A Guide to Implementing Tabu Search

Manuel Laguna [1]

Key words

Tabu search, *n*-queens problem, local strategies.

Abstract

In recent years the number of tabu search (TS) users has dramatically increased, as reflected by the number of TS-related publications appearing in scientific journals. However, the literature offers little help to researchers and practitioners interested in applying the tabu search framework for the first time. This paper is intended to be a guide for those newcomers to the tabu search field. The TS elements are described using pseudo-code "modules" in the context of the *n*-queens problem. This combinatorial problem has been widely used as testbed to develop and benchmark search procedures in the artificial intelligence area. Alternative TS designs are presented and their relative merits are discussed.

Resumen

Recientemente, el número de usuarios de la técnica de búsqueda tabu ha aumentado considerablemente, lo cual se refleja en el número de publicaciones relacionadas con éste tema. Sin embargo, la literatura no ofrece mucha ayuda a aquellos científicos y practicantes interesados en aplicar la técnica de búsqueda tabu por primera vez. Este artículo pretende ser una guia para las personas que son nuevas en el campo. Los elementos de la búsqueda tabu son descritos usando modulos en forma de seudo-programas en el contexto del problema de las *n* reinas. Este problema combinatorio ha sido usado como prueba para desarrollar y calibrar procedimientos de búsqueda en el area de inteligencia artificial. El artículo presenta diseños alternativos de búsqueda tabu y discute su relativa importancia.

Received April 1994, accepted April 1994.

1 Graduate School of Businéss and Administration, Campus Box 419, University of Colorado, Boulder, CO 80309-0419, Estados Unidos, E-mail: manuel@mayan.colorado.edu.

1. Introduction

The term *tabu search* (TS) is now familiar to many researchers and practitioners in the fields of operations research and artificial intelligence. This paper is written for those who are interested in tabu search and would like to implement it effectively. We do not assume that the reader is familiar with tabu search and its components, and therefore we include a basic introduction to this methodology in section 3.

In order to discuss TS implmentation issues, a combinatorial problem with a simple and regular structure is desirable as an example. The *n-queens* problem provides an excellent context for our discussion. The *n*-queens is considered a classical combinatorial problem in the artificial intelligenge (AI) literature and has been extensively used as a benchmark for AI search problem-solving strategies. In fact, this problem has even found some practical significance in the area of VLSI testing and traffic control (Sosic and Gu 1989).

2. The N-Queens Problem

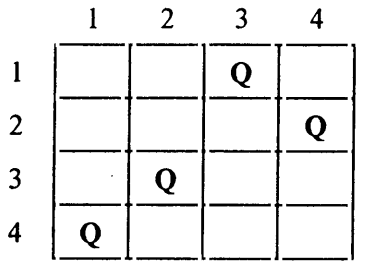
In simple terms, the *n*-queens problem consists of placing n queens on a $n \times n$ chessboard in such a way that no two queens capture each other. In order for this to happen, no two queens should be placed on the same row, the same column, or the same diagonal. If two queens are placed such that they are able to capture each other, it is said that a "collision" has occurred. Therefore, from an optimizaltion point of view the problem is to find a configuration (i.e., a set of n cells where to place the queens) that minimizes the total number of collisions. The optimal solution to this problem is obviously zero, since it is known that there exists at least one configuration for which no collisions occur. Alternatively, the problem may be viewed as maximizing the number of queens that can be placed on a $n \times n$ chessboard subject to the restriction that no row, column, or diagonal may contain more than one queen. In this case the optimal solution is n. These two interpretations of the *n*-queens problem may be used to formulate altenative 0-1 integer programming models. However, for our purposes this is not necessary, instead we will represent the *n*-queens as a permutation problem.

Let the queens be indexed by the letter i (for $i = 1,\ldots,n$), and let queen i be placed always on the i^{th} row. If we let $\pi(i)$ be the index of the column where queen i is placed, then a configuration is given by the following permutation:

$$\Pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$$

which fully specifies the exact position of the n queens on the chessboard. This representation guarantees that no two queens will attack each other on the same row or the same column. The problem is then to minimize the total number of collisions on the diagonals. Figure 1 shows a 4 × 4 chessboard that corresponds to the permutation $\Pi = \{3,4,2,1\}$. Note that this configuration has a total of 2 collisions (i.e., queens 1 and 2 as well as queens 3 and 4 are attacking each other).

Figure 1: A 4 × 4 queen configuration.



A relevant piece of information during the search will be the number of queens that are placed on the same diagonal on a given configuration. Since the chessboard in the n-queens problem contains $2n-1$ *negative* diagonals and $2n-1$ *positive* diagonals, two arrays (referred as d_1 and d_2) of size $2n-1$ will be sufficient to store this information. In order to identify each diagonal, we note that the sum of the row index and the column index is constant on any negative diagonal, and that the difference of these indexes is constant on any positive diagonal. For example, queens 1 and 2 are placed on the positive diagonal indexed −2 (i.e., $1-\pi(1) = 2-\pi(2) = -2$). Likewise, queens 3 and 4 are placed on the negative diagonal with index 5 (i.e., $3+\pi(3) = 4+\pi(4) = 5$). Therefore, the indexes for the positive diagonals range from $1-n$ to $n-1$, while the range for the negative diagonal is 2 to $2n$. The description of the permutation problem and the notation presented above are consistent with those given by Sosic and Gu (1989).

3. Basic Tabu Search

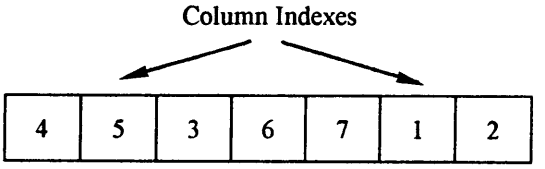
Webster's disctionary defines *tabu* or *taboo* as "set apart as charged with a dangerour supernatural power and forbidden to profane use or contact ..." or "banned on grounds of morality or taste or as constituting a risk ...". Tabu search scarcely involves reference to supernatural or moral considerations, but instead is concerned with imposing restrictions to guide a search process to negotiate otherwise difficult regions. These restrictions operate in serveral forms, both by direct exclusion of search alternatives classed as "forbidden," and also by translation into modified evaluations and probabilities of selection.

The philosophy of tabu search is to derive and exploit a collection of principles of intelligent praoblem solving. A fundamental element underlying tabu search is the use of flexible memory. From the standpoint of tabu search, flexible memory embodies the dual processes of creating and exploiting structures for taking advantage of history (hence combining the activities of acquiring and profiting from information).

Before presenting implmentation details, we focus on the 7-queens problem to introduce and illustrate the basic components of tabu search. First we assume that an initial solution for this problem can be constructed, e.g., randomly, although in most cases it will be preferred to do it

in some intelligent fashion (i.e., by taking advantage of some problem-specific structure). Suppose the initial solution to the problem is the one shown in Figure 2.

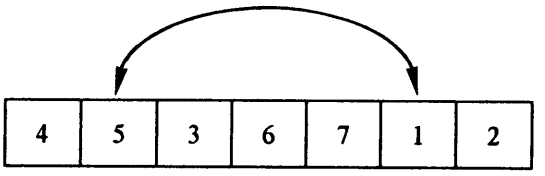
Figure 2: Initial Permutation.



The oradering in Figure 2 specifies that there is a queen in row 1 column 4, another in row 2 column 5, etc. The resulting configuration has a total of four collisions (i.e., queens pairs (1, 2), (4, 5), (6, 7), and (2, 6) are on the same diagonals). TS methods operate under the assumption that a neighborhood can be constructed to identify adjacent solutions that can be reached from any current solution. Pairwise exchanges (or swaps) are frequently used to define neighborhoods in permutation problems, identifying *moves* that lead from one solution to the next. In our problem, a swap exchanges the position of two queens as illustrated in Figure 3. Therefore, the complete neighborhood of a given current solution consists of the 21 adjacent solutions that can be obtained by such swaps.

Associated with each swap is a move value, which represents the change on the objective funtion value (i.e., the total number of collisions) as a result of the proposed exchange. Move values generally provide a fundamental basis for evaluating the quality of a move, although other criteria can also be important, as indicated later. A chief mechanism for exploiting memory in tabu search is to classify a subset of the moves in a neighborhood as forbidden (or tabu). The classification depends on the history of the search, particularly as manifested in the recency or frequency that certain move or solution components, called *attributes*, have participated in generating past solutions. For example, one attribute of a swap is the identity of the pair of elements that change positions (in this case, the two queens exchanged).

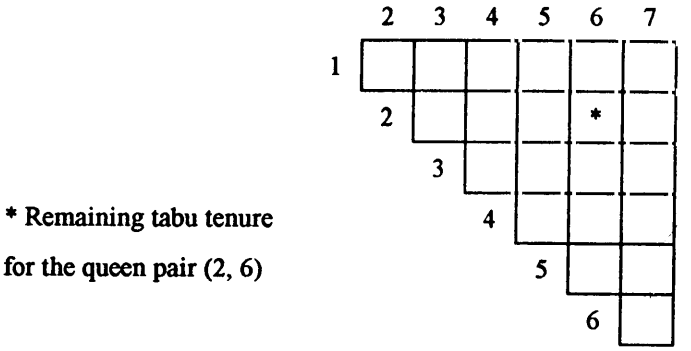
Figure 3: Swap of Queens 2 and 6.



As a basis for preventing the search from repeating swap combinations tried in the recent past, potentially reversing the effects of previous moves by interchanges that might return to previous positions, we will classify as tabu all swaps composed of any of the most recent pairs of such queens; in this case, for illustrative purposes, the three most recent pairs. This means that a queen pair will be kept tabu for a duration (tenure) of 3 iterations. Since exchanging

queens 2 and 6 is the same as exchanging queens 6 and 2, both may be represented by the pair (2, 6). Thus, a data structure such as the one shown in Figure 4 may be used.

Figure 4: Tabu Data Structure for Attributes Consisting of Queen Pairs Exchanged.



Each cell of the structure in Figure 4 contains the number of iterations remaining unitl the corresponding queens are allowed to exchange positions again. Therefore, if the cell (3, 5) has a value of zero, then queens 3 and 5 are free to exchange columns. On the other hand, if cell (2, 4) has a value of 2, then queens 2 and 4 may not exchange column assignments for the next two iterations (i.e., a swap that exchanges these queens is classified tabu).

The type of move attributes illustrated here for defining tabu restrictions are not the only ones possible. For example, reference may be made to separate queens rather than queen pairs, or to the column assignments of queens, and so forth. Some choices of attributes are better than others, when considered in combination with the tabu restrictions being imposed. (Attributes involving created and broken links between immediate predecessors and successors are often among the more effective for many permutation problems, including the traveling salesman and a variety of single machine scheduling problems.)

To implement tabu restriction such as those based on queen pairs, an important exception must be taken into account. Tabu restrictions are not inviolable under all cirlcumstances. When a tabu move would result in a solution better than any visited so far, its tabu classification may be overriden. A condition that allows such an override to occur is called an *aspiration criterion*. The following shows 5 iterations of the basic tabu procedure that employs the *paired queen* tabu restriction and the *best solution* aspiration criterion.

The starting solution has a total of 4 collisions, and the tabu data structure is initially empty (i.e., it is filled with zeros, indicating no moves are classified tabu at the beginning of the search). After evaluating the candidate swap moves, the top five moves (in terms of move values) are shown in the table for iteration 0 above.

Iteration 0 (Starting Point)

Current solution

5	3	6	7	1	2

Number of collisions = 4

Tabu structure

	2	3	4	5	6	7
1						
2						
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
1	7	-2	*
2	4	-2	
2	6	-2	
5	6	-2	
1	5	-1	

This information is provided by an independent evaluation subroutine designed to identify move values for this particular problem. (Of course, it is not necessary for the subroutine to sort and identify each of the 5 best moves, since we are interested only in the best). To locally minimize the total number of collisions, we swap the column assignments of queens 1 and 7, as indicated by the asterisk. The total gain of such a move equals 2 collisions. Notice that the choice of the swap (1,7) was arbitrary since the same gain may be achieved by exchanging queens 2 and 4, 2 and 6, or 5 and 6. In this case the selection rule is such that among moves with the same value the one found first is selected as the best of its class. Other alternatives are possiable specially if the concept of *frequency* is applied (as shown later).

Iteration 1

Current solution

2	5	3	6	7	1	4

Number of collisions = 2

Tabu structure

	2	3	4	5	6	7
1						3
2						
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
2	4	-1	*
1	6	0	
2	5	0	
1	2	1	
1	3	1	

The new current solution has 2 collisions (i.e., the previous number of collisions plus the value of the selected move). The tabu structure now shows that swapping the positions of queens 1 and 7 is forbidden for 3 iterations. The most improving move at this step is to swap 2 and 4 for a gain of 1. Note that performing the exchange (1,6) or (2,5) results in a new configuration with the same number of collisions as the old one, since the value associated with either of these moves is zero.

Iteration 2

Current solution

2	6	3	5	7	1	4

Number of collisions = 1

Tabu structure

	2	3	4	5	6	7
1						2
2			3			
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
1	3	0	*
1	7	1	T
2	4	1	T
4	5	1	
6	7	1	

The new current solution becomes the best solution found so far with 1 collision. At this iteration, two exchanges are classified tabu, as indicated by the nozero entries in the tabu structure. Note that entry (1, 7) has been decreased from 3 to 2, indicating that its original

Iteration 3

Current solution

3	6	2	5	7	1	4

Number of collisions = 1

Tabu structure

	2	3	4	5	6	7
1		3				1
2			2			
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
1	3	0	T
1	7	0	T
5	7	1	*
6	7	1	1
1	2	2	

tabu tenure of 3 now has 2 remaining iterations to go. This time, none of the candidates (including the top 5 shown) has a negative move value. Therefore, a nonimproving move has to be made. The most attractive nonimproving move is the swap of queens 1 and 3 for a value of zero. In some implementations of tabu search, where a large proportion of moves in a neighborhood have a value of zero, researchers have found convenient to forbid the selection of moves with zero move value. If we had imposed that restriction in our example, the move (4, 5) would have been selected since the second and third moves in the list are classified tabu.

The new current solutions has the same number of collisions as the previous one, as a result of executing a move with a zero move value. The tabu data structure now indicates that 3 moves are classified tabu, with different remaining tabu tenures. At the top of the candidate list, we find the swap of queens 1 and 3, which in effect represents the reversal of the previous move performed, and is classified tabu. The second move in the list is also classified tabu, and although it has become more attractive from iteration 2 to iteration 3, its selection does not lead the search to a better solution than the best found so far, and therefore this move is discarded. The next most attractive move is then to swap queens 5 and 7, which results in an increase in the total number of collisions.

Iteration 4

Current solution

3	6	2	5	4	1	7

Number of collisions = 2

Tabu structure

	2	3	4	5	6	7
1		2				
2			1			
3						
4						
5						3
6						

Top 5 candidates

Swap		Value	
4	7	-1	*
5	7	-1	T
1	5	0	
2	5	0	
2	4	2	T

The neighborhood of the current solutions includes an improving move that is not classified tabu. The exchange of column assignments for queens 4 and 7 results in a reduction of 1 collision, and therefore it is selected as the best move in the current iteration.

Iteration 5

Current solution

3	6	2	7	4	1	5

Number of collisions = 1

Tabu structure

	2	3	4	5	6	7
1		1				
2						
3						
4						3
5						2
6						

Top 5 candidates

Swap		Value	
1	3	-1	T*
5	6	-1	
5	7	0	T
1	6	0	
1	7	2	

As in iteration 3, at the top of the list is the swap (1, 3), which still represents the reversal of the moved performed at iteration 2. However this time, performing this move produces a solution with an objective function value that is superior to any previous one found. Therefore, we make use of the aspiration criterion to override the tabu classification of this move and select it as the best on this iteration. The resulting solution becomes the incumbent new best solution and the process may continue, however the new solution already represents a configuration with the minimum possible number of collisions (i.e., zero).

Note that the chosen tabu restriction and tabu tenure of 3 results in forbidding only 3 out of 21 possible swaps, since the queen pair with a residual tenure of 1 always drops to a residual tenure of 0 each time a new pair with tenure 3 is introduced. (By recording the iteration to determine the remaining tabu tenure, it is unnecessary to change these entries at each step as we do here.)

In some situation, it may be desirable to increase the percentage of available moves that receive a tabu classification. This may be achieved either by increasing the tabu tenure or by changing the tabu restriction. For example, a tabu restriction that forbids swaps containing at least one queen of a queen pair will prevent a larger number of moves from being executed, even if the tenure remains the same. (In our case, this restriction would forbid 15 out of 21 swaps if the tabu tenure remains at 3) Such a restriction is based on single queen attributes instead of paired queen attributes, and can be implemented with much less memory, i.e., by an array that records a tabu tenure for each queen separately. Generally speaking, regardless of the type of restriction selected, improved outocomes are often obtained by tabu tenures that vary dynamically, as described later.

The accompaniment of recency based memory with frequency based memory adds a component that typically operates obver a longer horizon. To illustrate one of the useful

longer term applications of frequency based memory, suppose that 25 TS iterations have been performed, and that the number of times each queen pair has been exchanged is saved in an expanded tabu data stratucture. The lower diagonal of this structure now cointains the frequency counts.

Iteration 26

Current solution

1	3	6	2	7	5	4

Number of collisions = 1

Tabu structure

	1	2	3	4	5	6	7
1						3	
2							
3	5					2	
4		3					1
5	2			4			
6	1				2		
7		4	3	1			

Frequency

Top 5 candidates

Swap		Value	Penalized Value	
1	6	0	1	T
1	3	1	6	
1	5	1	3	*
2	7	1	5	
3	7	1	4	

At the current iteration (iterations 26), the recency memory indicates that the last three queen pairs exchanged were (1, 6), (3, 6), and (4, 7). The frequency counts show the distribution of moves throughout the first 25 iterations. We use these counts to diversify the search, driving it into new regions. This diversifying influence is restricted to operate only on particular occasions. In this case, we select those occasions where no admissible improving moves exist. Our use of the frequency information will penalize nonimproving moves by assigning a larger penalty to swaps of queen pairs with greater frequency counts. (Typically these counts would be normalized, as by dividing by the total number of iterations or their maximum value.) We illustrate this in the present example by simply adding a frequency count to the associated move value.

The list of top candidates for iteration 26 shows that the most attractive move is the swap (1, 6), but since this queen pair has a residual tabu tenure of 3, it is classified tabu. The move (1, 3) has a value of 1, and it might otherwise be the one next preferred, except that its associated queens have been exchanged frequently during the history of the search (in fact, more frequently than any other quenn pair). Therefore, the move is heavily penalized and it looses its attractiveness. The swap of queens 1 and 5 thus is selected as the best move on the current iteration. (Note that in this case the frequency-based penalities helped to differentiate among moves with the same move value.) The strategy of instituting penalties only under particular conditions is used to preserve the aggressiveness of the search. In addition, frequencies defined over different subsets of past solutions, particularly subsets of elite solutions consisting of high quality local optima, give rise to complementary strategies called

intensification strategies. Intensification and diversification strategies interact to provide fundamental cornerstones of longer term memory in tabu search. (An example of implementing an intensification strategy is presented at the end of section 4.)

Following the first tabu search proposal, which was implemented under the name of *oscillating assignment heuristic* (Glover 1977), researchers have expanded the number of mechanisms employed by TS methods to achieve a balanced interplay between intensification and diversification. Some of these procedures were designed in the context of particular problems, but in general they can be adapted to other situation. Regardless of the sophistication of particular TS implementations, the *short term memory function* is often considered the core of the methodology, due to its design for allowing the search to go beyond locally optimal points (as illustrated in the previous example). Hence, we will start by discussing the implementation of a simplistic TS procedure for the n-queens problem based on this component. In general, TS users consider very simple forms of the TS methodlogy in their first attempts to solve a new problem, and it is not until enhanced outcomes are desired that more elaborate schemes are added. The discussion will be organized in three different modules: Initializaltion, Move Evaluation, and Execution and Updating. A good practice is to program these modules as separate functions (or subroutines), thus facilitating future additions or changes to the method. In the following subsection we discuss each of these modules in detail, using an experimental TS code for the n-queens problem. The code is written in ANSI C, and an electronic copy can be otained via e-mail from the author.

3.1 Initialization

The initialization module is used to generate a starting solution (that may be feasible or not), to calculate the objective function value of this sollution, and to initialize all data structures (tabu or otherwise). For the n-queens problem, we have selected a random permutation to start the procedure. Constructive approaches are often used in place of random starts, because they generally provide a "better" starting point (measured by the objective function value). The random permutation in our example is found by first generating n random numbers r_i, and then by letting the array Π contain the ascending order of the elements r_i. Since the the elements of r are random, then the elments of Π are also random.

Before the objective function (i.e., the total number of collisions) can be evaluated, the data structures d_1 and d_2 must be initialized. The number of queens on each diagonal is originally set to zero, and then for each queen the counts on the corresponding negative and positive diagonals are incremented as follows:

for $(i = 2,\ldots,2n)$ {
  $d_1(i) \leftarrow 0;$
  $d_2(i-n-1) \leftarrow 0;$
}

for $(i = 1,\ldots,n)$ {
  $d_1(i+\pi(i)) \leftarrow d_1(i+\pi(i))+1;$
  $d_2(i-\pi(i)) \leftarrow d_2(i-\pi(i))+1;$
}

Once the arrays d_1 and d_2 have been initialized, the objective function may be calculated as follows:

collisions ← 0;
for ($k = 2,\ldots,2n$)
  collisions ← *collisions* + **max** $(0, d_1(k) - 1)$ + **max** $(0, d_2(k - n - 1) - 1)$;

Note that the number of collisions on a particular diagonal is given by the maximum between zero and the number of queens placed on the diagonal minus one.

Finally, the tabu data structure is initialized. As discussed earlier, the tabu data structure is intimately linked to the choice of the move mechanism and the move attributes (for defining tabu restrictions). To be consistent with our previous example, we choose swaps as the move mechanism and *queen pairs* as the move attrributes. A useful tabu data structure in this case takes the form of a "time stamp" that indicates the interation number at which a move looses its tabu status. Specifically, the two dimensional array *tabu_time* is created, where the (i,j) element of this array contains the iteration number in which the queen pair (i,j) looses its tabu status. Therefore, at the beginning of the search , *tabu_time* must be initialized to zero:

for ($i = 1,\ldots,n-1$)
  for ($j = i+1,\ldots,n$)
    tabu_time (i,j) ← 0.

Similar move attributes may be defined to create different tabu restrictions. In occasions the same tabu data structure may be preserved, by changing the definition of its elements. An example of this is given in section 4.

3.2 Best Move

One of the main differences between Monte Carlo methods, such as simulated annealing, and tabu search is the aggressive orientation of TS. Tabu search methods are designed to select at each step what is considered the best move available given the current search state. In basic procedures the rule of selecting the "most improving" move or the "least nonimproving" move is generally followed (as we did in the first 5 iterations of our example). However, other variants are possible that might lead to more powerful procedures. We will examine some of these options in sections 4 and 5.

The *best move* module is computationally more expensive than any other module in most TS procedures. In our example, finding the best swap move as defined above requires $O(n^2)$ time. Therefore, it is extremely important that a computationally efficient procedure is used to evaluate the merit of each move. (This evaluation is commonly referred to as *move value*, which as before, it represents the change on the objective function value.) In addition to implementing efficient move evaluators, particular attention must be put to designing memory structure that allow the evaluation of only a subset of moves (i.e., those whose value changes as a result of executing the "best" move at any given iteration).

Another functional characteristic of this module consists of checking the admissibility of moves. A move is admissible if it is non-tabu or its tabu status is overridden by the aspiration level criterion being used. The most simple form of this criterion (and also the one most widely used) is the one employed in our example, i.e., the one that renders a tabu move admissible if its execution leads to a solution that is better than the best found so far. The search for the best admissible move is done by the following partial pseudo-code:

```
move_value ← n+1
for (i = 1,...,n-1) {
    for (j = i+1,...,n) {
        value ← evaluate (i,j,Π,d1,d2);
        if (tabu_time (i,j) < iter or c_curr + value < c_best) {
            if (value < move_value) {
                move_value ← value;
                move_i ← i;
                move_j ← j;
            }
        }
    }
}
```

In our implementation, the best move value found is originally set to $n+1$, which is the maximum possible number of collisions (when each queen collides on both of its diagonals). Then the *value* of each possible swap move is calculated by the *evaluate* function, which requires the indexes of the queens that participate in the exchange, the current permutation, and the number of queens on each diagonal. The evaluation of a move is an operation that requires constant time (i.e., it does not depend on n), since only eight diagonals need to be checked (i.e., four diagonals for each of the queens participating in the exchange). After a move is evaluated, its tabu status is checked. The swap of queens i and j is tabu (under the current tabu restriction) if its *tabu_time* is greater than or equal to the current iteration number (*iter*). The candidate moves that become admissible are those that are not tabu, or the aspiration criterion overrides their tabu status (i.e., when *c_curr* + *value* < *c_best*, where *c_curr* and *c_best* are respectively the total number of collisions in the current and best configurations). If an admissible move has a better value than the current best (i.e., *value* < *move_value*), this move becomes the new best and is stored on the structure move that has the elements i, j, and *value*. This structure contains all the information necessary to execute the best move and update both the current configuration and the corresponding objective function value.

Traditionally in the literature the notion of a best move corresponds to a move that yields the best objective function change, and often the convention is relied on for convenience. However, the TS philosophy more broadly views "best"to be a context, which encompasses a variety of dimensions in addition to objective function change. For example, in some problems

the relative objective function change produced by a move is a poor indicator of its quality (in terms of the likelihood that it will lead to an elite local optimum). For problem classes where this is true, a move that yields any objective function improvement satisfying a reasonable threshold may be considered equivalent to a best move. (A simple approximation is to rely on a *first improving* move for such problems.) Greater sophistication, by differentiating the relative contribution of different move attributes to the overall quality of a move (which is made possible in TS by the use of attribute based memory), can lead to improved outcomes. An example is provided in Laguna, et al. (1991).

3.3 Executing a Move and Updating

The execution of the best move modifies the current trial solution. In this module it is also customary to update data structures that are directly affected by changes in the current solution. For example, in single machine scheduling, it is useful to store the contribution towards the objective function value of individual jobs. This contribution clearly depends on the current schedule and must be updated after a move is performed. Similarly, in the context of the n-queens problem, the number of collisions on each diagonal (d_1 and d_2) must be updated. The *execute_move* function in the appendix shows both the updating procedure for the eight diagonals involved in the swap of the queen pair (i,j) and the actual modification of the current permutation Π.

The updating of the tabu structure is in this case a very simple operation. The attributes of the swap move are stored on the structural variables *move_i* and *move_j*. Then, *tabu_time* must be updated as follows in order to impose a tabu status on the exchange (i,j) for *tabu_tenure* iterations:

tabu_time (*move_i, move_j*) ← *iter* + *tabu_tenure*;

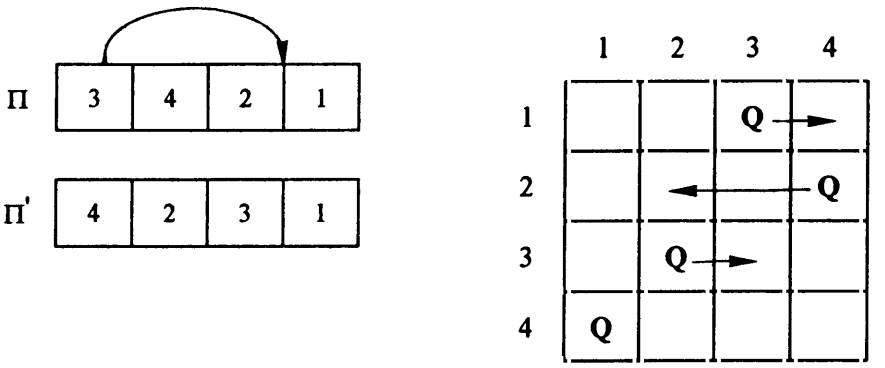
where *iter* is the current iteration number. Note that the *tabu_tenure* value used in our example was 3, and it is often referred to as the "short term memory size." Also note that, the update must be performed before executing the move (i.e., before exchanging the positions of *move_i* and *move_j*). Finally, the updating of the best solution found so far may be performed by simply copying the vector Π onto the vector Π^* (which contains the best configuration seen during the search), and the value *c_curr* onto *c_best*.

4. Additonal Tabu Search Features

A popular way of increasing the power of a TS procedure is to devise different move mechanisms that can be used either separately or simultaneously during a given search. One possibility in the current context is to introduce the use of insert moves. Figure 5 illustrates the effect of an insert move that modifies a permutation $\Pi = \{3, 4, 2, 1\}$ into the permutation $\Pi' = \{4, 2, 3, 1\}$.

Although the mechanisms and strategies in our implementation are based on a TS procedure that employs swap moves, the general structure would be identical for a procedure based on insert moves (or one that is based on both kinds of moves).

Figure 5: Effect of an Insert Move that Transforms Π into Π'.



Another possible modification to our basic procedure consists of redefining the tabu restrictions. For example, a different tabu restriction could be such that it prevents a queen from occupying a specific column. The *tabu_time* array previously defined can also be used to impose this restriction. In this case, the (i,j) element of the *tabu_time* matrix contains the iteration number at which queen i is allowed to return to column j. The initialization process should then blank the entire *tabu_time* matrix (instead of only the upper diagonal as done in the previous case). To enforce this new tabu restriction, the first *if* statement of the best move function (see section 3.2) must be modified as follows:

if (*tabu_time* $(i, \pi(j))$ and *tabu_time* $(j, \pi(i)) <$ *iter* or *c_curr* + *value* < *c_best*)

This condition checks that both queens are allowed to be placed on the proposed columns. If one of the positions is tabu, the whole move is classified tabu. Alternatively, the *and* operator in the *if* statement may be changed to an *or*, in order to create a less restrictive tabu structure. The updating procedure must be also modified , in order to record the appropriate move attributes in the tabu data structure. The updates are performed as follows:

tabu_time (*move_i*, π (*move_i*)) ← *iter* + *tabu_tenure*;

tabu_time (*move_j*, π (*move_j*)) ← *iter* + *tabu_tenure*.

The *tabu_time* array now indicates that queen *move_i* is not allowed to be assigned to column π(*move_i*) for *tabu_tenure* iterations, in the same way that queen *move_j* is not allowed to be assigned to column π(*move_j*) for the same number of iterations.

Other mechanisms that may improve performance are related to an increase in search diversification. One frequently used strategy to achieve this (i.e., to encourage the method to search unexplored regions), is to implement a *long term memory function* via frequency counts (as shown at the end of our example in section 3). A two-dimensional array (*freq_move*) may be created to store the number of times a particular swap has been executed. This array must be blanked by the *initialization* module, and after a swap of the queen pair (*move_i, move_j*), must be updated as follows:

freq_move (*move_i, move_j*) ← *freq_move* (*move_i,move_j*) + 1.

The information contained on the *freq_move* array may be used to create a penalized move evaluation function (as in iteration 26 of our example). A simple form of this function is:

penalized_value ← *value* + *weight***freq_move* (*i,j*);

where *weight* is a parameter that may be empirically adjusted (and it was set to 1 in the example). The purpose of this function is to penalize frquently executed moves with the goal of directing the search to configrations that have not been visited before. As mentioned earlier, the penalized function may be only used in non-improving regions (i.e., search states for which no improving admissible move is available). If saving computer memory is a concern, the lower diagonal of the *tabu_time* array can be used to store frquency information. When the exchange of columns between queen *move_i* and queen *move_j* is selected, its recency information is stored in the cell location *tabu_time* (*move_i, move_j*), while its frequency inforamtion is stored in *tabu_time* (*move_j,move_i*).

An alternative form of frequency count may be otained by defining *freq_confg*(i,j) to be the number of times that queen i has been placed on column j throughout the search. This information then is used to devise a re-starting procedure where the probability for queen i to be placed on column j is inversely proportional to *freq_confg*(i,j). The updating of *freq_confg* in this case is an $O(n)$ procedure, since the position of each queen must be recorded at every iteration.

The only search parameter of our basic tabu search implementation is the *tabu_tenure*. The value of this parameter must be adjusted to prevent the search from cycling (i.e., from indefinitely repeating the same sequence of moves) and to allow enough flexibility (as measured by the percentage of available moves at any given iteration). In an attempt to decrease the probability of cycling and the dependency of TS methods on fixed values of *tabu_tenure*, some researchers have incorporated what is known as dynamic tabu tenures. This strategy has bee implemented both in a random fashion (Taillard 1991) and also using deterministic patterns (Glover and Hubscher 1991). A random implementation, for example, may be such that the *tabu_tenure* parameter is replaced by *tabu_max* and *tabu_min*. Then, *tabu_tenure* is calculated as follows:

tabu_tenure ← *tabu_min* + *uniform* (*tabu_max* – *tabu_min*);

where *uniform*(x) is a function that returns a random integer uniformly distributed between zero and x. The *tabu_tenure* value is changed every *factor***tabu_tenure* iterations, where *factor* becomes an additional search parameter. Typical values for *factor* range between 2 and 3. Alternatively, a random *tabu_tenure* could be assigned to every move that is being classified tabu. This option avoids the need for adjusting the *factor* parameter.

Another form of *dynamic tabu tenures* is such that the value of *tabu_tenure* is made *move_dependent*. For example, suppose that queen i collides more times when placed on column $\pi(i)$ than when placed on column $\pi(j)$, also suppose that the opposite is true for queen j. Then a move-dependent strategy may be to allow queen j to return to column $\pi(j)$ faster than queen i is allowed to comeback to column $\pi(i)$. If the standard tenure is given by *tabu_tenure*,

then one way of achieving the move-dependent strategy is to assign the tabu tenure values for each move reversal as follows:

$$tabu\_time\ (move\_i, \pi\ (move\_i)) \leftarrow iter + tabu\_ternure$$

$$tabu\_time\ (move\_j, \pi\ (move\_j)) \leftarrow iter + tabu\_ternure/2.$$

More sophisticated forms of the move-dependent tabu tenure strategy are possible, for example by making the percentage of *tabu_tenure* (which is fixed to 100% and 50% above) a function of the move value.

For some applications it is not feasible to evaluate the entire neighborhood as defined by the move mechanism. In this case, *candidate list* strategies are often employed as a way of accelerating the best move selection. These strategies may be designed to take advantage of special problem structures. For example, for scheduling problems with deadlines, the candidate list of swaps moves may be reduced by considering only those pairs of jobs whose absolute deadline difference is less than a given value. In the *n*-queens problem, a candidate list of moves may be generated by considering exchanges of only those queens that participate in collisions at a given iteration. Moves are then drawn from the list for a number of fixed iterations at the end of which the list is rebuilt. For example, suppose that at the beginning of the search in a 7-queen problem, queens 1, 5 and 7 are the only one participating in collisions. Then for the next, say, three iterations, the candidate list of moves will consist only of those exchanges that contain queens 1, 5 and 7. At the fourth iteration a search on the entire neighborhood is performed to reconstruct the list of the queens that are participating in collisions.

Another strategy that is also common (and easier to implement) consists of selecting the *first* improving move available or the least non-improving one. In our example, this strategy means that the *best_move* function would require $O(n^2)$ evaluations only in the worst case (i.e., when no improving move is found). To implement this change, the following line must be added at the end of each of the two *for-loops* in the pseudo-code of section 3.2:

if (*move_value* < 0) break;

where, as before, *move_value* contains the value of the best admissible move found so far. An agressive version of the *first improving* strategy can be designed by extending the local search once the first improving move has been detected. For example, suppose that the first improving move consists of swapping queens *i* and *j*, then a more aggressive procedure would select the most improving move that exchanges the positions of either queen *i* or *j*.

Finally, some TS methods also include additional forms of search *intensification*. This strategy relates to *intermediate memory functions*. The general idea here is to identify solution structures that result in "good" objective function values, where the sense of "goodness" may change as the search progresses. Components of these structures may be encouraged to become part of the current solution by modifying evaluation criteria to favor their inclusion, or may more forcibly be injected to "seed" the values of certain variables or location of certain elements. When these component structures are present, then associated elements of the solution may be "locked," and an intensified search performed on the free elements. For

example, in production scheduling problems some job sequences may frequently appear as part of good solutions (which could be defined as local optima with objective function values that are only marginally different than the one corresponding to the best schedule found so far). The set of jobs that belong to these sequences are placed on an intermediate memory that prevents them from being moved, and the search continues on the set of "free" jobs. A termination criterion, such as a fixed number of iterations, is set to finish the intensification period.

Another form of intensification is possible in our examaple by using information from a modified *freq_confg* array (referred to as *freq_good_confg*). After a move is executed, the frequency counts on this array are only updated when the current solution is *percent* away from the best. The updating may be performed in the following way:

if (*c_curr* <= *c_best** (1+ *percent*))
  for ($i = 1,\ldots,n$)
    freq_good_confg $(i, \pi(i)) \leftarrow$ *freq_good_confg* $(i, \pi(i))+1$

Then, while intensifying the search, the best move selection may be modified to forbid exchanges of queens that are placed on columns with a "high" frequency count on the array *freq_good_confg* (where "high" is defined in terms of the maximum frequency count on the array).

In this section we have described some additional components that can be used to enhance the performance of TS methods. It is important to note that these additional levels of shophistication do not come "for free." They usually add to the computational complexity of the method, and at the same time they generally increase the number of parameters that require adjustment.

5. Hybrid Approaches

An intriguing approach to solving hard optimization problems is the development of hybrid search heuristics. The main idea behind this approach is to combine two or more methods in such a way that the resulting procedure is more effective than any of its components alone. In addition to tabu search, methodologies such as genetic algorithms (GAs) and simulated annealing (SA) are often candidates for the creation of hybrid procedures. Lin, Kao, and Hsu (1991) integrate GAs and SA to solve traveling salesman problems; Whitley and Hanson (1989) use a GA to optimize neural networks; Fox (1992) creates a hybrid approach by combining GAs, SA, and TS; and Glover, Kelly, and Laguna (1992) study ways of exploiting the similarities and differences between GAs and TS.

For illustration purposes, suppose that it is desired to create a search method that employs an SA sampling procedure embedded in a TS framework. One way of implementing such hybrid procedure within the context of the *n*-queens problem follows. We simply substitute the best move module presented in section 3.2 by the following pseudo-code:

```
accept_move ← 0;
do {
        (i,j) ← pick_two (n);
        value ← evaluate (i,j,Π,d1,d2);
        if (tabu_time (i,j) < iter or c_curr + value < c_best)
                move_value ← value;
                move_i ← i;
                move_j ← j;
                accept_move ← 1
        }
} while (accept_move = 0);
r ← uniform (1);
if (r> exp (-move_value/t)
        accept_move ← 0
else
        t ← t* f;
```

First note that the indicator variable *accept_move* has been added to the memory structure that contains the information related to the "best" move. The "do-loop" is performed until an admissible move is found under our standard tabu restriction. (The *if* statement within the loop may be replaced by the one presented at the beginning of section 4, in order to enforce an alternative form of the tabu restriction.) The *pick_two* function is a simple procedure that returns two integers randomly selected between 1 and n, such that $j > i$. Once an admissible move is found, it is accepted with probability $e^{-\Delta/t}$, where Δ is the move value and t is the current temperature. The function *uniform* (x) returns a random real value uniformly distributed between 0 and x. If the move is not accepted the indicator variable *accept_move* is switched back to zero, otherwise its value remains as one, and the temperature is decrease by a factor f. In this context, typical values for the starting temperature t range from 1 to 2, while the values for f range between 0.9 and 0.99. The initial t value and the f factor value define what is known in the simulated annealing literature as the *temperature schedule*.

The hybrid approach presented above differs from the "pure" tabu search procedure in that an iteration may or may not result in a move being executed. When a move is not accepted, the main TS iteration is still performed (i.e., remaining tabu tenures for tabu moves are modified), however the trial solution is not altered. Due to the sampling nature of the hybrid approach, an iteration of this procedure genrally requires less computational effort than in the "pure" TS case. Note that the hybrid procedure incorporates an additional diversification component by means of randomness, but this approach lacks the aggressiveness typical to TS methods.

It is also appropriate to point out the differences between this hybrid procedure and a method called *probabilistic tabu search*. In probabilistic TS, the neighborhood of a trial solution is fully or partially (if a candidate list is used) explored, and moves are ranked according to a given criterion (e.g., move value). Then, a move is selected as best using a probability distribution that assigns higher probability of selection to those moves with higher rank.

Therefore, an iteration of this method always results in a move being executed. Probabilistic TS methods preserve the aggressive orientation of the tabu search framework, while incorporating a random element for diversification purposes. For more details on this form of tabu search see Glover (1989).

6. Conclusions

We have undertaken to illustrate and discuss implementation issues that relate to most tabu search applications. For this purpose, we have presented a detailed description of a tabu search implementation for the solution of the *n*-queens problem, showing how to incorporate additional TS strageries into a basic mehtod, and discussing how much such strategies may affect the performance of the resulting parocedures.

An interesting extension of the method presented here consists of allowing the search to continue after finding the first queen configuration with zero collisions. In fact, many AI search procedures are tested for their ability to find all solutions to the *n*-queens problem for a given *n*. Long term memory functions and other advanced TS mechanisms may be expected to be useful for solving this more challenging problem.

Acknowledgment

I would like to tank Fred Glover and Bill Stewart (who honored us by taking his sabbatical year at the University of Colorado) for their valuable comments and criticisms.

References

[1] FOX, B., "Integrating and Accelerating Tabu Search, Simulated Annealing, and Genetic Algorithms," *Annals of Operations Research*, 41 (1992), 47-67.

[2] GLOVER, F., "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, 8 (1977), 156-166.

[3] GLOVER, F., "Tabu Search - Part I," *ORSA Journal on Computing*, 1, 3 (1989), 190-206.

[4] GLOVER, F., "Tabu Search: A Tutorial," *Interfaces*, 20, 4 (1990) 74-94.

[5] GLOVER, F. and R. HUBSCHER, "Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling," to appear in *Computers and Operations Research*, 1991.

[6] GLOVER, F., J.P. KELLY, and M. LAGUNA, "Genetic Algorithms and Tabu Search: Hybrids for Optimization," to appear in *Computers and Operations Research*, 1992.

[7] LAGUNA, M., J.P. KELLY, J.L. GONZALEZ-VELARDE, and F. GLOVER, "Tabu Search for the Multilevel Generalized Assignment Problem," to appear in the *European Journal of Operation Research*, 1991.

[8] LIN, F-T, C-Y KAO, and C-C HSU, "Incorporating Genetic Algorithms into Simulated Annealing," Dept. of Computer Science and Information Engineering, National Taiwan University, 1991.

[9] SOSIC, R. and J. GU, "A Polynomial Time Algorithm for the N-Queens Problem," *SIGART Bulletin*, 1, 3 (1990), 7-11.

[10] TAILLARD, E., "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing*, 17 (1991), 443.

[12] WHITLEY, D. and T. HANSON, "Optimizing Neural Networks Using Faster, More Accurate Genetic Search," Computer Science Department, Colorado State University, 1989.