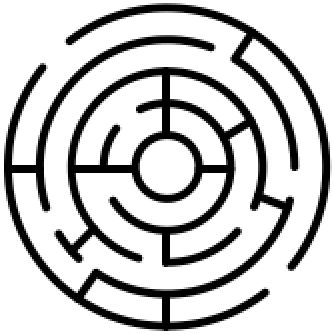
Tabu Search

Winter 2024

Motivation

- The search space being considered may often be discrete: combinatorial optimization
- If search has already visited a point in the search space, why waste computation time re-visiting and re-evaluating that point?
- Keeping track of every point and its evaluation result is often computationally expensive



What is Tabu

- The points being visited before should be made "**tabu**" (taboo)
- **Tabu** – socially or culturally proscribed
  - Forbidden to be used, mentioned, or approached because of social or cultural rather than legal prohibitions
  - Unacceptable to society or ordinary people

Examples of Tabu

- Exposure of body parts
- Restrictions on the use of offensive language and gesture
- Foods and beverages which people avoid
- Burial grounds or places of death
- Discrimination on the grounds of race, age, or gender

Definition

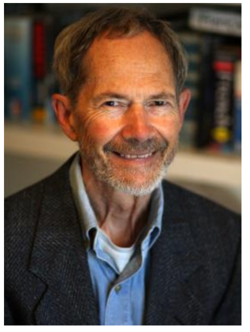
- Tabu search is based on introducing flexible memory structures in conjunction with strategic restrictions and aspiration levels as a means for exploiting search spaces
- Typically used in combinatorial optimization
- The overall approach is to avoid entrapment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited

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 values of variable(s) are changed (to immediate adjacent values) in a sorted list of discrete values.

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.

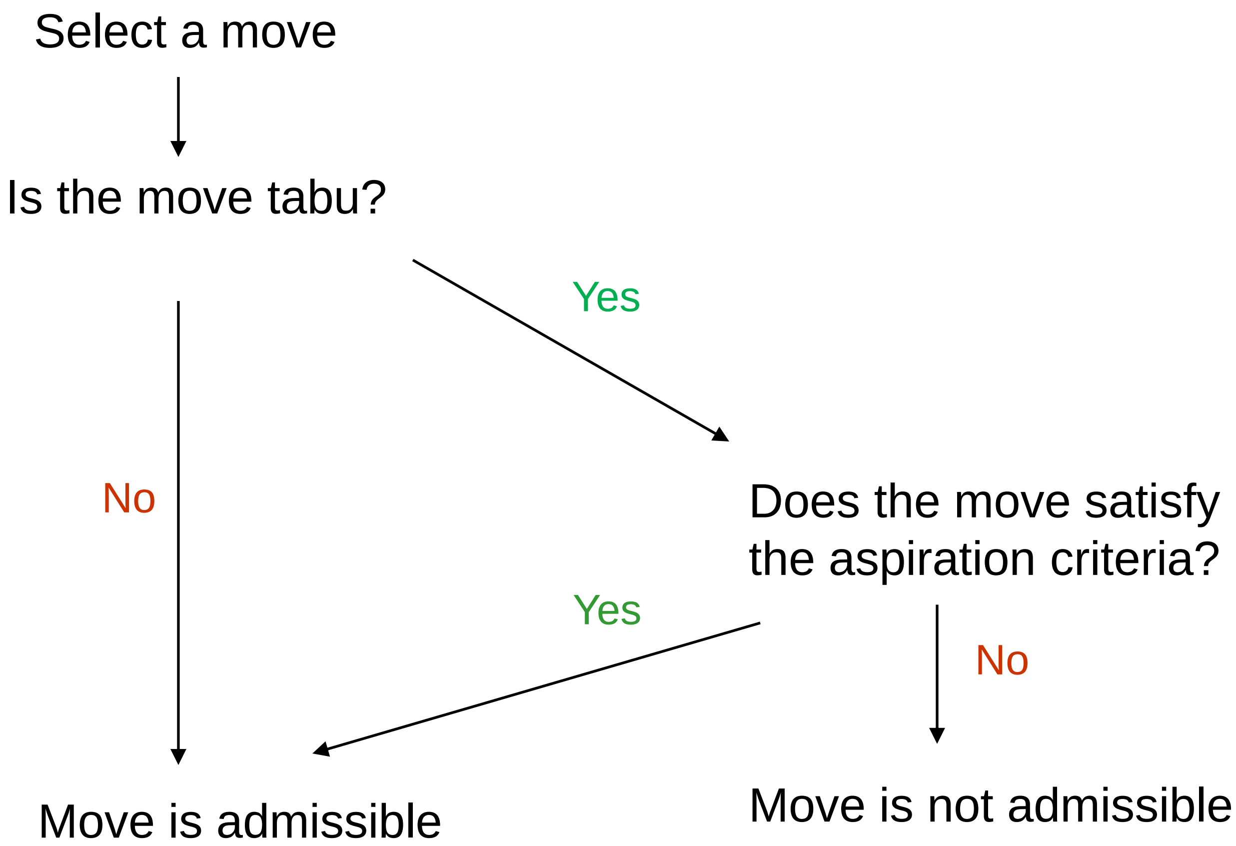


Prof. Fred Glover

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 (tabu status is overridden)
- Update tabu list if necessary
- Replace current solution with new solution if it is better
- Repeat

Tabu Decision



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

Illustrative Example

- We are working on a discrete optimization 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
- How will the solutions look like?

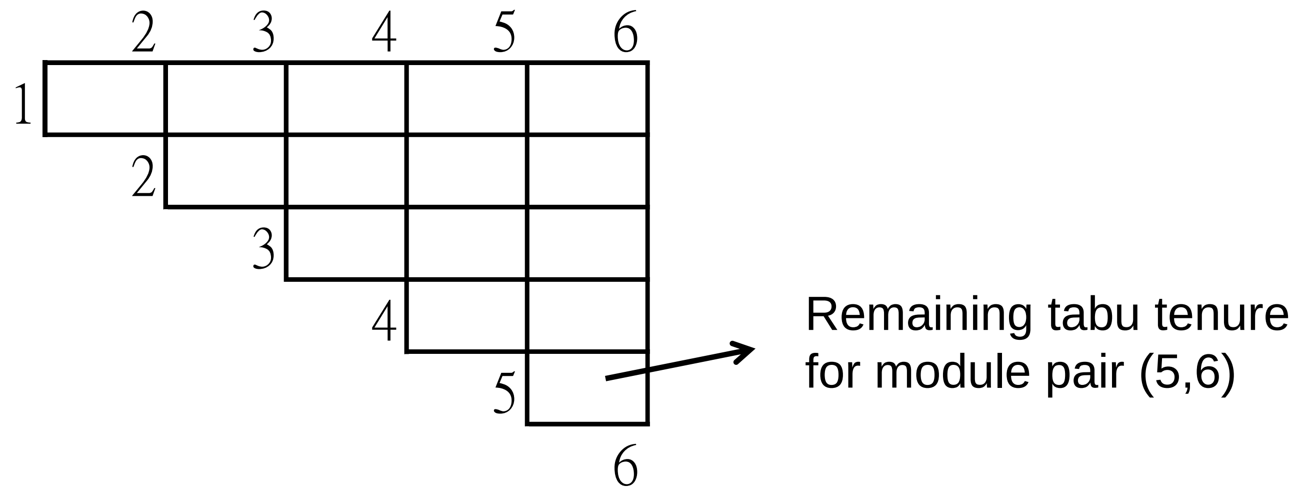
Solution and Swap

- 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
  - The move would result in a solution much better than any visited before
- This override condition is called “aspiration criterion”
- The aspiration criterion overrides a solution's tabu state, thereby accepting the otherwise-excluded solution

Basic Tabu Procedure

- Hereafter, we illustrate iterations of the basic tabu procedure; 5 moves at each iterations; tabu tenure = 3; aspiration criterion (>5)
- At iteration 0; Generate a start solution

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						

Iteration 1

- Generate 5 random swap moves (user-specified)
- Evaluate the candidate moves
- Swap by the top candidate move (5,4); update the tabu list

Swap	Obj. Value
5, 4	16
7, 4	14
3, 6	12
2, 3	10
4, 1	9

Previous Solution

2	5	7	3	4	6	1

Objective Value = 10

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						

Iteration 2

- Generate 5 random swap moves
- Evaluate the candidate moves
- Swap by the top candidate move (3, 1); update the tabu list

Swap	Obj. Value
3, 1	18
2, 3	17
3, 6	15
7, 1	14
6, 1	12

Current Solution (end of iteration 1)

2	4	7	3	5	6	1

Objective Value = **16**

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						

Iteration 3

- Generate 5 random swap moves; Evaluate the moves; swap by move (2, 4); update the tabu list
  - Why not (1, 3)?

  Move (1,3) has been listed as tabu

Swap	Obj. Value
1, 3	16
2, 4	14
7, 6	12
4, 5	11
5, 3	9

Current Solution (end of iteration 2)

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

Objective Value = **18**

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						

Iteration 4

- Generate 5 random swap moves; Evaluate the moves; swap by move (4, 5); update the tabu list
- Although (4,5) is tabu, it meets aspiration criterion (20-14=6 > 5); thus, still pick (4, 5)

Swap	Obj. Value
4, 5	20
5, 3	16
7, 1	14
1, 3	11
2, 6	8

Current Solution (end of iteration 3)

4	**2**	7	1	5	6	3

Objective Value = **14**

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						

Further discussion on iteration 4

- Why (4,5) leads to 20, not back to 16 in Iteration 1?

Current Solution

2	5	7	3	4	6	1

Objective Value = 10

Iteration 0

Current Solution (end of iteration 1)

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

Objective Value = **16**

Iteration 1

Current Solution (end of iteration 2)

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

Objective Value = **18**

Iteration 2

Current Solution (end of iteration 3)

4	**2**	7	1	5	6	3

Objective Value = **14** → **20**

Iteration 3 ~ 4

Iteration 5

- Generate 5 random swap moves
- Continue for a pre-specified number of iterations

Swap	Obj. Value
7, 1	20
4, 3	17
6, 3	15
5, 4	14
2, 6	12

Recency-based memory

- We may keep track of solution attributes that have changed during the recent past iterations
- The value decreases as iteration proceeds, short-term memory

Tabu List

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

Frequency-based memory

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

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

- Suppose, the most improving move is (1, 4), which is classified tabu
- The second best solution (2, 4) is penalized (-1 for each previous move, pre-specified) for being used frequently in history (5 times among 25 iterations)
- Thus, we pick (3, 7)
- The penalty may be applied since the beginning

Swap	Obj. Value	Penalized by Frequency
1, 4	24	24-1=23
2, 4	23	23-5=18
3, 7	21	21-0=21
1, 6	18	18-0=18
6, 5	16	16-2=14

	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			

Why not memorize all the moves

- Be careful about the definition of “long-term memory”
- To avoid cycling, it may be good to memorize all the moves and their appearance sequence
- However, during progress, every new move has to be compared with previous moves
- It would take large memory and runtime

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

Application 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

Application 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

Application 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

Possibilities:

1. Better than the currently known best solution.
2. Significant improvement
3. Least tabu
4. Search direction (positive or negative)

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:

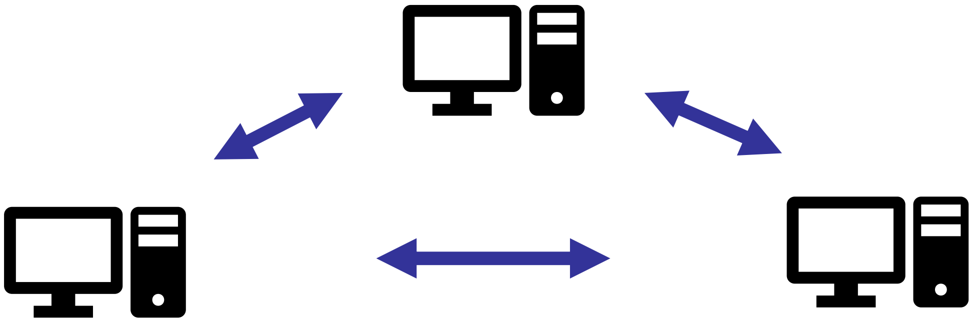
- Force a few rarely used attributes and restarting the search from this point
- Bias the evaluation of possible moves by changing the objective value related to frequency: penalize options that have been frequently chosen.

Critical Choices of TS

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

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 (recency) and long-term (frequency) memory



Conclusions

- Tabu Search is individual-based, focusing on discrete search space
- Tabu Search utilizes short and long term memory
- Structure of memory is crucial
- Tabu Search places less emphasis on randomness
- Tabu Search requires better understanding of the problem at hand
  → aspiration criteria and frequency penalty