approach is used as the basic neighborhood generation mechanism.

The next step in tabu search is to specify a move strategy. The classic approach is to evaluate the entire neighborhood and choose the best allowable move. However, the required computational time could be unacceptably long when the problem size is large. To overcome this problem, partial or random search schemes have also been proposed [10].

In following, six different neighborhood generation and move selection mechanisms are discussed. These policies are then evaluated using a numerical example of 100 cities.

Pairwise Interchange of all neighbors (PI): This is the most common police in which all neighbors of the current solution are generated and evaluated at each iteration. For a problem with J variables (cities), the transposition range (the range of interchange) would be from 1 to *(J-1)*. The neighborhood size for PI policy is $|N(s)| = J(J-1)/2$.

This number of solutions should be generated and evaluated for each move. This policy guarantees complete neighborhood search and results in a better solution quality. However, as problem size grows, generation and evaluation of all neighbors can be very time consuming and would cause sluggish convergence. To increase the search speed, partial neighborhood can be generated and evaluated.

Partial Neighborhood (PN): The range of interchange in PN is from 1 to *m (m < (J-1))*. In other word, neighborhood is generated by interchanging the position of each city with a limited number of next cities. Indeed, this will lead to a smaller neighborhood and will increase search speed. The number of neighbors for this case is :

$$|N(s)| = m(J-m) + \sum_{i=1}^{m}(i-1).$$

First Improving Neighborhood (FIN): In FIN policy, neighbors for current solution are generated one at the time and once an improving neighbor is found the process of neighborhood generation will stop. If no improving neighbor is found, the entire neighborhood is generated and move is made to the best one; just like PI. Therefore, the size of neighborhood is not constant and can vary from 1 to $|N(s)| = J(J-1)/2$.

Random Neighborhood (RN): This mechanism is similar to PN in terms of neighborhood size, but has random nature. In this method, m cities from *J-1* possible cities are selected randomly and interchange will be done on them.

Adjacent Pairwise Interchange (API): This is a special case of PN method, in which transposition range is only 1. This mechanism generates the smallest neighborhood size.[1] The number of neighbors in each move is: $|N(s)| = (J-1)$.

Dynamic Neighborhood (DN): This policy aims at taking advantage of both API and simple PI mechanisms by dynamically changing the transposition range of variables (cities) in the solution string (tour). The idea is to guild the search as fast as possible towards unsearched areas where the optimum solution may be located and then find such solution by complete evaluation of entire neighborhood. To achieve this, at the early stages, only API is used to speed up the search process. As the search progresses, the neighborhood size is enlarged by increasing the range of transposition. This in turn, improves the solution quality at each iteration. Finally, in final iterations, complete PI interchange is employed to generate and evaluate entire neighborhood. To this end, we proposed the following linear function to determine the transposition range at each iteration:

$$TR(n) = \begin{cases} 1 & \text{if} \quad Z \le 1 \\ Z & \text{if} \quad 1 < Z < J-1 \\ J-1 & \text{if} \quad Z \ge J-1 \end{cases}$$

Where: $Z = X + Y\,n$

In the above equation *TR(n)* is the upper limit of transposition range at *n*th iteration, and *X* and *Y* are the constant values determined experimentally. The size of neighborhood is controlled by *Z*. The values of $Z \le 1$ and $Z \ge J-1$ refer to the transposition ranges of API and PI respectively. Since the value of *Z* is a function of number of moves, *n*, as search progresses the neighborhood size also grows so that a through search can be perform at final iterations.

4 Test problem for Tabu Search

To evaluate the performance of TS under different neighborhood generation and move selection mechanisms, the algorithm has been applied to a set of constraint Traveling Salesman Problem (TSP) problems. TSP is one of the most studied optimization problems, [11,12,13], in which a salesperson has to visit all interconnected cities only once in such way that the total traveling distance is minimized. In constrained TSP, there is no direct connection between some cities. A schematic example of constrained TSP is shown in Figure 1 in which all 5 nodes are interconnected but there is no direct connection between nodes C and D. Each solution (tour) is represented by a sequence of cities such as A,B,C,E,D. For a problem of size J, there is a total of $J!$ possible sequences or tours. The cost function is the sum of total crossed distances in a given tour. It should be noted that in constrained TSP some tours may be infeasible due to the absence of direct connection between two consecutive cities (e.g. A,B,C,D,E,).

The search starts with a feasible tour or sequence of cities, say S. Then the neighborhood $N(S)$, is generated by performing one transition in the current solution; i.e. pairwise interchange. Each neighbor in $N(s)$ has its associated objective function value or total distance $G(s)$, and the one with the smallest $G(s)$ is defined as the best neighbor, denoted by s^*. A move is then made from s^{**}, the best sequence of the immediate previous neighborhood, to s^*, provided that s^* is not in the current tabu list. The best solution found so far, and its corresponding sequence are then updated if necessary and kept in memory. The sequence s^* is then stacked into the tabu list of pre-defined size and the oldest sequence is removed from the list. The search is stopped when termination criterion is met.

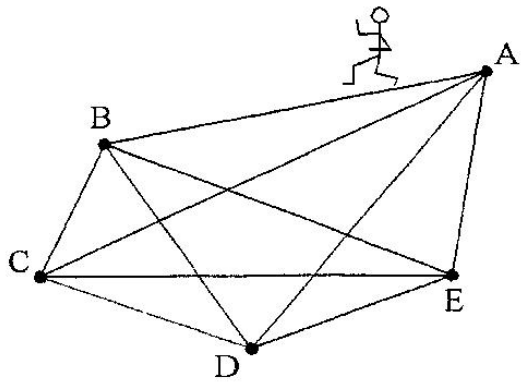

Fig. 1, A representation of constrained TSP

5 Numerical example and results

For any neighborhood search procedure, the size of neighborhood and the way that the next move is selected has a great effect on search performance. This becomes more evident when the problem size is large.

For comparison, the computations were carried out on a set of TSP problems with 100 cities (variables). It should be noted that for a problem with 100 cities, the number of possible solutions (tours) is 100!; and hence using an efficient solution procedure is inevitable. To make the optimization problem more realistic, constrained TSP has been used. This is a more practical version of TSP that can be implemented to many real life optimization problems [14,15], and hence considered in this research as the benchmark problem.

The algorithm was coded on MATLAB R6.5 software and run on a Pentium 4 computer. For more accurate and fair comparison, in all runs the arrangement of cities, the termination criterion and the starting points are kept the same.

The algorithm was run for each of the six strategies and the results for the best search parameters are presented in Table 1.

Table 1. Results of TSP example with 100 cities

Neighborhood Mechanism	Initial distance	Final distance	Objective improvement (%)	Neighborhood size $\lvert N(s)\rvert$
PI	42743	16160	164.5	4950
RN		16120	165.1	2535
PN		16980	151.7	2535
API		25550	67.3	99
FIN		16020	166.8	~
DN		16160	164.5	~

As mentioned before, starting points and tabu list size for all policies are kept the same. The neighborhood sizes are also shown in Table 1. It is noted that for FIN and DN strategies the size of neighborhood can be determined due to their dynamic behaviors.

The results in Table 1 show considerable objective function (distances) improvements for all policies during 30 seconds of search time. The least improvement in the objective function is more than 67% while there is about 167% reduction for the

best scenario. Nevertheless, there is a big difference in terms of search performance under different policies. This illustrates the importance of neighborhood generation and move selection policies on the search effectiveness.

The convergence curves for different strategies are also shown in Figure 2. As shown in Figure 2, all mechanisms (excluding API) converge to the similar answer. Although API neighborhood demonstrates a fast convergence rate at the beginning, it performs inferior in terms of solution quality. The fast convergence rate of API is a result of its small neighborhood size; whereas the same feature probably causes it being trapped in local optima. Other strategies converge to similar solutions. However, among five other mechanisms, FIN and DN strategies seem to be the best as they demonstrate both superior convergence rate and solution quality. They both somehow use dynamic neighborhoods that allow the algorithm to move fast towards promising parts of solution space. They are also capable of doing a complete neighborhood search at the final stages where possible optimum is near by. This reduces the computational time at the beginning of the search while improves solution quality at final iterations. Nevertheless, since DN method employs a guiding function to determine transposition range, search movement under this scenario is more controlled. This feature results in a slightly better performance for this method than FIN policy.

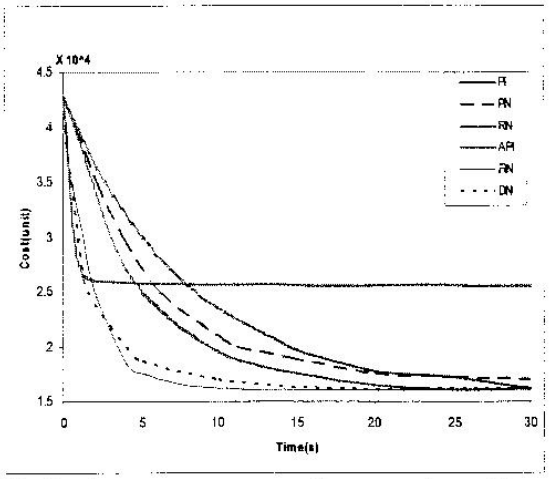

Fig. 2 Convergence curves for example problem using different neighborhoods

Amongst various neighborhoods PI, PN, and RN mechanisms demonstrate somehow similar patterns. Relative dullness of PI mechanism is because of its neighborhood size and move selection policy that requires generating and evaluation of all neighbors for each move. On the other hand, evaluating all neighbors will increase the probability of finding an optimum solution. Although in the example problem the neighborhood sizes of RN and PN are the same, RN clearly outperforms PN. The random nature of search in RN, that extends the search area, is the main reason for this advantage. This is somehow similar to what happens in Simulated Annealing, another powerful probabilistic neighborhood search.

6 Discussions and Conclusions

Many neighborhood search algorithms have evolved during last few decades. Among these, Tabu Search is an efficient and robust technique. With its few search parameters, it requires little parameter tuning. Neighborhood generation and move selection policies are indeed some the most important parameters for any neighborhood search heuristics, including TS. In this research, the effects of different neighborhood generation and move selection mechanisms on search efficiency were analyzed. Computational results have shown that in the longer run times, most neighborhood mechanisms result in similar solutions. However, the rate of improvements largely depends on the type of neighborhood. This becomes more evident for shorter computational times and for larger problem sizes. It seems dynamic neighborhoods such as DN and FIN lead to better solutions in shorter search times. This is mostly because of the part of neighborhood to be evaluated varies in different phases of the search. In the beginning, smaller neighborhood is usually evaluated allowing the search to expand search domain. After several iterations when the algorithm converges to promising part of solution space, bigger neighborhoods are generated and evaluated to increase solution quality. Therefore, it may be beneficial to use different neighborhood mechanisms at different stages of the search. Due to more regulated neighborhood generation procedure in DN, it generally performed better than FIN; although the difference was not so large in our example problem.

Indeed, we can not extend the results find here to all optimization problems. However, it is clearly shown that the performance of Tabu Search, and any other neighborhood search for this matter, is greatly affected by its neighborhood generation and move selection mechanisms. This could be a promising area for future research works.

References:

[1] M. Amen, Heuristic methods for cost-oriented assembly line balancing: A survey, *International Journal of Production Economics*, Vol. 68, No.1, 2000, pp. 1-14.

[2] M. Gen and Y. Su Yun, Soft computing approach for reliability optimization: State-of-the-art survey, *Reliability Engineering & System Safety*, Vol. 91, No. 9, 2006, pp. 1008-1026.

[3] G. Andreatta,, L. Brunetta and G. Guastalla, The flow management problem: recent computational algorithms, *Control Engineering Practice*, Vol. 6, No. 6, 1998, pp. 727-733.

[4] F. Glover. Tabu Search – a tutorial, *Interfaces*, Vol. 20, 1990, pp. 74-94.

[5] F. Kolahan, M.Liang, An adaptive TS approach JIT sequencing with variable processing times and sequence-dependent setups, *European Journal of Operations Research*, No. 109, 1998, pp. 142-159.

[6] D.L. Woodruff, Simulated annealing and tabu search: Lessons from a line search, *Computers and Operations* Research, Vol. 41, No. 8, 1994, pp. 829-831.

[7] A. Hertz, D. De Werra, The tabu search metaheurestic: how we used it, *Mathematics in Artificial Intel*, Vol. 1, 1991, pp 111–121.

[8] U. Al-Turki, C. Fedjki, and A. Andijani, Tabu search for a *class* of single-machine scheduling problems, *Computers & Operations Research*, Vol. 28, 2001, pp. 1223-1230.

[9] B. Adenso-Diaz, Restricted neighborhood in the tabu *search* for the flowshop problem, *European Journal of Operational Research*, Vol. 62, 1992, pp. 27-37.

[10] F. Della Croce, Generalized pairwise interchanges and machine scheduling. *European Journal of Operations Research*, Vol. 83, 1995, pp 310-319.

[11] V. Dimitrijevi, and Z. Sari, An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs, *Information Sciences*, Vol. 102, No. 14, 1997, pp. 105-110.

[12] K. Katayama, H. Sakamoto and H. Narihisa, The efficiency of hybrid mutation genetic algorithm for the traveling salesman problem, *Mathematical and Computer Modeling*, Vol. 31, No. 12, 2000, pp. 197-203

[13] B. Golden, L. Bodin, and T. Doyle, Approximation traveling salesman algorithm, *Operations Research*, No. 28, 1980, pp. 694-712.

[14] F. Kolahan, and M. Liang, Optimization of hole-making operations: a tabu-search approach, *International Journal of Machine Tools and Manufacture*, Vol. 40, No. 12, 2000, pp. 1735-1753

[15] J. Hurink and S. Knust, Tabu search algorithms for job-shop problems with a single transport robot, *European J. of Operational Research*, No. 162, 2005, pp. 99 –111