Tabu Search
Part 3: Minimum k-Tree Problem Example

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Definition
- Initial Solution
- Neighborhood Structure and Move Mechanism
- Tabu Structure
- Illustrative Tabu Structure
- A First Level Tabu Search Approach
- Diversification
- References

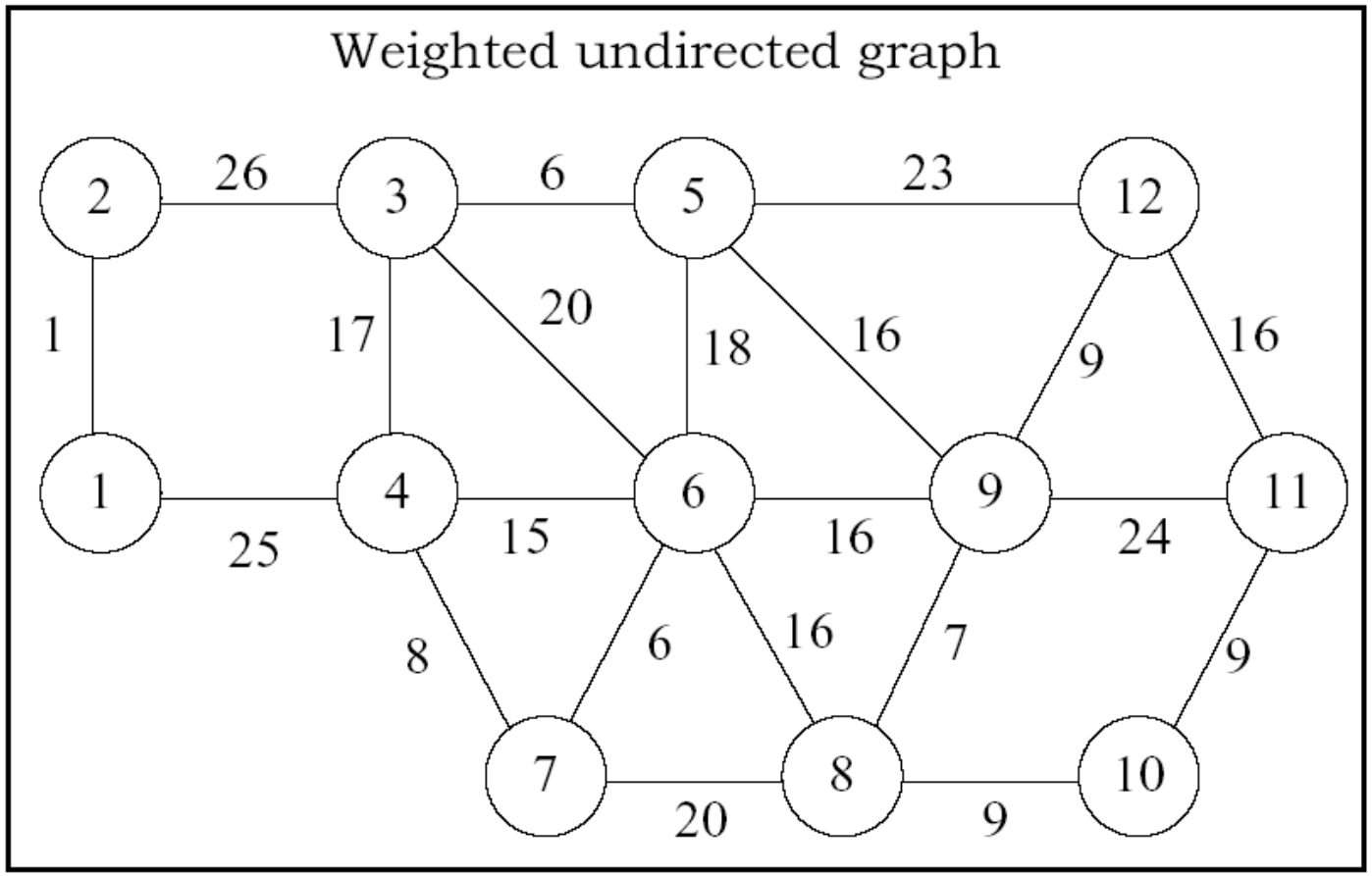
Definition

Definition

- **Problem Definition:**
  - The Minimum k-Tree problem seeks a tree consisting of k edges in a graph so that the sum of the weights of these edges is minimum.
  - A tree is a set of edges that contains no cycles, i.e., that contains no paths that start and end at the same node (without retracing any edges).

Definition

- An instance of the minimum 4-tree problem:



Definition

- An instance of the minimum 4-tree problem:
  - where nodes are shown as numbered circles, and
  - edges are shown as lines that join pairs of nodes (the two "endpoint" nodes that determine the edge)
  - Edge weights are shown as the numbers attached to these lines.

Tabu search components

- **Tabu search components:**
  - Search Space
  - Neighborhood Structure and Move Mechanism
  - Tabu Structure
  - Aspiration Criteria
  - Candidate List Strategy
  - Termination Criteria
  - Initial Solution

  - Intensification
  - Diversification

Search Space

Search Space

- Search Space
  - All trees that consisting of k edges in a graph that contains no cycles, i.e., that contains no paths that start and end at the same node (without retracing any edges).

Neighborhood Structure and Move Mechanism

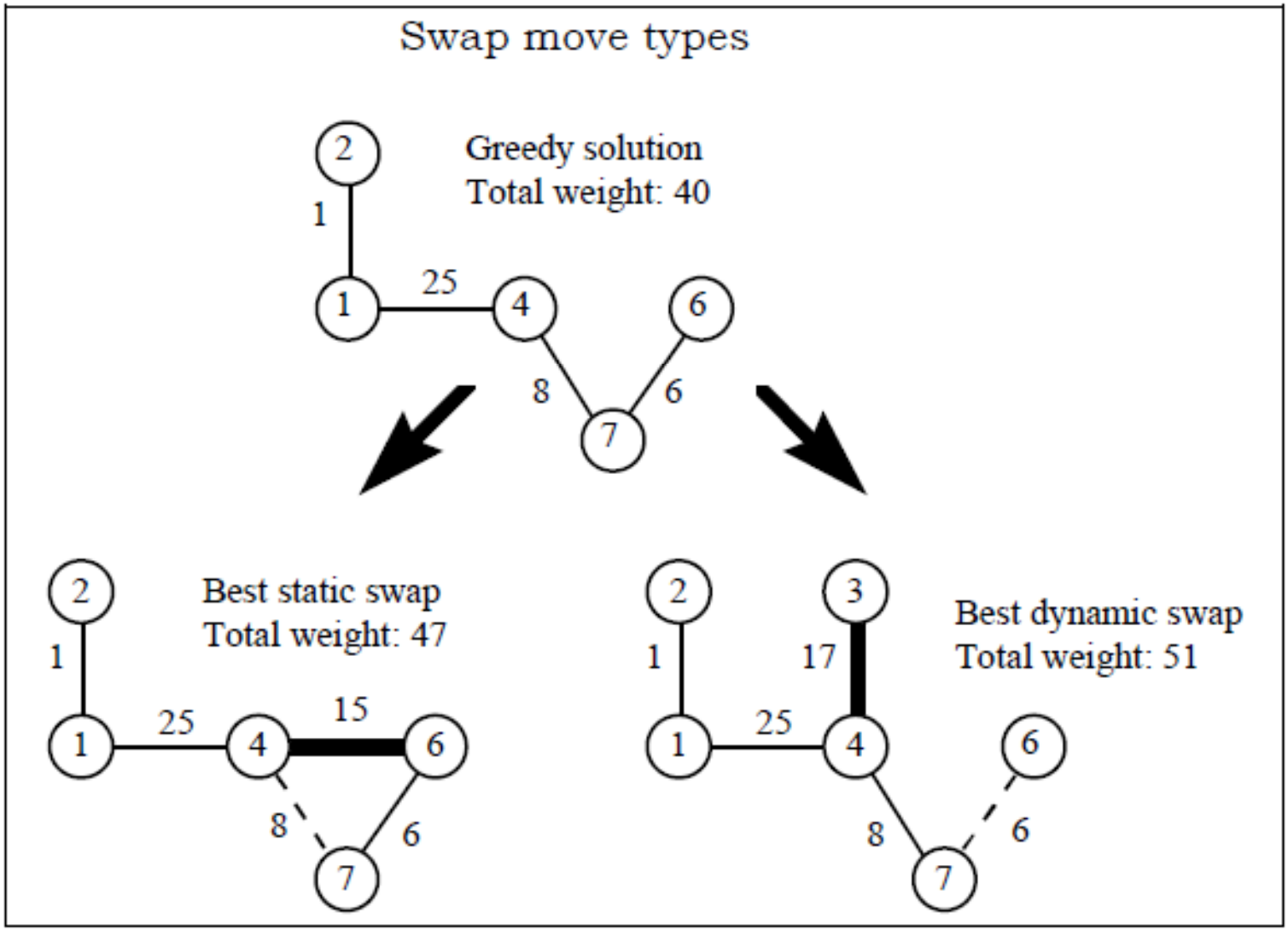
Neighborhood Structure

- Neighborhood Structure
  - Replacing a selected edge in the tree by another selected edge outside the tree makes a neighbor solution, subject to requiring that the resulting subgraph is also a tree.
  - The move mechanism is defined by edge-swapping

The Move Mechanism

- The swap move mechanism is used
- Types of such edge swaps:
  - Static swap:
    - one that maintains the current nodes of the tree unchanged
  - Dynamic swap:
    - one that results in replacing a node of the tree by a new node

The Move Mechanism



The Move Mechanism

- The best move of both types is the **static swap** of where for our present illustration
- We are defining best solely in terms of the change on the objective function value.
- Since this best move results in an increase of the total weight of the current solution, the execution of such move abandons the rules of a descent approach and sets the stage for a tabu search process.

The Move Mechanism

- The feasibility restriction that requires a tree to be produced at each step is particular to this illustration
- In general the TS methodology may include search trajectories that violate various types of feasibility conditions.

The Move Mechanism

- The next step is to choose the key attributes that will be used for the tabu structure.
- In problems where the moves are defined by adding and deleting elements, the labels of these elements can be used as the attributes for enforcing tabu status.

Tabu Structure

Choosing Tabu Classifications

- The tabu structure can be designed to treat added and dropped elements differently.
- Suppose for example that after choosing the static swap, which adds edge (4,6) and drops edge (4,7), a tabu status is assigned to both of these edges.
- Then one possibility is to classify both of these edges **tabu-active** for the same number of iterations.

Choosing Tabu Classifications

- The tabu-active status has different meanings depending on whether the edge is added or dropped.
- **For an added edge**,
  - tabu-active means that this **edge is not allowed to be dropped** from the current tree for the number of iterations that defines its tabu tenure.
- **For a dropped edge**,
  - on the other hand, tabu-active means **the edge is not allowed to be included** in the current solution during its tabu tenure.

Choosing Tabu Classifications

- Since there are many more edges outside the tree than in the tree,
  - it seems reasonable to implement a tabu structure that keeps a recently dropped edge tabu-active for a longer period of time than a recently added edge.
- Notice also that for this problem the tabu-active period for added edges is bounded by *k*,
  - since if no added edge is allowed to be dropped for k iterations, then within k steps all available moves will be classified tabu.

Choosing Tabu Classifications

- **The tabu-active structure may in fact prevent the search from visiting solutions that have not been examined yet.**
- Tabu tenure of the edges
  - **dropped** edges are kept tabu active for **2 iterations**,
  - **added edges** are kept tabu-active for only **1 iteration**.
- The number of iterations an edge is kept tabu-active is called the tabu tenure of the edge.
- Also assume that we define a swap move to be tabu if either its added or dropped edge is tabu active.

Aspiration Criteria

Aspiration Criterion

- In some situations, however, additional precautions must be taken to avoid missing good solutions.
- These strategies are known as **aspiration criteria**.
- Alternative forms of aspiration criteria are very important in tabu search.

Aspiration Criterion

- **Aspiration criterion for Min k-Tree problem**
  - if the tabu solution encountered at the current step instead had a weight of 39, which is better than the best weight of 40 so far seen, then we would allow the tabu of this solution to be overridden and consider the solution admissible to be visited.
  - The aspiration criterion that applies in this case is called the **improved-best aspiration criterion**.

Illustrative Tabu Structure

- In our preceding discussion:
  - We consider a swap move tabu if either its added edge or its dropped edge is tabu-active.
- However, we could instead consider that a swap move is tabu only if both its added and dropped edges are tabu-active.

Candidate List Strategy

Candidate List Strategy

- We examine the full neighborhood of available edge swaps at each iteration, and always choose the best that is not tabu

Termination Criteria

Termination Criteria

- For simplicity, we select an arbitrary stopping rule that ends the search at iteration 10.

Initial Solution

Initial Solution

- Initial Solution
  - a greedy procedure is used to find an initial solution
- The greedy construction starts by choosing the edge (i, j) with the smallest weight in the graph,
  - where i and j are the indexes of the nodes that are the endpoints of the edge.
- The remaining k-1 edges are chosen successively to minimize the increase in total weight at each step,
  - where the edges considered meet exactly one node from those that are endpoints of edges previously chosen.

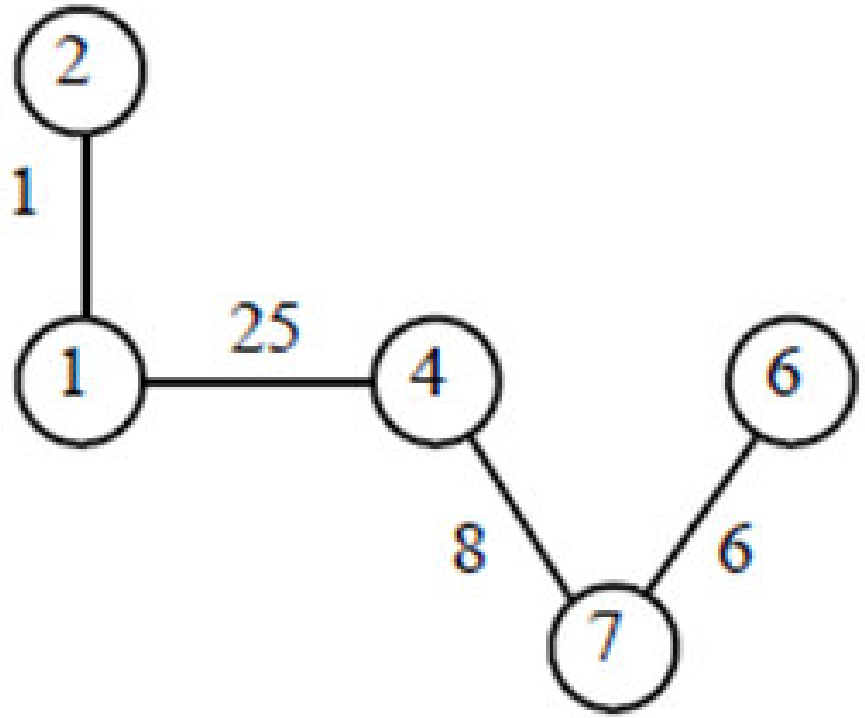
Initial Solution

- For k = 4, the greedy construction performs the following steps:

Greedy construction			
Step	Candidates	Selection	Total Weight
1	(1,2)	(1,2)	1
2	(1,4), (2,3)	(1,4)	26
3	(2,3), (3,4), (4,6), (4,7)	(4,7)	34
4	(2,3), (3,4), (4,6), (6,7), (7,8)	(6,7)	40

Initial Solution

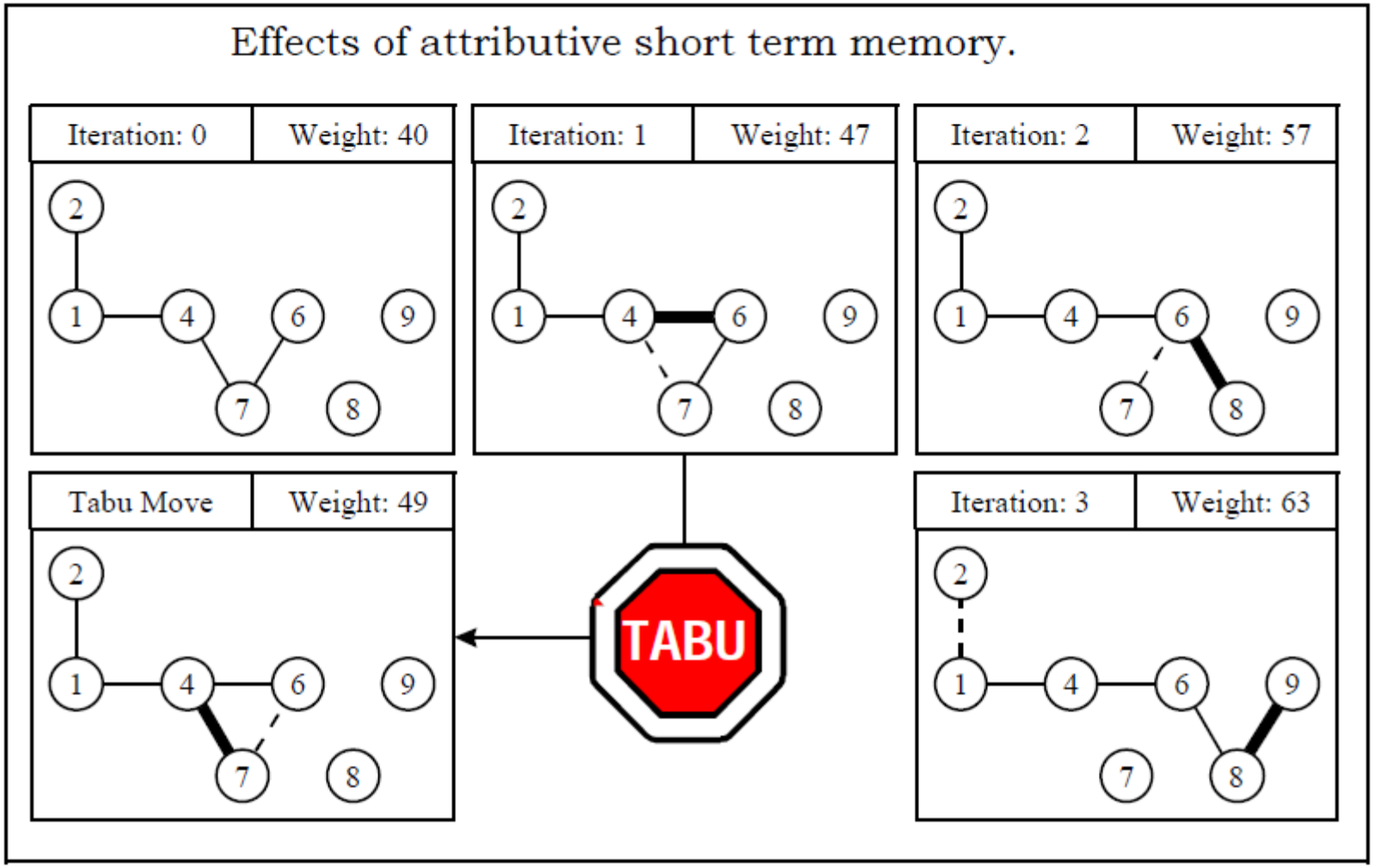
- Greedy solution with total weight 40



Search Approach

Search Approach

- The first three moves is shown :



Search Approach

- The first three moves are as shown in this Table:

TS iterations					
Iteration	Tabu-active net tenure		Add	Drop	Weight
	1	2			
1			(4,6)	(4,7)	47
2	(4,6)	(4,7)	(6,8)	(6,7)	57
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	63

Search Approach

- At iteration 2, the move that now adds (4,7) and drops (4,6)) is **clearly tabu**, since both of its edges are tabu-active at iteration 2 (the reversal of the move of iteration 1)
- In addition, the move that adds (4,7) and drops (6,7) is also classified tabu, because it contains the tabu-active edge (4,7) (with a net tenure of 2).
  - This move leads to a solution with a total weight of 49, a solution that clearly **has not been visited before**

Search Approach

- The tabu-active classification of (4,7) has modified the **original neighborhood** of the solution at iteration 2, and has forced the search to choose a move with an inferior objective function value (i.e., the one with a total weight of 57).
- In this case, excluding the solution with a total weight of 49 has no effect on the quality of the best solution found (since we have already obtained one with a weight of 40).

Search Approach

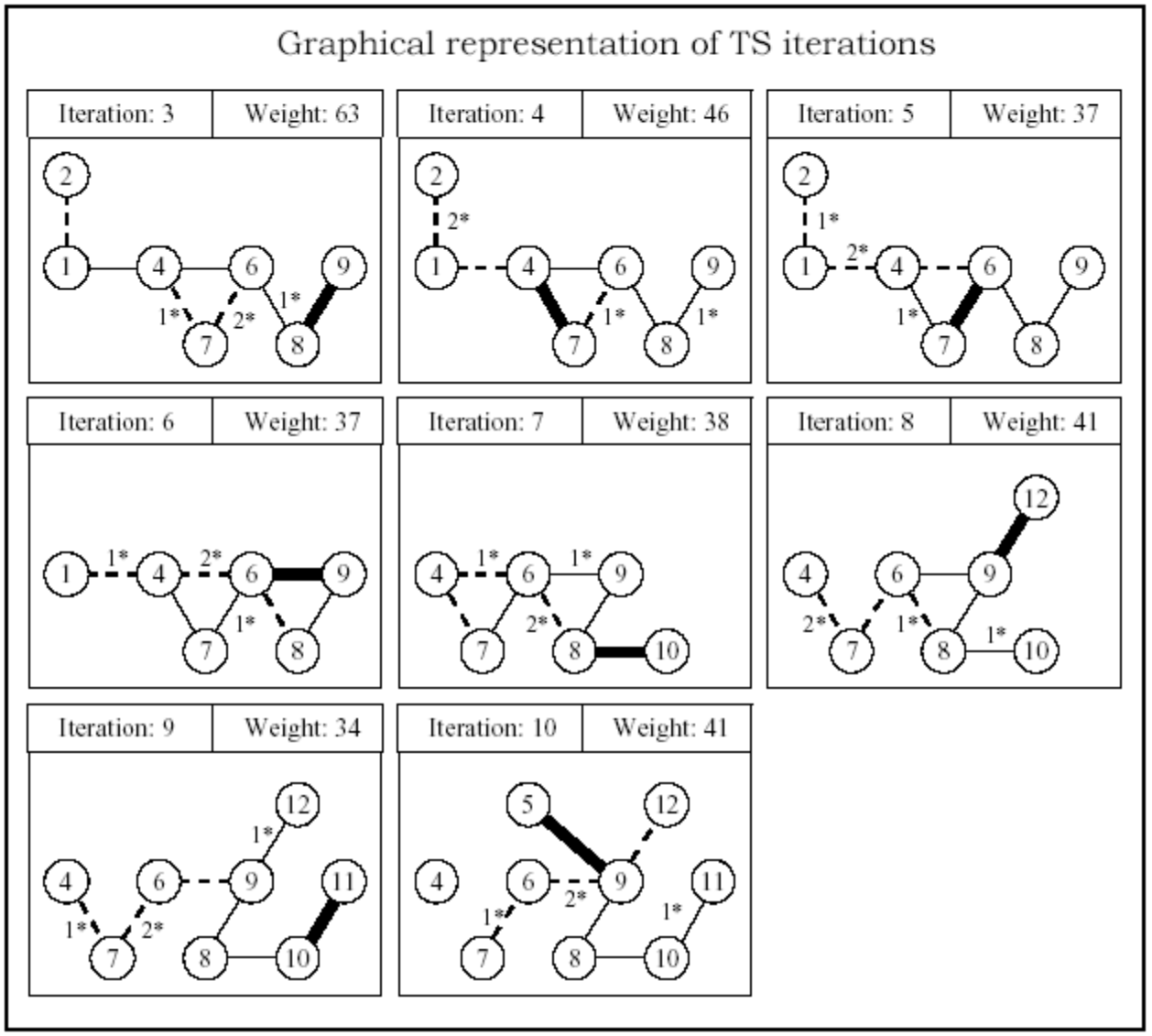
- We continue the solution with a weight of 63 as shown previously, which was obtained at iteration 3.
- At each step we select the least weight non-tabu move from those available, and use the improved-best aspiration criterion to allow a move to be considered admissible in spite of leading to a tabu solution.
- The outcome leads to the series of solutions shown in next slide, which continues from iteration 3, just executed.

Search Approach

Iterations of a first level TS procedure						
Iteration	Tabu-active net tenure		Add	Drop	Move Value	Weight
	1	2				
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	6	63
4	(6,7), (8,9)	(1,2)	(4,7)	(1,4)	-17	46
5	(1,2), (4,7)	(1,4)	(6,7)	(4,6)	-9	37*
6	(1,4), (6,7)	(4,6)	(6,9)	(6,8)	0	37
7	(4,6), (6,9)	(6,8)	(8,10)	(4,7)	1	38
8	(6,8), (8,10)	(4,7)	(9,12)	(6,7)	3	41
9	(4,7), (9,12)	(6,7)	(10,11)	(6,9)	-7	34*
10	(6,7), (10,11)	(6,9)	(5,9)	(9,12)	7	41

Tabu Search: Part 3

Search Phase



Search Phase

- To identifying
  - We identify the **dropped edge** from the immediately preceding step as a dotted line which is labeled 2*, To indicate its current net tabu tenure of 2.
  - We identify the **dropped edge** from one further step back by a dotted line which is labeled 1*, to indicate its current net tabu tenure of 1.
  - The edge that was added on the immediately preceding step is also labeled 1 to indicate that it likewise has a current net tabu tenure of 1.

Search Phase

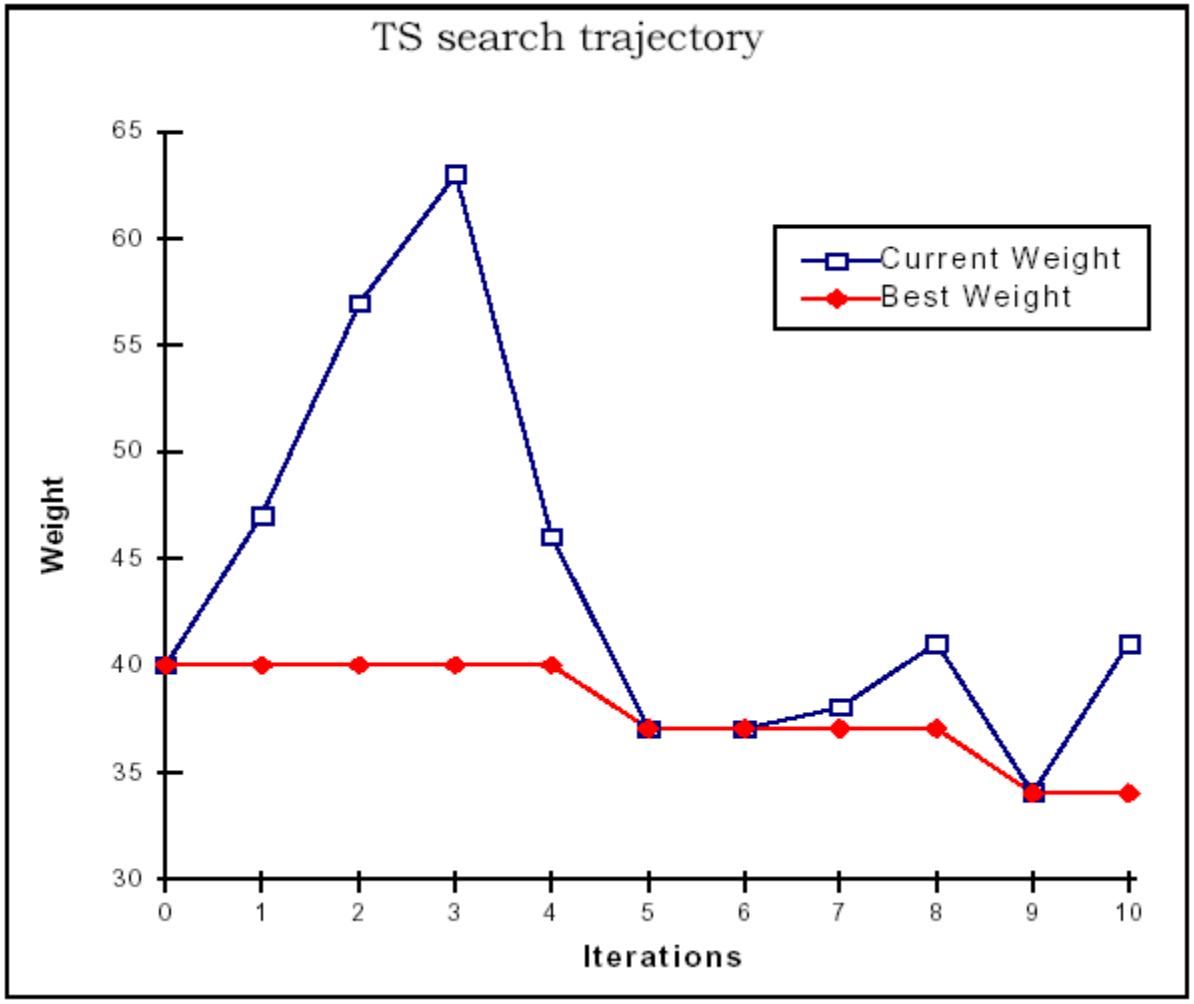
- Thus the edges that are labeled with tabu tenures are those which are currently tabu-active, and which are excluded from being chosen by a move of the current iteration (unless permitted to be chosen by the aspiration criterion).
- The method continues to generate different solutions, and over time the best known solution (denoted by an asterisk) progressively improves.
- In fact, it can be verified for this simple example that the solution obtained at iteration 9 is optimal.

Search Phase

- In general, of course, there is no known way to verify optimality in polynomial time for difficult discrete optimization problems,
  - i.e., those that fall in the class called **NP-hard**.
  - The Min *k-Tree* problem is one of these.

- It may be noted that at iteration 6 the method selected a move with a move value of zero.
  - Nevertheless, the configuration of the current solution changes after the execution of this move.

Search Phase



Search Phase

- One natural way to apply TS is to periodically discontinue its progress, particularly if its rate of finding new best solutions falls below a preferred level, and **to restart** the method by a process designated to generate a new sequence of solutions.
- Classical **restarting procedures** based on randomization evidently can be used for this purpose, but TS often derives an advantage by employing more strategic forms of restarting.
- We illustrate a simple instance of such a restarting procedure, which also serves to introduce a useful memory concept.

Diversification

Diversification

- Since the solution at **iteration 9** happens to be optimal, we are interested in the effect of restarting before this solution is found.
- Assume we had chosen to restart after **iteration 7**, without yet reaching an optimal solution.
- Then the solutions that correspond to critical events are the **initial solution** and the **solutions of iterations 5 and 6**.

Diversification

- We treat these three solutions in aggregate by combining their edges, to create a subgraph that consists of the edges (1,2),(1,4), (4,7), (6,7), (6,8), (8,9) and (6,9).
- **Frequency-based memory** refines this representation by accounting for the number of times each edge appears in the critical solutions, and allows the inclusion of additional weighting factors.

Diversification

- To execute a restarting procedure, we penalize the inclusion of the edges of this subgraph at various steps of constructing the new solution.
- It is usually preferable to apply this penalty process at **early steps**, implicitly allowing the penalty function to decay rapidly as the number of steps increases.

Diversification

- We will use the memory embodied in the subgraph of **penalized edges** by introducing a large penalty that effectively excludes all these edges from consideration on **the first two steps** of constructing the new solution.
- Then, because the construction involves four steps in total, we will not activate the critical event memory on subsequent construction steps.
- Applying this approach, we restart the method by first choosing edge (3,5), which is the minimum weight edge not in the penalized subgraph.

Diversification

- This choice and the remaining choices that generate the new starting solution are shown:

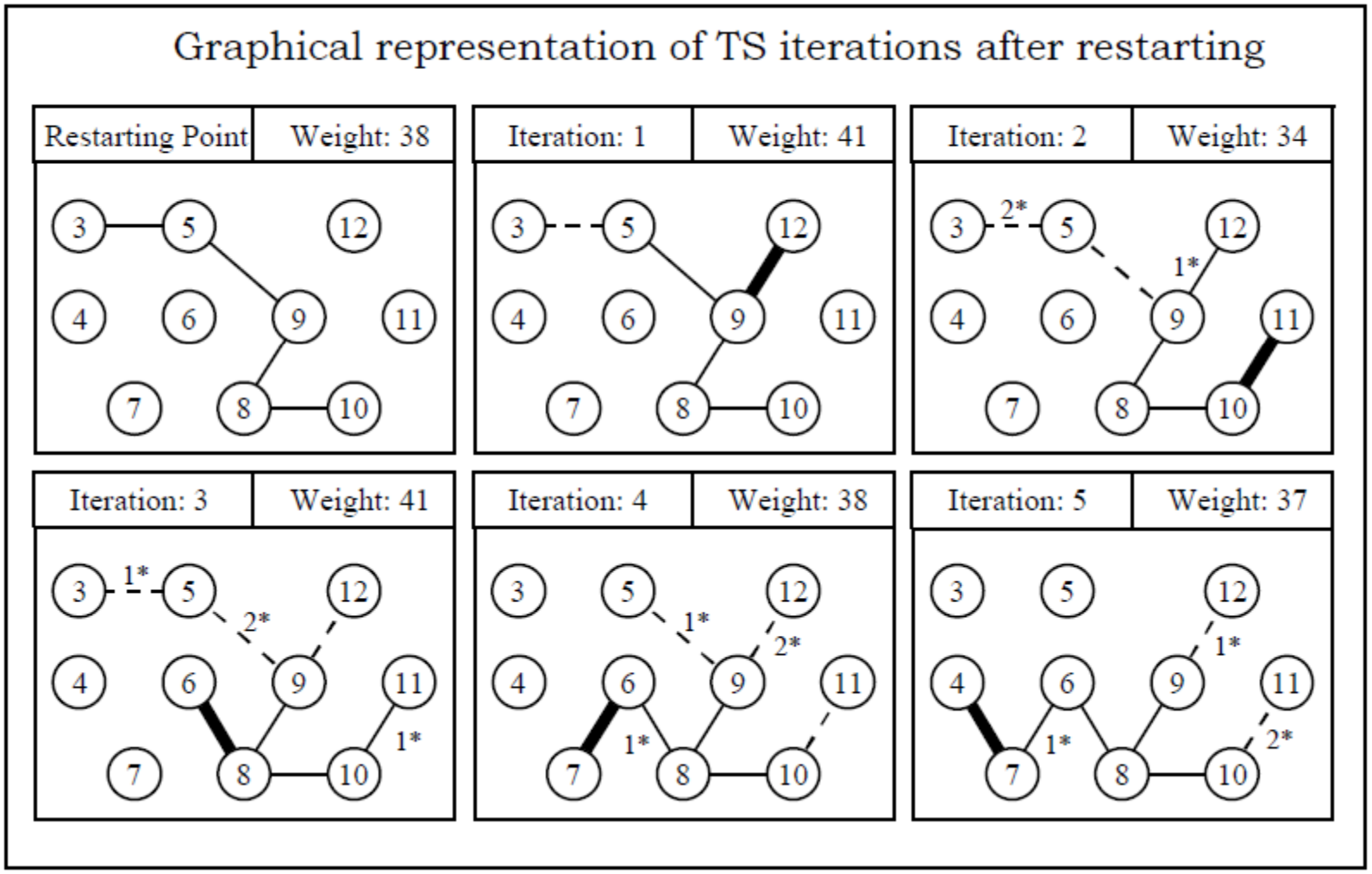
Restarting procedure			
Step	Candidates	Selection	Total Weight
1	(3,5)	(3, 5)	6
2	(2,3), (3,4), (3,6), (5,6), (5,9), (5,12)	(5, 9)	22
3	(2,3), (3,4), (3,6), (5,6), (5,12), (6,9), (8,9), (9,12)	(8, 9)	29
4	(2,3), (3,4), (3,6), (5,6), (5,12), (6,8), (6,9), (7,8), (8,10), (9,12)	(8, 10)	38

Diversification

- Beginning from the solution constructed and applying the first level TS procedure exactly as it was applied on the first pass, generates the sequence of solutions shown in the next Table and Figure
- Again, we have arbitrarily limited the total number of iterations, in this case to 5.

TS iterations following restarting						
Iteration	Tabu-active net tenure		Add	Drop	Move Value	Weight
	1	2				
1			(9,12)	(3,5)	3	41
2	(9,12)	(3,5)	(10,11)	(5,9)	-7	34*
3	(3,5), (10,11)	(5,9)	(6,8)	(9,12)	7	41
4	(5,9), (6,8)	(9,12)	(6,7)	(10,11)	-3	38
5	(9,12), (6,7)	(10,11)	(4,7)	(8,10)	-1	37

Diversification



Diversification

- It is interesting to note that the restarting procedure generates a better solution (with a total weight of 38) than the initial solution generated during the first construction (with a total weight of 40).
- Also, the restarting solution contains 2 "optimal edges" (i.e., edges that appear in the optimal tree).
- This starting solution allows the search trajectory to find the optimal solution in only two iterations, illustrating the benefits of applying a critical event memory within a restarting strategy.

Diversification

- Related memory structures can also be valuable for strategies that drive the search into new regions by "partial restarting" or by directly continuing a current trajectory with modified decision rules.

References

References

- F. Glover and M. Laguna, "**Tabu Search**," In Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 194-208 (2002).

The End

Diversification

- **Critical Event Memory** in tabu search monitors the occurrence of certain critical events during the search, and establishes a memory that constitutes an aggregate summary of these events.
- For our current example, where we seek to generate a new starting solution, a critical event that is clearly relevant is the generation of the previous starting solution.
- Correspondingly, if we apply a restarting procedure multiple times, the steps of generating all preceding starting solutions naturally qualify as critical events.
- That is, we would prefer to depart from these solutions in some significant manner as we generate other starting solutions.

Diversification

- Different degrees of departure, representing different levels of **diversification**, can be achieved by defining solutions that correspond to critical events in different ways (and by activating **critical event memory** by different rules)
- In the present setting we consider it important that new starting solutions not only differ from preceding starting solutions, but that they also differ from other solutions generated during previous passes.

Diversification

- Method 1:
  - One possibility is to use a **blanket approach** that considers each complete solution previously generated to represent a critical event.
- Method 2:
  - It is quite sufficient (and, sometimes preferable) to isolate a smaller set of solutions.

Diversification

- For the current example, therefore, we will specify that the critical events of interest consist of generating not only the starting solution of the previous pass(es), but also each subsequent solution that represents a "**local TS optimum**,"
  - i.e. whose objective function value is better (or no worse) than that of the solution immediately before and after it.
- Using this simple definition we see that four solutions qualify as critical (i.e., are generated by the indicated critical events) in the first solution pass of our example:
  - the initial solution and the solutions found at iterations 5, 6 and 9 (with weights of 40, 37, 37 and 34, respectively).