Tabu Search

Winter 2011

Similarity to Neighborhood Search

- Tabu Search (TS) begins in the same way as ordinary neighborhood search, moving iteratively from one solution to another until a satisfactory solution is obtained.
- Going from one solution to another is called a move. The neighborhood of a solution is made of all the solutions in which the value of one variable is changed to its immediate adjacent values in a sorted list of discrete values

Motivation

- If search has already visited a point in the search space, why waste computation time revisiting and re-evaluating that point?
- Such a point should be made "**tabu**"
- **Tabu** – socially or culturally proscribed: **forbidden** to be used, mentioned, or approached because of social or cultural rather than legal prohibitions.

Examples of Taboo

- Exposure of body parts
- Restrictions on the use of offensive language and gesture
- Foods and beverages which people abstain
- Burial grounds or places of death
- Sex outside of marriage

Definition

- Tabu search is based on introducing ***flexible memory*** structures in conjunction with ***strategic restrictions*** and ***aspiration levels*** as a means for exploiting search spaces
- 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

References

- Glover F. (1986). " Future Paths for Integer Programming and Links to Artificial Intelligence", *Computers and Operations Research*, 5, 533-549.
- Glover F. (1990). "Tabu Search: A Tutorial", *Interfaces*, 20(4), 74-94.

At a Glance

- Construct a candidate list from the neighborhood
- Pick a move that is not tabu
- Allow tabu move if it meets aspiration criteria
- Update tabu list if necessary
- Replace current solution with new solution if it is better
- Repeat

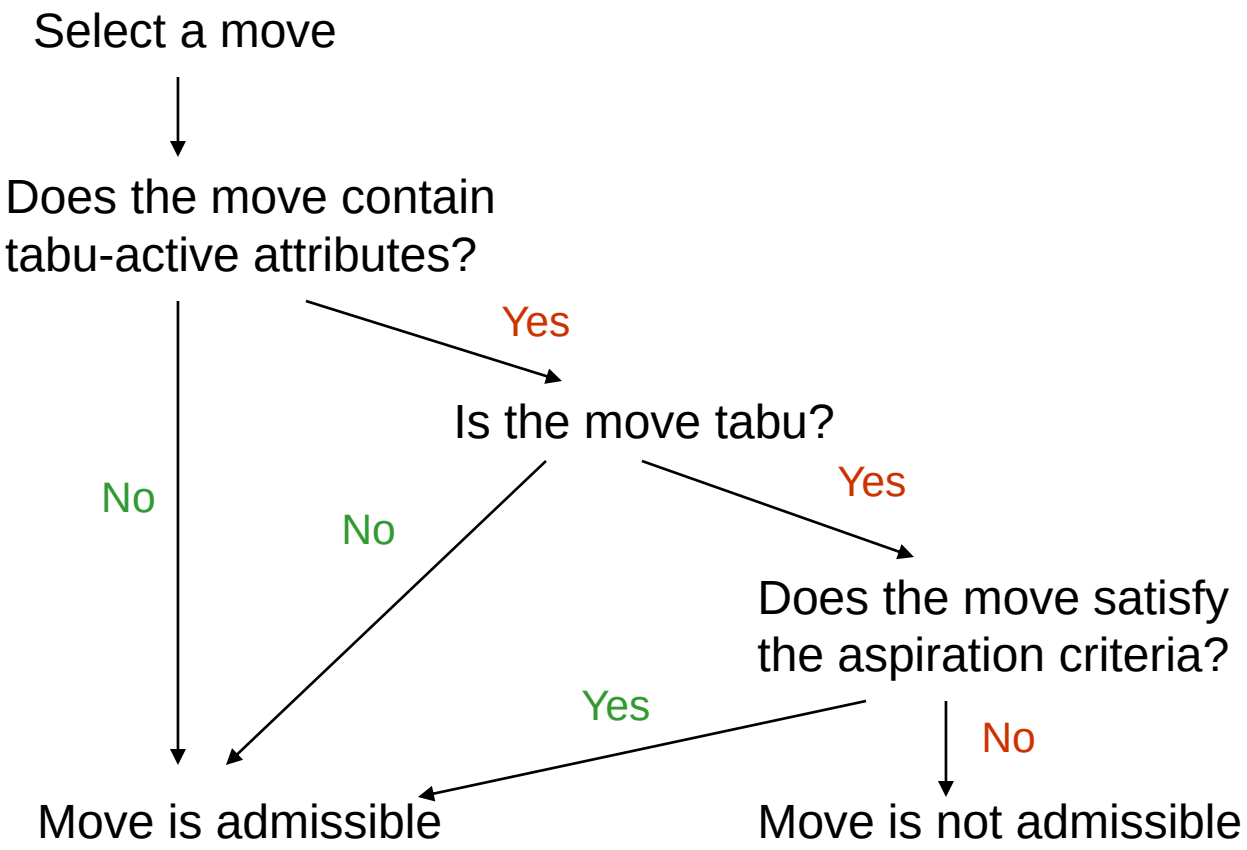
Main Strategies

- **Forbidding strategy**: control what enters the tabu list
- **Freeing strategy**: control what exits the tabu list and when
- **Short-term strategy**: manage interplay between the forbidding strategy and freeing strategy to select trial solutions

Main Elements

- **Tabu list**: to record a limited number of attributes of solutions (moves, selections, assignments, etc) to be discouraged
- **Tabu tenure**: number of iterations a tabu move is considered to remain tabu;
- **Aspiration criteria**: accepting an improving solution even if generated by a tabu move

Tabu Decision



Illustrative Example

- Permutation problem
- Design a material consisting of a number of insulating modules; the order in which the modules are arranged determines the overall insulating property
- Find the ordering of modules that maximizes the overall insulating property

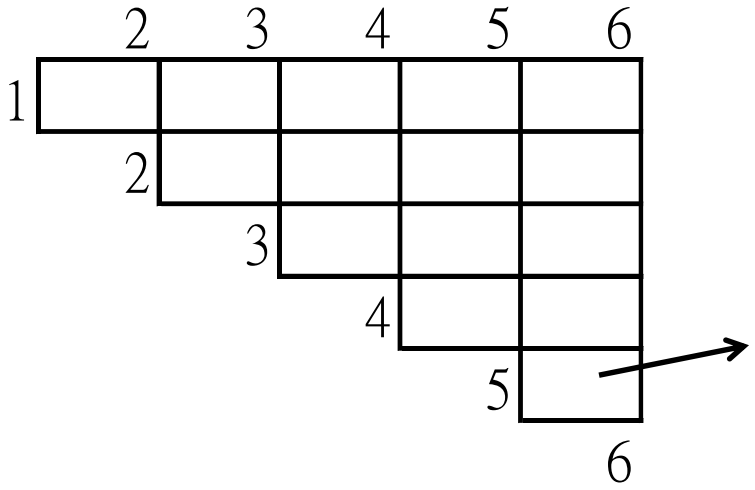
Solution and Swap

- Possible solution: [2 **5** 7 3 4 **6** 1]
- Adjacent solutions (neighborhood) can result from swapping: [2 **6** 7 3 4 **5** 1]
- There are 21 adjacent solutions (i.e., moves) in this case

  Combination(7,2) = 7!/(2!5!) = 21

Tabu List

- Each cell contains the number of iterations remaining until the corresponding modules are allowed to exchange (after certain **tabu tenure**)



Aspiration

- Tabu restriction may be violated when
  - A move would result in a solution much better than any visited before
- This override condition is called "aspiration criterion"

Basic Tabu Procedure

- Hereafter, we illustrate 4 iterations of the basic tabu procedure; tabu tenure = 3; aspiration criterion>+5
- At iteration 0; Generate a start solution
- Evaluate the candidate swap moves by an independent subroutine

Current Solution

2	5	7	3	4	6	1

Objective Value = 10

Tabu List

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

Change in the objective value ↑

Swap	Move Value
5, 4	6
7, 4	4
3, 6	2
2, 3	0
4, 1	-1

Iteration 1

- Swap by the top candidate move (5,4); update tabu list; update tabu list; evaluate candidate solutions

Current Solution (end of iteration 1)

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

Objective Value = **16**

Tabu List

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

Swap	Move Value
3, 1	2
2, 3	1
3, 6	-1
7, 1	-2
6, 1	-4

Iteration 2

- Swap by the top candidate move (3, 1); update tabu list; evaluate candidate solutions
- Non-improving move is necessary
  - pick (2,4) because (1,3) is classified tabu

Current Solution (end of iteration 2)

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

Objective Value = **18**

Tabu List

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

Swap	Move Value
1, 3	-2
2, 4	-4
7, 6	-6
4, 5	-7
5, 3	-9

Iteration 3

- Swap by the top candidate move (2, 4)
- Although (4,5) is tabu, it is much superior to other solutions; thus, **aspiration criterion** overrides the tabu classification; pick (4,5)

Current Solution (end of iteration 3)

4	**2**	7	1	5	6	3

Objective Value = **14**

Tabu List

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

Swap	Move Value
4, 5	6
5, 3	2
7, 1	0
1, 3	-3
2, 6	-6

Iteration 4

- Swap by the top candidate move (4, 5)
- Pick (7, 1); continue the process

Current Solution (end of iteration 4)

5	2	7	1	**4**	6	3

Objective Value = **20**

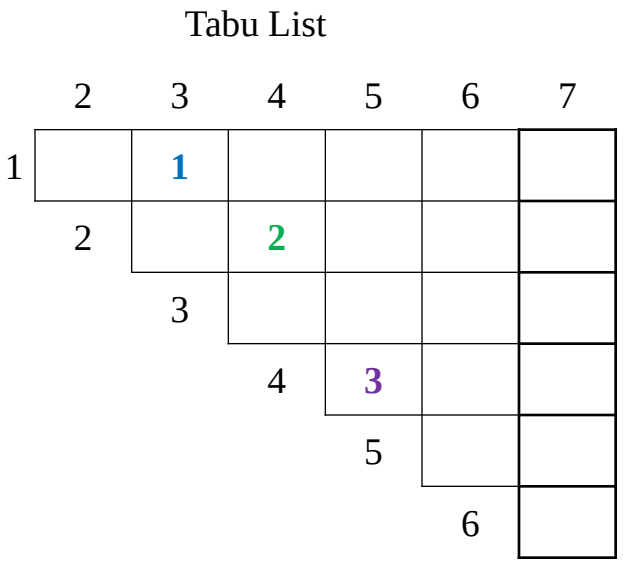
Tabu List

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

Swap	Move Value
7, 1	0
4, 3	-3
6, 3	-5
5, 4	-6
2, 6	-8

Recency-based memory

- Keeps track of solution attributes that have changed during the recent past iterations
- The value decreases as iteration proceeds



Frequency-based memory

- Measured by the counts of the number of occurrences of a particular move
- Long-term Memory
- Goal: to diversify the search, driving it into new regions
- Now, we proceed to iteration 26

Iteration 26: Frequency

- Use the lower diagonal of the tabu list to contain the frequency counts, whose sum is 25

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

Iteration 26

Swap	Move Value	Penalized by Frequency
1, 4	3	2
2, 4	-1	-6
3, 7	-3	-3
1, 6	-5	-5
6, 5	-4	-6

- The most improving move is (1, 4), which is classified tabu
- The second best solution (2, 4), however, is penalized (-1 for each previous move; a specified constant) for being used frequently in history (5 times among 25 iterations)
- Thus, we pick (3, 7)

Short-Term Memory

- The main goal of the short-term memory is to avoid reversal of moves and cycling
- The most common implementation of the short-term memory is based on move attributes and the recency of the moves

Example 1

- Permutation problem

  After a move that exchanges the positions of element *i* and *j* in a sequence, we would like to prevent elements *i* and *j* from exchanging positions in the next *TabuTenure* iterations

  – Attributes to record: *i* and *j*
  – Tabu activation rule: move ($i \leftrightarrow j$) is tabu if both *i* and *j* are tabu-active

Example 2

- Binary integer programming problem

  After a move that changes the value of x_i from 0 to 1, we would like to prevent x_i from taking the value of 0 in the next *TabuTenure* iterations

  - Attribute to record: *i*
  - Tabu activation rule: move ($x_i \leftarrow 0$) is tabu if *i* is tabu-active

Example 3

- Knapsack problem

  After a move that drops element *i* from and adds element *j* to the current solution, we would like to prevent element *i* from being added to the solution in the next *TabuAddTenure* iterations and prevent element *j* from being dropped from the solution in the next *TabuDropTenure* iterations

  - Attributes to record: *i* and *j*
  - Tabu activation rules:
    - move (Add *i*) is tabu if *i* is tabu-active
    - move (Drop *j*) is tabu if *j* is tabu-active

Tabu Tenure

- The length of time during which a certain move is classified as tabu
- Static
  - Constant
  - Function of problem dimension
- Dynamic
  - Vary (randomly or by systematic pattern) between upper and lower bounds

Aspiration Criteria

- Conditions that can override tabu restriction

Examples:

- Significant improvement
- Least tabu
- Search direction

Intensification

- Store and exploit elite (good) solutions

Implementations:

- Each time the search progress slows, use elite solutions to re-initiate the search
- Identify consistent attributes frequently found in elite solutions; “lock in” the attributes
  - Be aware: Elite solutions may have attributes against each other

Diversification

- Explore regions of the search space which have not been (or less frequently) visited so far

Implementations:

- Forcing a few rarely used components and restarting the search from this point
- Biasing the evaluation of possible moves by changing the objective value related to frequency

Critical Choices

- Neighborhood structure
- Candidate list
- Recency and tabu tenure
- Frequency and penalty
- Aspiration criteria
- Underlying search method

Parallel Tabu Search

- Perform Tabu searches independently in parallel, starting with different initial solutions
- Perform Tabu searches in parallel, that cooperate with each other
  - Share the short-term and long-term memory

Conclusion

- Tabu Search utilizes short and long term memory; the structure of memory is crucial
- Memory space is an important issue
- Tabu Search is NOT population-based
- Tabu Search places less emphasis on randomness
- Tabu Search requires better understanding of the problem at hand