Algorithm 1: Improve method for Lin-Kernighan

```
   Output1: True if improved, False if not
 1 foreach node along the path do
 2 |   foreach prenode, before and after the current node do
 3 |   |   gain = distance between current node and prenode
 4 |   |   removed_edges = (current node, prenode)
 5 |   |   close_nodes, reduced_gains = closest(node, gain, removed_edges)
 6 |   |   attempt = 5
 7 |   |   foreach close_node in close_nodes do
 8 |   |   |   if close_node is not a prenode then
 9 |   |   |   |   added_edges = (prenode, close_node)
10 |   |   |   |   if sucessfully remove_edge(node, close_node, reduced_gain,
   |   |   |   |    removed_edges, added_edges) then
11 |   |   |   |   |   return True
12 |   |   |   |   end
13 |   |   |   |   attempt -= 1
14 |   |   |   |   if attempt == 0 then
15 |   |   |   |   |   break
16 |   |   |   |   end
17 |   |   |   end
18 |   |   end
19 |   end
20 end
```

Algorithm 2: Closest method for Lin-Kernighan

```
   Input  : target node, gain, removed edges, added edges
   Output: Sorted Dictionary of nodes with potential and reduced gains
 1 foreach neighbor near target node do
 2 |   reduced_gain = gain - distance between target node and neighbor
 3 |   if reduced_gain > 0, (target_node, neighbor) is not in removed_edges
   |     and edges of the path then
 4 |   |   foreach near_node do
 5 |   |   |   if (near_node, neighbor) is not in removed_edges or
   |   |   |     added_edges then
 6 |   |   |   |   potential_gain = (distance between near_node and
   |   |   |   |     neighbor) - (distance between neighbor and target node)
 7 |   |   |   |   if neighbor in neighbors and potential_gain > gain to
   |   |   |   |     closest neighbor then
 8 |   |   |   |   |   save potential_gain in the neighbor dictionary
 9 |   |   |   |   else
10 |   |   |   |   |   save potential_gain and reduced_gain in the neighbor
   |   |   |   |   |     dictionary
11 |   |   |   |   end
12 |   |   |   end
13 |   |   end
14 |   end
15 end
16 return sorted neighbor dictionary
```

Algorithm 3: Remove edge method for Lin-Kernighan

```
   Input  : node, close node, gain, removed edges, added edges
   Output: True if successfully removed, False if not
 1 Check how many edges have been removed from the path, and only
    allow up to 5 edges to be removed.
 2 foreach near_node do
 3 |   current_gain = gain + distance between close_node and near_node if
   |    edge is not in added_edges and removed_edges then
 4 |   |   added = added_edges + (node, near_node)
 5 |   |   removed = removed_edges
 6 |   |   new_gain = current_gain - distance between node and near_node
 7 |   |   valid = create new tour with added and removed if valid or
   |   |    added length is less than 3 then
 8 |   |   |   if new tour is a known solution then
 9 |   |   |   |   return False
10 |   |   |   end
11 |   |   |   if valid then
12 |   |   |   |   save new tour as current path for TSP
13 |   |   |   |   return True
14 |   |   |   else
15 |   |   |   |   return add_edge(node, near_node, current_gain, removed,
   |   |   |   |    added_edges)
16 |   |   |   end
17 |   |   end
18 |   end
19 end
20 return False
```

Algorithm 4: Add edge method for Lin-Kernighan

```
   Input  : node, near node, gain, removed edges, added edges
   Output: True if successfully added, False if not
 1 close = closest(near_node, gain, removed_edges, added_edges)
 2 foreach close_node with reduced_gain do
 3 |   added = added_edges + (near_node, close_node)
 4 |   if remove_edge(node, close_node, reduced_gain, removed_edges,
   |    added) then
 5 |   |   return True
 6 |   end
 7 end
 8 return False
```