Tabu Search Foundation: Additional Aspects of Short Term Memory

Tabu Search and Candidate List Strategy

For situations where N*(x) is larger or its elements are expensive to evaluate, candidate list strategies are essential to restrict the number of solutions examined on a given iteration

Efficient rules for generating and evaluating good candidates are critical to the search process

Candidate List Strategies

Focus on rules for constructing explicit candidate lists that are context-independent

A preferable measure of performance for a given candidate list strategy is the quality of the best solution found given a specified amount of computer time

Candidate List Strategies

Aspiration Plus

Aspiration Plus strategy establishes a threshold for the quality of a move, based on the history of the search pattern

The procedure operates by examining moves until finding one that satisfies this threshold

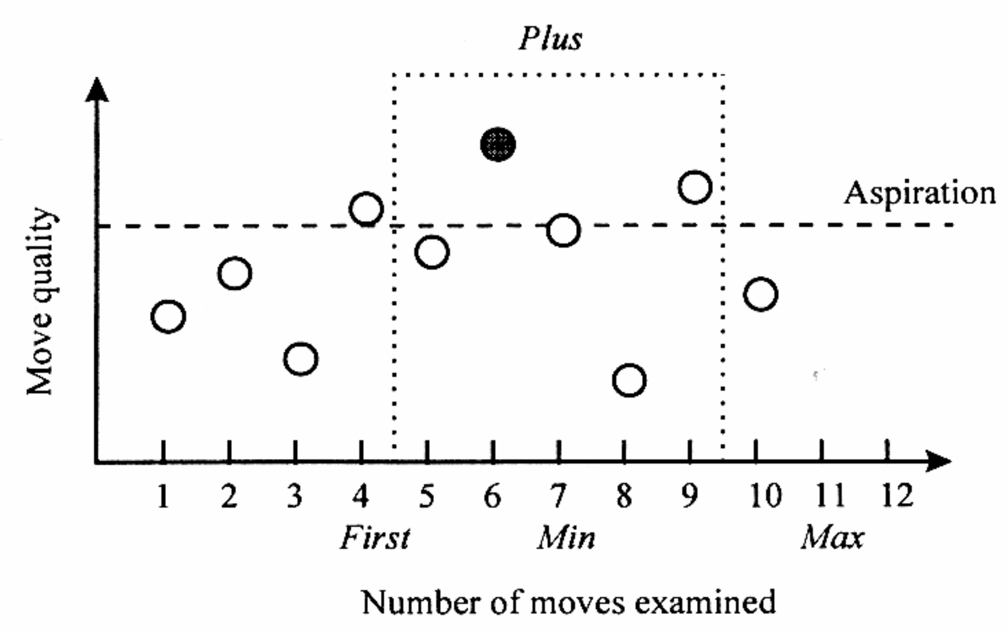
Upon reaching this point, additional moves are examined, equal in number to the selected value *Plus*, and the best move overall is selected

To ensure that neither too few nor too many moves are considered, this rule is qualified to require that at least *Min* moves and at most *Max* moves are examined

See Fig. 3.1

Candidate List Strategies

Fig. 3.1 Aspiration *Plus* strategy.



Candidate List Strategies

Elite Candidate List

The Elite Candidate List approach first builds a Master List by examining all (or relatively large number of) moves, selecting the *k* best moves encountered

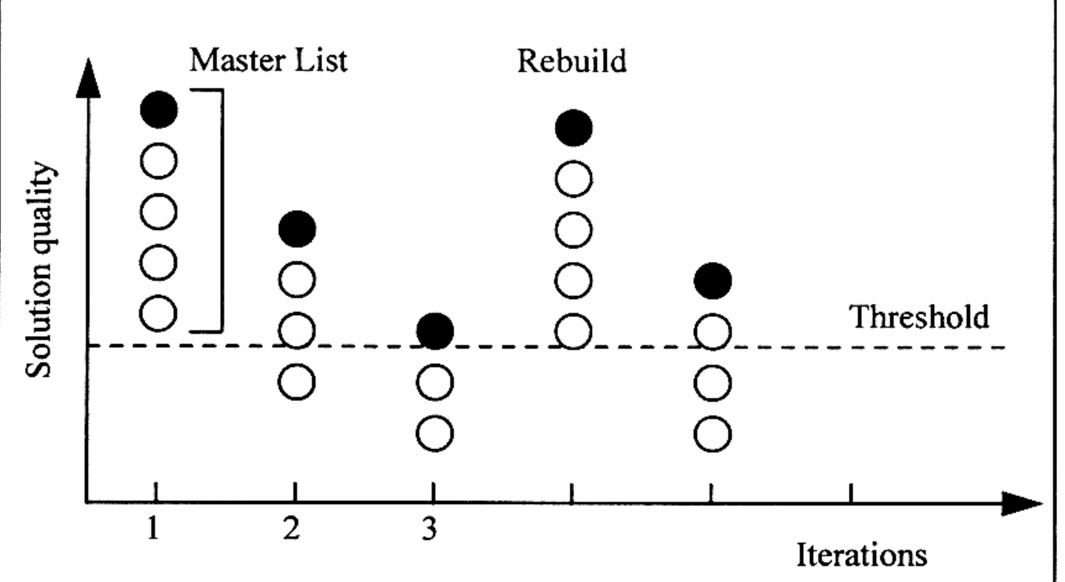
Then at each subsequent iteration, the current best move from the Master List is chosen to be executed, continuing until such a move falls below a given quality threshold, or until given number of iterations have elapsed

Then a new Master List is constructed and the process repeats

See Fig. 3.2

Candidate List Strategies

Fig. 3.2 Elite candidate list strategy.



Candidate List Strategies

Successive Filter Strategy

Moves can often be broken into component operations, and the set of moves examined can be reduced by restricting consideration to those that yield high quality outcomes for each operation separately

An exchange move: “add component” and “drop component” (100*100)

Restrict exchanges to a relatively small subset of “best add” and “best drop” (8*8)

Candidate List Strategies

Sequential Fan Candidate List

Sequential fan candidate list is highly exploitable by parallel processing

It generates some *p* best alternative moves at a given step, and then create a fan of solution streams, one for each alternative

The several best available moves for each stream are again examined, and only the *p* best moves overall provide the *p* new streams at the next step

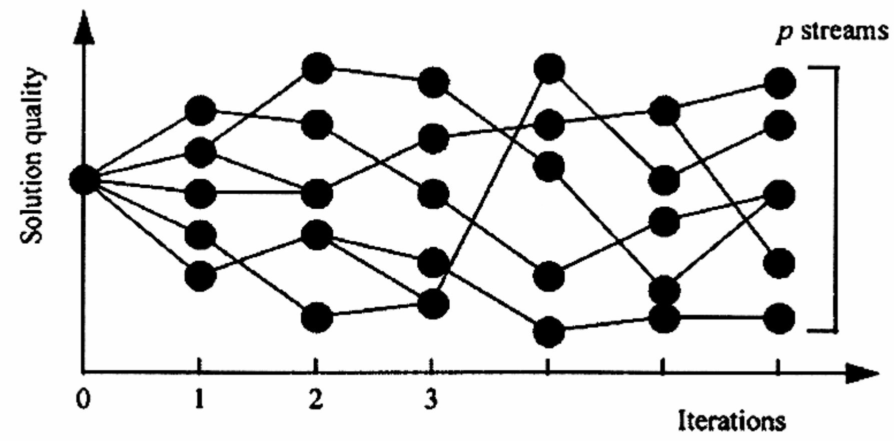
See Fig. 3.3

The sequential fan approach can be applied using *one type of move* to create a set of initial solutions, and then can continue using *another type of move* to generate additional solutions

The best moves from this solution are used to generate *p* different solutions

Candidate List Strategies

Fig. 3.3 Sequential fan candidate list.



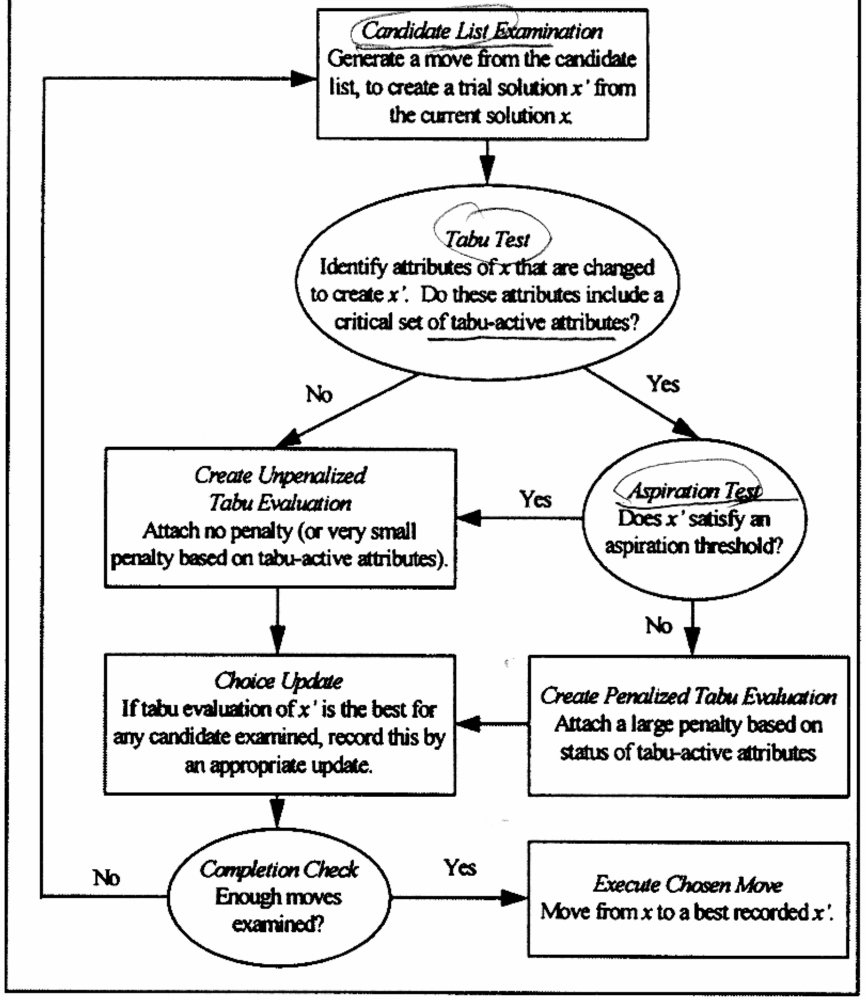
Connection between Candidate Lists, Tabu Status and Aspiration Criteria

Exclude (not admissible) tabu moves from being selected, subject to the outcome of aspiration tests

Fig. 3.4 expresses tabu restrictions in terms of penalized evaluation (admissible with penalty)

Connection between Candidate Lists, Tabu Status and Aspiration Criteria

Fig. 3.4 Short term memory operation.


Candidate List Examination
Generate a move from the candidate list, to create a trial solution x' from the current solution x.
Tabu Test
Identify attributes of x that are changed to create x'. Do these attributes include a critical set of tabu-active attributes?
No
Yes
Create Unpenalized Tabu Evaluation
Attach no penalty (or very small penalty based on tabu-active attributes).
Yes
Aspiration Test
Does x' satisfy an aspiration threshold?
No
Choice Update
If tabu evaluation of x' is the best for any candidate examined, record this by an appropriate update.
Create Penalized Tabu Evaluation
Attach a large penalty based on status of tabu-active attributes
No
Completion Check
Enough moves examined?
Yes
Execute Chosen Move
Move from x to a best recorded x'.

Logical Restructuring

Logical restructuring is an important element of adaptive memory solution approach, which gives a connection between short and long term strategies

The goal is to exploit the ways in which influence (structural, local, and global) can uncover improved routes to high quality solutions

For this purpose, integrate the two elements:

(1) the identification of changes that satisfy properties that are essential in order to achieve improvement

(2) the use of anticipatory ("means-ends") analysis to bring about such essential changes

Logical Restructuring

“What conditions assure the existence of a trajectory that will lead to an improved solution?”

“What intermediate moves can create such conditions?”

The intermediate moves may be generated by

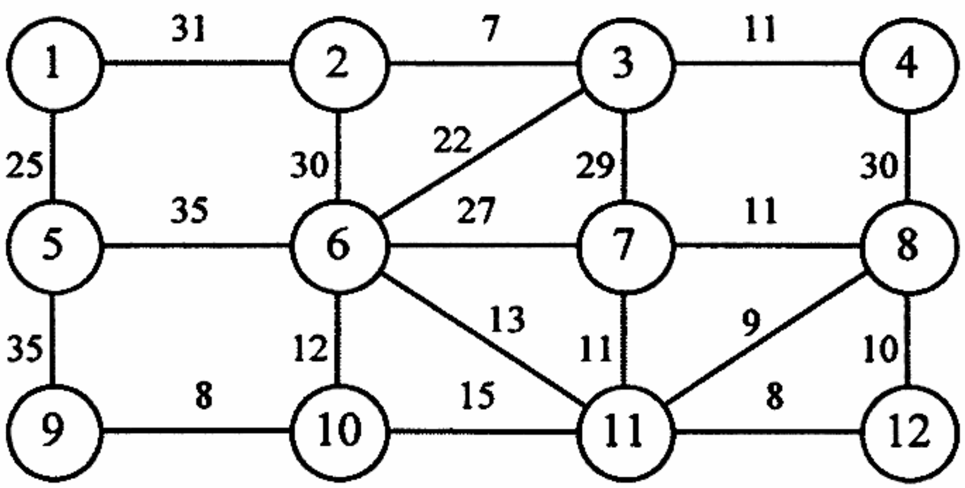
modifying the evaluation used to select transitions between solutions

modifying the neighborhood structure that determines these transitions

Logical Restructuring

Example

Fig. 3.5 Illustrative Min *k*-Tree Problem.



Logical Restructuring

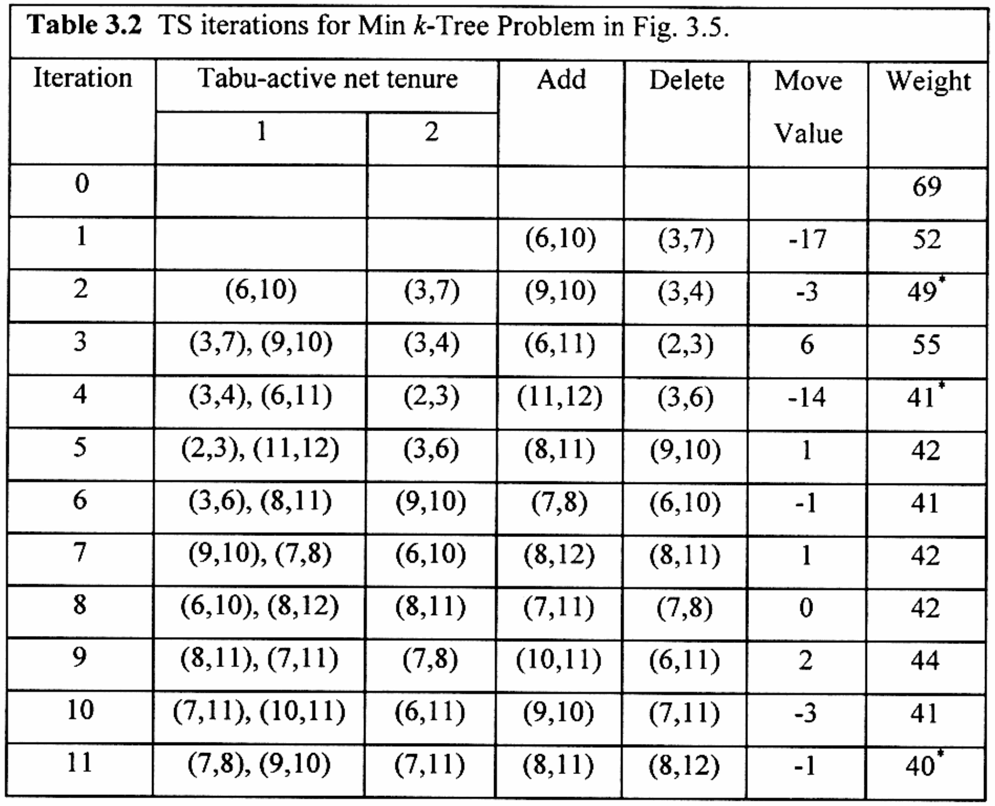


Table 3.2 TS iterations for Min *k*-Tree Problem in Fig. 3.5.

Iteration	Tabu-active net tenure		Add	Delete	Move Value	Weight
	1	2				
0						69
1			(6,10)	(3,7)	-17	52
2	(6,10)	(3,7)	(9,10)	(3,4)	-3	49*
3	(3,7), (9,10)	(3,4)	(6,11)	(2,3)	6	55
4	(3,4), (6,11)	(2,3)	(11,12)	(3,6)	-14	41*
5	(2,3), (11,12)	(3,6)	(8,11)	(9,10)	1	42
6	(3,6), (8,11)	(9,10)	(7,8)	(6,10)	-1	41
7	(9,10), (7,8)	(6,10)	(8,12)	(8,11)	1	42
8	(6,10), (8,12)	(8,11)	(7,11)	(7,8)	0	42
9	(8,11), (7,11)	(7,8)	(10,11)	(6,11)	2	44
10	(7,11), (10,11)	(6,11)	(9,10)	(7,11)	-3	41
11	(7,8), (9,10)	(7,11)	(8,11)	(8,12)	-1	40*

Logical Restructuring

Fig. 3.6 Graphical representation of TS iterations.

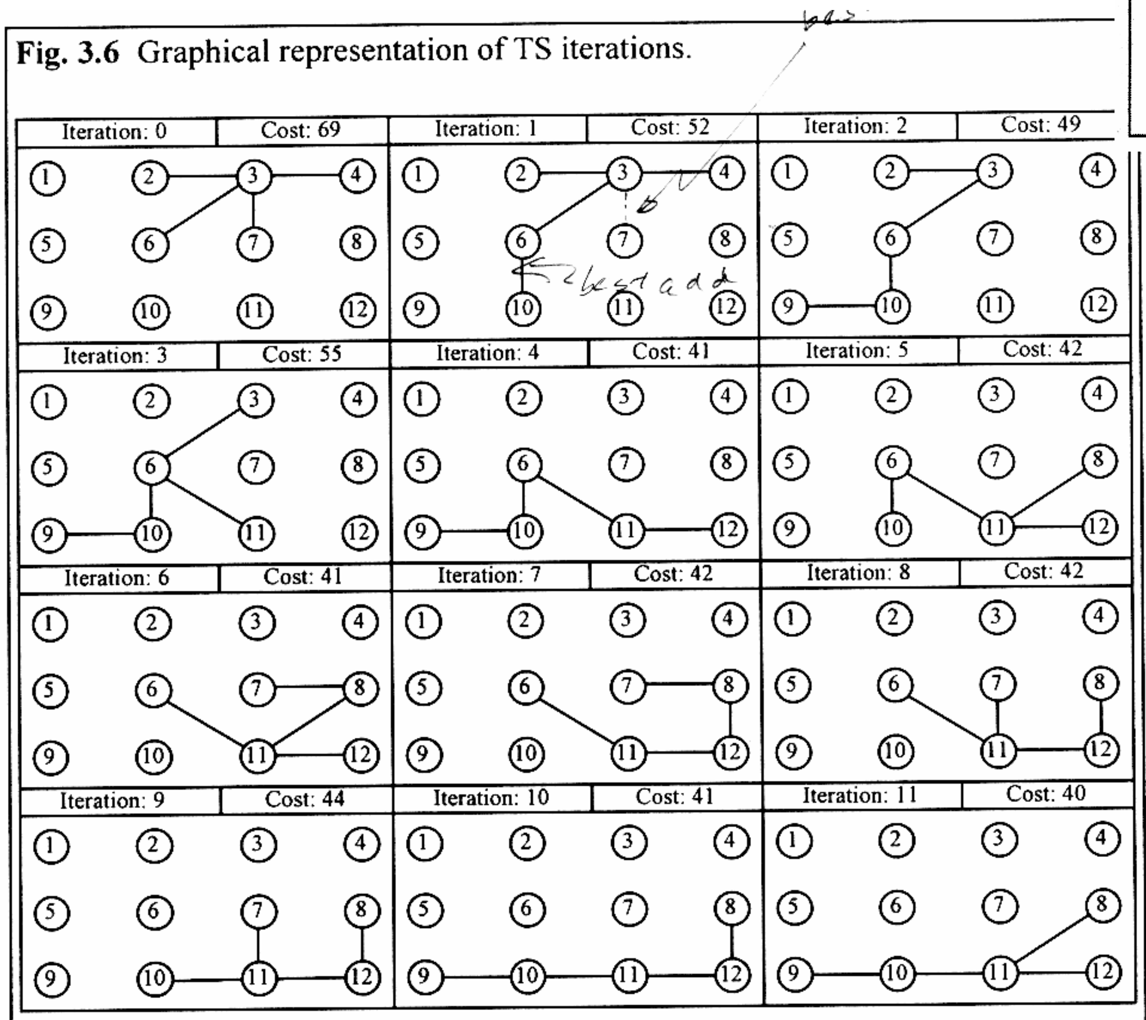
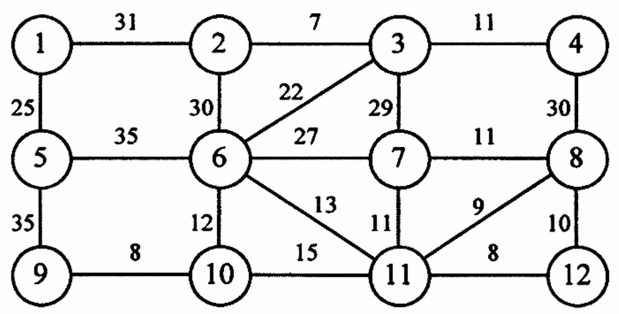


Fig. 3.5 Illustrative Min *k*-Tree Problem.



Logical Restructuring

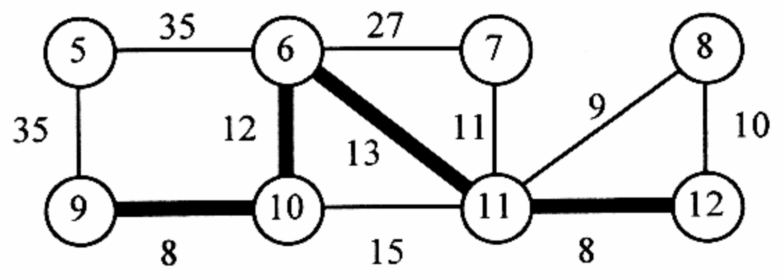
The delayed process of finding a route to an optimal solution can be accelerated by means of logical restructuring

Restructuring by Changing Evaluations and Neighborhoods

Edge (8,11) is a candidate to be added

Consider an anticipatory goal of making moves that cause more heavily weighted edges, i.e., edge (6,11) to become terminal edges, and hence eligible to be dropped (A dynamic swap can select an edge to be dropped only if it is a terminal edge)

Fig. 3.7 Solution and candidate edges to add to iteration 4 tree.

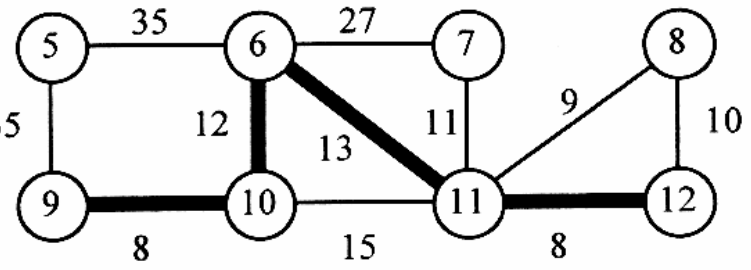


Restructuring by Changing Evaluations and Neighborhoods

The static swap that adds edge (10,11) and drops edge (6,10) with a move value of 3, produces a terminal edge (6,11)

Joining the static swap with the subsequent dynamic swap is a net gain (+3-4=-1)

Fig. 3.7 Solution and candidate edges to add to iteration 4 tree.



Restructuring by Changing Evaluations and Neighborhoods

The restructuring is accomplished as follows:

1. Identify the best edge to add for a dynamic swap, subject to requiring that this edge meet different node of the tree
2. Find a best combined move sequence (static swap + dynamic swap) by examining candidate static swaps;

A modified anticipatory move value is created for each swap that creates a terminal edge, by subtracting the weight of terminal edge from the standard move value

Add the anticipatory move value to the weight of added edge

Threshold Based Restructuring and Induced Decomposition

A property that in fact must be shared by all better solutions can be expressed as a threshold involving the average weight of the tree edges

This average weight must be less than the threshold value of 41/4

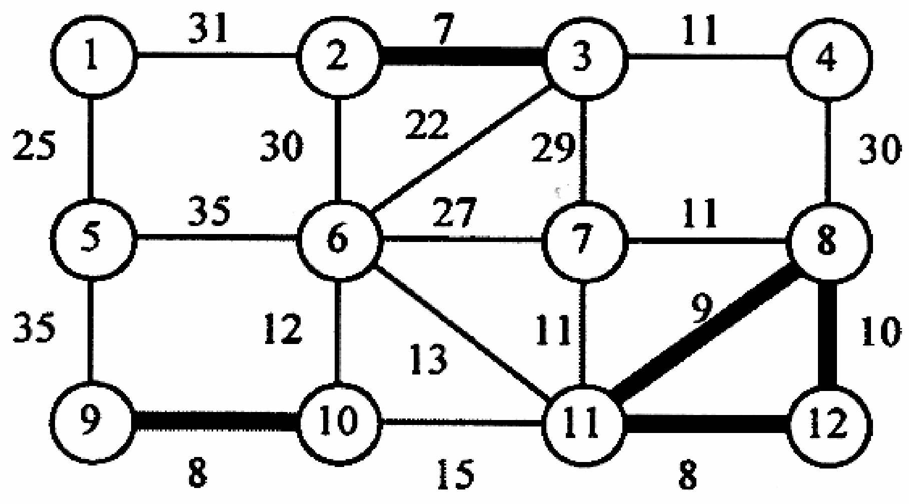
We look for ways to link the preferred edges to produce an improved solution

See Figure 3.8

A natural approach is to link such components by shortest paths, and then shave off terminal edges if the trees are too large, before returning to the swapping process

Threshold Based Restructuring and Induced Decomposition

Fig. 3.8 Threshold generated components.


1
2
3
4
5
6
7
8
9
10
11
12
31
7
11
25
30
22
29
30
35
27
11
35
12
13
11
9
10
8
15
8