Introduction to Tabu Search

Charles Nicholson
March 2017

School of Industrial and Systems Engineering, University of Oklahoma, Norman, OK

Abstract

A basic introduction to the Tabu Search algorithm written for the DSA/ISE 5113 course at the Univeristy of Oklahoma. Tabu Search is a powerful metaheuristic that has stood the test of time in terms of its effectiveness in wide application areas. Furthermore, the basic technique is relatively simple but can be extended with a variety of parameters and concepts. One of the creators of this technique also developed "path relinking" which will be introduced at the end of the article. Path relinking can be combined with Tabu Search or with many other metaheuristics.

1. Tabu Search

The core idea of Tabu Search (TS) was developed independently by two researchers in 1986:

> Fred Glover (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, **13**(5):533-549
>
> P. Hansen (1986). The steepest ascent mildest descent heuristic for combinatorial programming. In Proceedings of the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, 1986.

The term tabu, however, was coined by Fred Glover: "The overall approach is to avoid entrapment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence tabu)."

Similar to both Simulated Annealing and Guided Local Search, Tabu Search allows moves to non-improving solutions. This is done to potentially escape local optimum. However, TS is mostly a deterministic search procedure which also selects the best solution option available and this combination of characteristics can cause solution cycling. For example, if TS is at a solution x and moves to a non-improving solution $y \in N(x)$ (since there are no better moves in the neighborhood $N(x)$), then once it is there, as it evaluates the new neighborhood $N(y)$, it is quite possible that it will make its next move back to $x \in N(y)$ since $f(x)$ is better than $f(y)$. At x again, since TS is mostly deterministic, it will move back to solution y. And this cycling would continue for the duration of the iterations. The tabu status of solutions is setup to stop this from occurring.

> **Tabu:** moves to some solutions (and equivalently, the solutions themselves as subsets of the solution space) are temporarily prohibited; this includes moves to solutions which have been recently visited.

Tabu status is based on various *attributes* of recent solutions (or moves made to recent solutions). More on this later. For now, note that the idea of "recent" implies that TS has some sort of memory of its own search history. In fact, TS has two main types of memory that are commonly used: *short-term memory* and *long-term memory*.

1.1. Short-term memory

Short-term memory essentially records solution attributes that have changed "recently". The pre-defined tabu criteria is used to determine which types of attributes or moves will be prohibited.

> **Tabu-criteria:** Pre-defined rules to identify solution attributes that will cause part of the search space to be tabu.

The selected attributes in recently visited solutions become tabu-active. Solutions containing these attributes are classified as tabu. Tabu solutions are excluded from the neighborhood and not revisited during the tabu tenure.

> **Tabu-active:** Solution attributes that meet the tabu-criteria; if one or more attributes in a solution are tabu-active, then the solution is tabu.
>
> **Tabu tenure:** The length of time (usually in terms of algorithm iterations) that a given attribute is tabu-active; once this time expires, the attribute is no longer tabu-active.

Short-term memory is a key element of TS since it mitigates solution cycling. The short-term memory along with the tabu criteria effectively create a reduced neighborhood $N^*(x) \subseteq N(x)$ which is used for possible moves. Note that the reduced neighborhood (and temporarily modified search space) created by the tabu-active attributes is not limited to only solutions which have been visited, but in fact may prohibit a much larger area of the search space. Any solution with any any tabu-active attribute has a tabu status.

Short-term memory is also sometimes called recency-based memory. Once the tabu tenure for an attribute has expired, this attribute is no longer tabu-active and is essentially lost from the short-term memory.

1.2. Long-term memory

Long-term memory in TS keeps track of information from the entire search history. Long-term memory is also called frequency-based memory. It is not a detailed history however, but the history is aggregated in terms of frequency counts. One type of long-term memory is a "transition measure" which records the number of iterations that a given attribute has been changed (e.g., added or deleted from a solution). For example, the number of times that city i has been moved to an earlier position in a sequence in the Traveling Salesman Problem (TSP). Another type of long-term memory is a "residence measure" which records the number of iterations in which an attribute has stayed in a particular position (e.g., belonging to the current solution). For example, in the knapsack problem, the number of times in which object i has been included in the knapsack.

Long-term memory is used in TS to encourage or discourage exploration of certain solutions. For instance, if, after searching for many iterations, it is observed that the evaluation function is more likely to produce good values when element i is included in the solution (or arranged in a particular location within the solution), then modifications can be made to further encourage the algorithm

to include element i in its solution. On the other hand, if it is observed that element i is mostly associated with poor solutions, we can discourage TS from including element i. Such guidance is accomplished through an *extended evaluation or cost function* similar to what is used in Guided Local Search.

1.3. Aspiration criteria

The tabu status of a solution can be overruled for a preferable alternative using what is known as *aspiration criteria*. One basic example of an aspiration criteria is called the "best solution criteria": If a tabu solution encountered at the current iteration is better than the best solution found so far, then its tabu status is overridden.

Aspiration criteria: Rules for overriding the tabu status of a move or solution.

Other aspiration criteria might be created based on the degree of change or the degree of feasibility of a solution. Or possibly some ratio of evaluation function value and the feasibility level. Another possibility is when all neighbors of a given solution are tabu – you could create a rule for this specific case to randomly choose one of the tabu solutions and move to it.

1.4. Tabu Search – high level algorithm description

A very high-level description of the Tabu Search algorithm is presented below in pseudo code. Here, I've assumed that all of the preliminaries such as the neighborhood definition, tabu criteria, tabu tenure, aspiration criteria, etc. have been defined. Obviously, there are a lot more details that could be included in the pseudo code and would certainly have to be included in an implementation, but this provides an overview of the algorithm.

Algorithm 1: Tabu Search

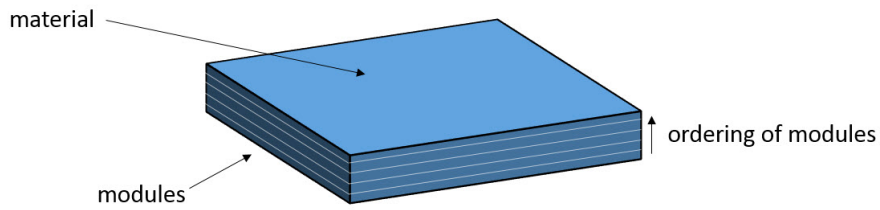
```
1 begin
2     x ← a starting solution
3     Initialize tabu memory
4     while stopping criterion not met do
5         Find a list of candidate moves from N*(x) ∈ N(x)
6         Select solution s ∈ N*(x) with best extended evaluation function value
7         Update tabu memory and perform the move: x ← s
8     end
9 end
```

2. Tabu Search example

Note: The following example and much of the text are taken directly from the Tabu Search chapter in *Modern Heuristic Techniques for Combinatorial Problems*, Colin R. Reeves (Ed.), 70-150, Blackwell Scientific Publications, Oxford, 1993. The chapter was written by Fred Glover and Manuel Laguna.

Permutation problems are an important class of problems in optimization, and offer a useful vehicle to demonstrate some of the considerations that must be faced in the combinatorial domain. Classical instances of permutation problems include traveling salesman problems, quadratic assignment

problems, production sequencing problems, and a variety of design problems. As a basis for illustration, consider the problem of designing a material consisting of a number of insulating modules. The order in which these modules are arranged determine the overall insulating property of the resulting material.



The problem is to find the ordering of modules that maximizes the overall insulating property of the composite material. Suppose that 7 modules are considered for a particular material, and that evaluating the overall insulating property of a particular ordering is a computationally expensive procedure. We desire a search method that is able to fmd an optimal or near-optimal solution by examining only a small subset of the total possible permutations (in this case, numbering 5040, though for many applications the number can be astronomical).

A representation of a 7 module solution in which module 2 comes first, followed by module 5, then 7, then 3, 4, 6, and lastly 1, could look like the following:

2573461

We can define the neighborhood of a solution based on swapping the position of two modules, for example, modules 4 and 4:

2473561

Any solution would then have $\binom{7}{2} = 21$ neighbors.

For this example assume the tabu attributes are selected as the most recently made swaps and the tabu tenure is set to 3 iterations. Hence, solutions involving the 3 most recent swaps will be classified as tabu. Finally, the aspiration criterion is the best solution criteria. For this example, we will not worry with long-term memory.

The short-term memory structures needs to be able to track which swaps have been made and how many iterations are left in the tabu tenure. Consider the upper triangle of a 7x7 matrix in which each position will record the remaining tabu tenure. For example, if modules 2 and 5 have been swapped the effective tabu tenure would be recorded in the position noted in Figure 1.

Let's consider the starting solution of 2573461 and a few iterations of a basic TS algorithm. The starting solution should be evaluated (let's say the insulation rating of this permutation is 10) and the tabu memory structure should be initialized to all zeros (I'll just use blanks for zeros). Consider the best 5 neighbors. Their relative values (a positive value is how much the swap would improve the solution, a negative value is how much the swap would decrease the solution) are listed in Figure 2.

Since swapping modules 5 and 4 is the best option of all neighbors this move is made and the insulation value will increase to 16.

Now in iteration 1 (see Figure 3), the tabu memory structure is updated. A value of 3 (the tabu tenure) is assigned to the associated spot in the tabu memory. Next, the top 5 candidate neighbors

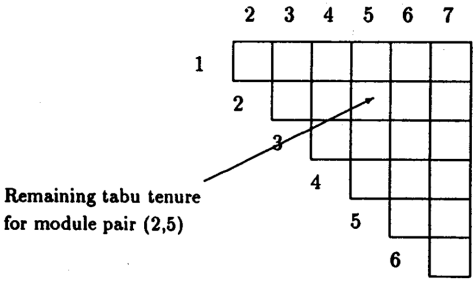


Figure 1: Upper triangle memory structure

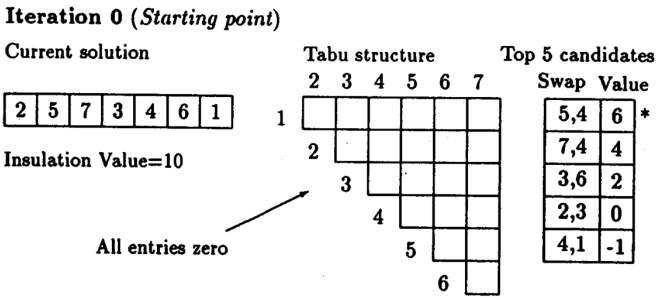


Figure 2: Iteration 0

of the current solution are listed and the swap of modules 3 and 1 is the best option which will improve the insulation to a value of 18.

In iteration 2 (see Figure 4), the tabu memory structure is updated so that the (4,5) swap tabu tenure has been decremented to 2 (there are only 2 more iterations in which this attribute will be tabu-active) and a value of 3 has been assigned to the (3,1) swap. Among the 5 best neighbors of the current solution 2471563, none are improving. Furthermore, two of the neighboring solutions are tabu (denoted with the "T"). The best possible move is a non-improving move made by swapping modules 2 and 4 and will result in a decreased insulation value of 14.

In the third iteration (see Figure 5), the tabu memory is updated and the neighbors are evaluated. Here, there are two improving options although the best swap will lead to a tabu solution. That is swap (4,5) is still tabu active. However, if this swap is made, it will produce an insulation value of 20. No solution so far has produced such a high insulation value. Therefore, the aspiration criteria

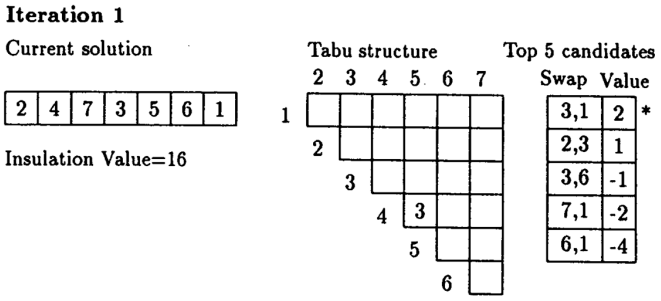


Figure 3: Iteration 1

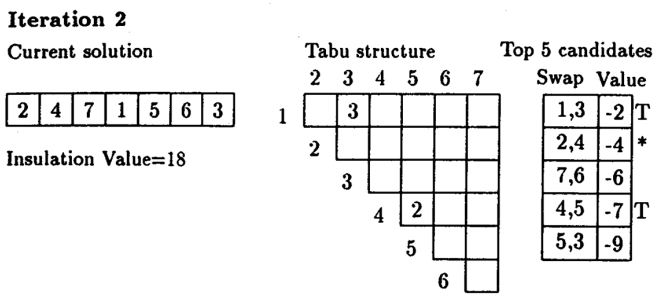


Figure 4: Iteration 2

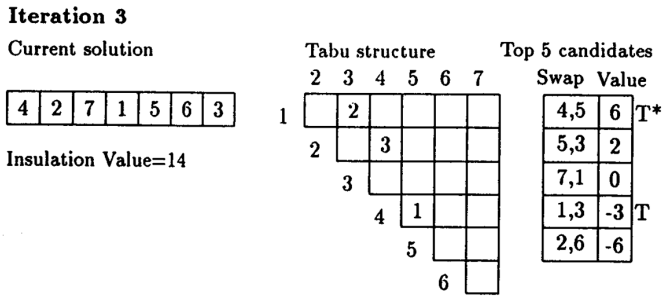


Figure 5: Iteration 3

is used to overule the tabu status. The swap (4,5) is made leading to Figure 6.

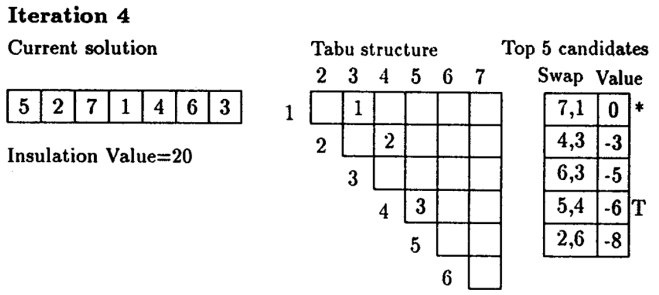


Figure 6: Iteration 4

3. Enhancements to tabu search

3.1. Tabu tenure management

In the example above we assumed the tabu tenure was static. That is, that the value of tabu tenure is fixed and remains fixed during the entire search and all attributes remain tabu-active for the same number of iterations. However, the tabu tenure can be managed dynamically: tabu tenure does not have to be constant throughout the entire search.

With dynamic tabu tenure, TS can be designed to change the tabu tenure at certain intervals. This forms the basis for "reactive tabu search" introduced by Battiti and Tecchiolli in 1994. Once

stagnation is detected (i.e., the solution is not getting changing), the tabu tenure is increased. However, when this area of the solution space has been escaped, the tabu tenure can be reduced again.

3.2. Extended evaluation function and long-term memory

As mentioned above the long-term memory can be used to help incentivize certain solution aspects through the use of an extended evaluation function. In a broader context, the extended evaluation function and long-term memory features can be used to balance *diversification* and *intensification* in the tabu search.

Basic Tabu Search often gets stuck in one area of the search space – diversification is essentially just trying to get to somewhere else. Frequency-based diversification penalizes elements of the solution that have appeared in many other visited solutions.

Intensification is used to more aggressively prioritize good solution attributes in solutions.

If we consider the module ordering problem again and let the function $f(x)$ denote the insulation value of solution x. If we are interested in diversification and considering a solution in which module i is in position j. Let ω_{ij} denote the frequency associated with the number of visited solutions where module i was in position j. The extended evaluation function might be

$$g_1(x) = f(x) - \alpha\omega_{ij}$$

where α is a weighting factor. If ω_{ij} is large, then this solution is much less appealing.

On the other hand, if we are interested in intensification and from the long-term memory we know that whenever module i is in position j, the solutions are frequently the top solutions, the extended evaluation function might be

$$g_2(x) = f(x) + \beta\omega_{ij}$$

where β is a weighting factor. Both of these functions g_1 and g_2 might be used during the same TS implementation. Diversification often is useful during the earlier stages of a metaheuristic, whereas intensification might be more useful in the later iterations.

3.3. Strategic oscillation

Strategic oscillation allows a metaheuristic to oscillate between feasible and infeasible space. In particular, the idea is that searching the boundary region of feasible solutions that are good with respect to the constraints and infeasible solutions that are good with respect to the objective function might be a good place to explore.

That is, if the search moves toward the feasible boundary, instead of stopping it in feasible space (e.g., no infeasible solutions are included in the neighborhood or the evaluation function of infeasible solutions produces very poor values), strategic oscillation would permit the search to cross this barrier and even possibly encourage it.

This can be implemented through a temporary modification of the evaluation function. For example, in the knapsack problem, normally you might evaluate a solution based on both its value and feasibility. For instance, if it the solution is infeasible, the evaluation function incorporates a large penalty. However, after a few iterations in infeasible space, the penalty might be reintroduced.

One reason that the search should be allowed to visit infeasible space is that the feasible space might be fragmented. That is, given a specific type of neighborhood operator, there simply is no path between feasible solutions s_A and s_B. Additionally, optimal solutions often lie near the boundaries of infeasibility: in such cases, there are infeasible solutions for which the optimal solution is a neighbor.

Stragetic oscilliation has been successeful in many application areas and implemented in a variety of ways as seen in Table I.

Table I. Guidance by Strategic Oscillation

Application	Element Controlled	Reference
Quadratic Assignment	Tabu Restrictions	Battiti and Tecchiolli[2]
Transportation	Infeasibility Penalties	Cao and Uebe[3]
Operation Timetables	Penalty Measures	Costa[4]
Multidimensional Knapsack	Infeasibility Depth	Freville and Plateau[11]
Vehicle Routing	Nodes and Infeasibility	Gendreau, Hertz and Laporte[12]
Employee Scheduling	Number of Employees	Glover and McMillan[21]
Network Design	Platform (node) assignment	Glover, Lee and Ryan[20]
Data Integrity	Verification Infeasibility	Kelly, Golden and Assad[32]
Multilevel GAP	Assignments/Infeasibility	Laguna et al.[32]
Classroom Scheduling	Assignments/Infeasibility	Mooney and Rardin[38]
Vehicle Routing	Objective Function	Osman[43]
Capacitated Clustering	Objective Function	Osman and Christofides[44]
Mixed Fleet VRP	Vehicles and Infeasibility	Osman and Salhi[45]
Delivery Systems	Penalty Measures	Rochat and Semet[47]
Graph Partitioning	Partitioned Nodes	Rolland, Pirkul and Glover[48]
Time Deadline VRP	Infeasibility	Thangiah et al.[54]
Graph Design	Objective Function	Verdejo, Cunquero and Sarli[55]
P-Median Problem	Values of P	Voss[56]
Traveling Purchaser Problem	Markets	Voss[57]

4. Path relinking

4.1. Background

Tabu search, scatter search (SS), and path relinking (PR) were all developed by Fred Glover and have a close relationship with each other. Originally, SS was one of the basic processes inherent to TS. Also, scatter search is a specific case of path relinking. As such, we will only discuss PR here. And while it is often combined with tabu search, PR can be combined with any number of metaheuristics. Therefore, we will consider path relinking as something separate from tabu search.

One note about scatter search and path relinking. Both searches can be thought of as population-based metaheuristics. They some multiple characteristics that resemble genetic algorithms (GA), namely in how existing solutions are combined to create new solutions. However, while GA is inherently a stochastic search procedure, SS and PR can be deterministic or include randomized elements.

For an excellent article detailing both SS and PR, please see:

> Glover, F., M. Laguna, and R. Marti (2000). Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics* **29**(3):653-684.

4.2. Path relinking

Path relinking generates new solutions by exploring trajectories that connect elite solutions.

During a search process, e.g., tabu search, across the various iterations, you are likely to discover several good solutions. Let this set of good solutions be defined as the "elite solutions". This set will likely be relatively smalle (e.g., top 2 or solutions found so far.)

The path relinking exploration starts from an initiating solution from within this elite set and generates a "path" in the neighborhood space that leads to a guiding solution (which is also within the elite set). Glover et al. (2000) explains this using the graphical depiction in Figure 7. Assume solutions x' and x'' represent two elite solutions found through a metaheuristic search such as tabu search. The original path from x' to x'' is shown by the solid black line. Each of the six solutions along this path were visited during the process.

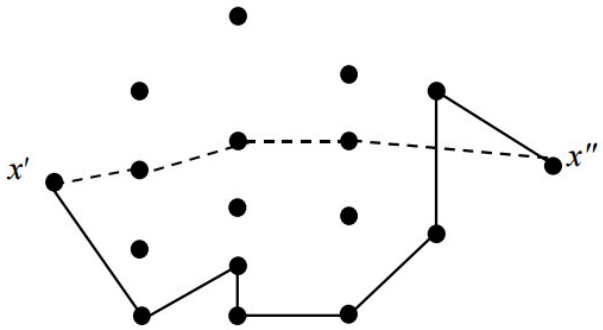


Figure 7: Image taken from Glover et al. (2000) – Original path shown by heavy line and one possible relinked path shown by dotted line.

With path relinking between x' and x'', a new series of solutions is likely to be explored. Solution x' would be the initiating solution and solution x'' would be called the guiding solution. In Figure 7, this path from initiating to guiding solution is depicted as a dotted line and three new solutions are explored along the way. The path between x' and x'' is created based on rules designed to incorporate attributes contained in the guiding solution.

One possible rule to determine the path is to consider the differences between x' and x'' and take the best steps possible (based on an evaluation function) to reduce those differences one at a time. Let $\Delta(x', x'')$ denote the differences between the solutions x' and x''. With this definition, then a basic, high-level algorithm to perform path relinking might be included within another metaheuristic like TS as depicted in lines 5-9 in Algorithm 2.

Algorithm 2: Path relinking high-level example

```
1  begin
2      Logic associated with top-level metaheuristic (e.g., TS)
3      ...
4      Some logic here to identify solutions x' and x'' and to trigger path relinking
5      while |Δ(x', x'')| > 0 do
6          Evaluate moves in Δ(x', x'')
7          Make best move
8          Update Δ(x', x'')
9      end
10     More logic here associated with the top-level metaheuristic
11     ...
12 end
```

To demonstrate this idea, consider the depiction in Figure 8. In the figure, the arrangement of colors represents the solution (red , orange, green, then blue, etc.) and might be associated with any permutation problem.

At left, we have the initiating solution, x', and at far right, the guiding solution, x''. The arrows indicate differences between a solution and the guiding solution. For instance, there are four differences between the initiating solution and the guiding solution, i.e., the orange, blue, yellow, and pink colors are in different places. These four differences would comprise the set $\Delta(x', x'')$.

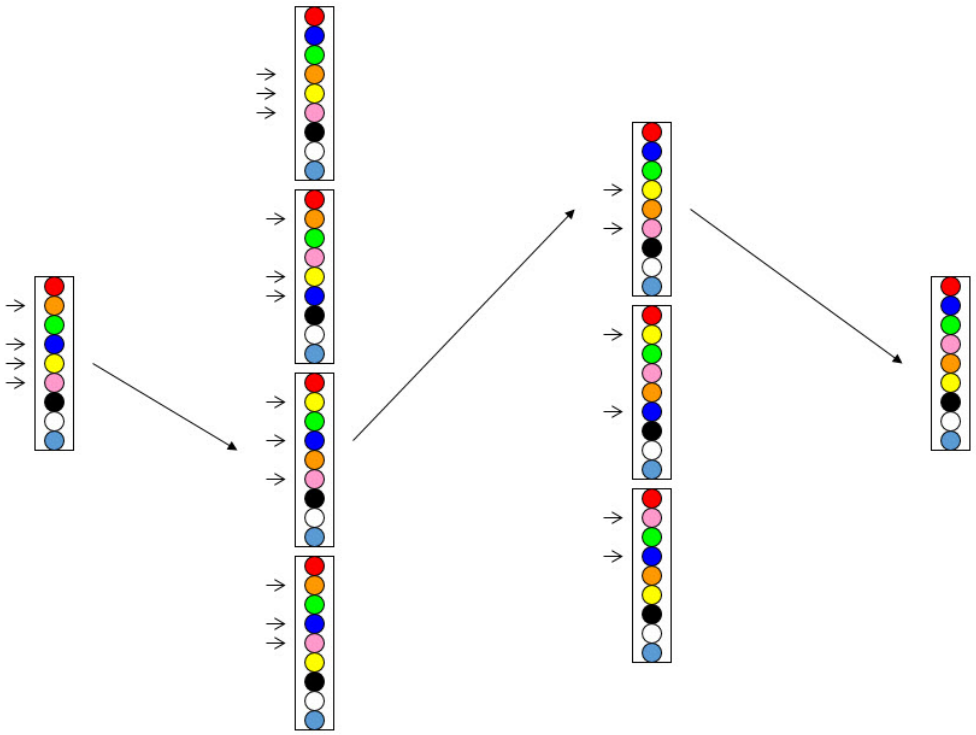


Figure 8: Example high-level PR

In the first iteration of PR, the moves to correct any one of the differences would be possible. All four possibilities are considered as shown in the second column of solutions in Figure 8. Assume the best evaluation is the one in which the yellow and orange positions are swapped (solution s_3). Once the move is made (i.e., $x' \leftarrow s_3$), then $\Delta(x', x'')$ is updated and the cardinality of set now drops to only 3 differences. Each of the three possible moves is evaluated, the best is selected and the move is made. This continues until the guiding solution is obtained.

The idea behind PR is that since solution x' and x'' are good solutions, the solutions "between" them are composed of "combinations" of both x' and x''. As such, the solutions which have not been explored along that path might be promising.

Of course, there are many variations on path relinking. For instance, instead of exploring only the solutions along the path between the initiating and guiding solutions, a local search of some sort might be performed along the way as depicted in Figure 9 with the green colored solutions.

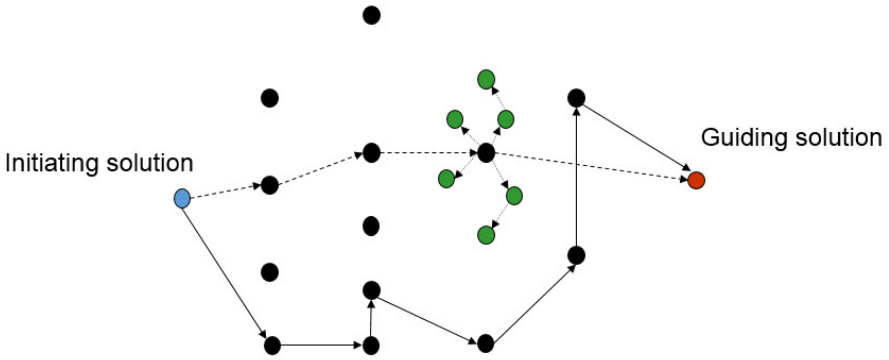


Figure 9: Path relinking with exploration

There are many other variations on how PR is implemented. The type described above is called "forward relinking" since it moves from solution x' to a better solution x''. Backward relinking is also possible. Other possibilities include exploring infeasible solutions, for instance, if no feasible path exists between x' and x'', allow the PR process to evaluate infeasible solutions and "tunnel" its way through the infeasible space.

These and other variations of path relinking are listed below.

Relinking strategies	Description
Forward relinking	better solution is the guiding solution
Backward relinking	better solution is the initiating solution
Back-and-forward relinking	both directions are explored
Mixed relinking	starts at both ends and meets in the middle
Tunneling	occasionally allow infeasible moves
Randomized relinking	stochastic selection of moves
Truncated relinking	guiding solution is not reached
Extrapolated relinking	do not stop when reach guiding solution but go beyond it

5. Summary

In summary, tabu search provides a powerful framework that can be applied relatively easily to many problem types. On the other hand, there are many variations on how it is to be implemented which provide opportunity for customization to specific problem types. The technique is designed in such a way as to work well with other exploratory processes such as strategic oscillation and path relinking. A depiction of the overall tabu search framework is provided in Figure 10.

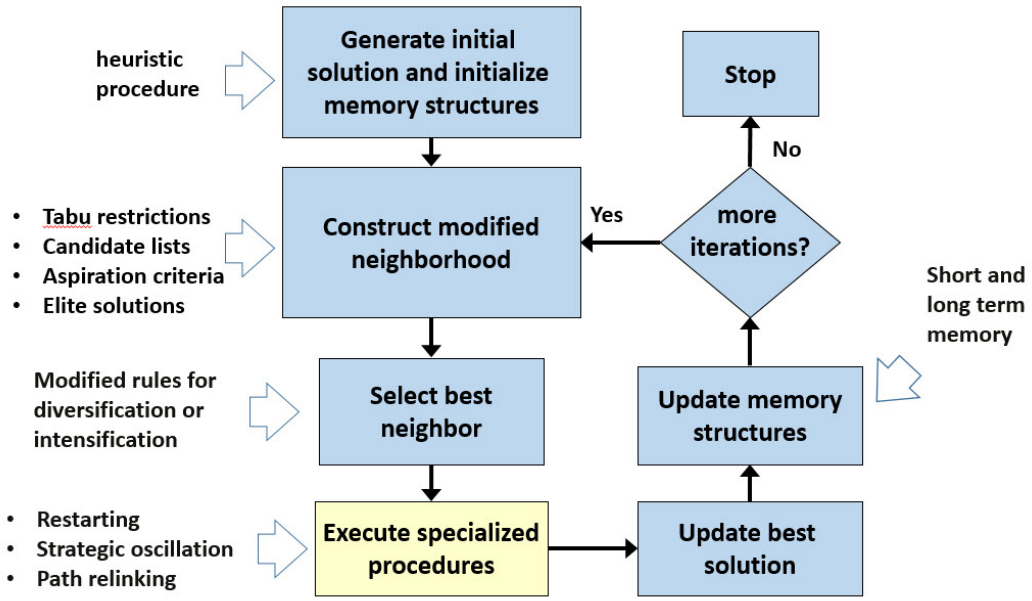


Figure 10: Tabu search framework