WIKIPEDIA

3-opt

In optimization, **3-opt** is a simple local search algorithm for solving the travelling salesperson problem and related network optimization problems. Compared to the simpler 2-opt algorithm, it is slower but can generate higher-quality solutions.

3-opt analysis involves deleting 3 connections (or edges) in a network (or tour), to create 3 sub-tours. Then the 7 different ways of reconnecting the network are analysed to find the optimum one. This process is then repeated for a different set of 3 connections, until all possible combinations have been tried in a network. A single execution of 3-opt has a time complexity of $O(n^3)$.[1] Iterated 3-opt has a higher time complexity.

This is the mechanism by which the 3-opt swap manipulates a given route:

```python
def reverse_segment_if_better(tour, i, j, k):
    """If reversing tour[i:j] would make the tour shorter, then do it."""
    # Given tour [...A-B...C-D...E-F...]
    A, B, C, D, E, F = tour[i-1], tour[i], tour[j-1], tour[j], tour[k-1], tour[k % len(tour)]
    d0 = distance(A, B) + distance(C, D) + distance(E, F)
    d1 = distance(A, C) + distance(B, D) + distance(E, F)
    d2 = distance(A, B) + distance(C, E) + distance(D, F)
    d3 = distance(A, D) + distance(E, B) + distance(C, F)
    d4 = distance(F, B) + distance(C, D) + distance(E, A)

    if d0 > d1:
        tour[i:j] = reversed(tour[i:j])
        return -d0 + d1
    elif d0 > d2:
        tour[j:k] = reversed(tour[j:k])
        return -d0 + d2
    elif d0 > d4:
        tour[i:k] = reversed(tour[i:k])
        return -d0 + d4
    elif d0 > d3:
        tmp = tour[j:k] + tour[i:j]
        tour[i:k] = tmp
        return -d0 + d3
    return 0
```

The principle is pretty simple. You compute the original distance d_0 and you compute the cost of each modification. If you find a better cost, apply the modification and return δ (relative cost). This is the complete 3-opt swap making use of the above mechanism:

```python
def three_opt(tour):
    """Iterative improvement based on 3 exchange."""
    while True:
        delta = 0
        for (a, b, c) in all_segments(len(tour)):
            delta += reverse_segment_if_better(tour, a, b, c)
        if delta >= 0:
            break
    return tour

def all_segments(n: int):
    """Generate all segments combinations"""
    return ((i, j, k)
        for i in range(n)
        for j in range(i + 2, n)
        for k in range(j + 2, n + (i > 0)))
```

For the given tour, you generate all segments combinations and for each combinations, you try to improve the tour by reversing segments. While you find a better result, you restart the process, otherwise finish.

See also

- 2-opt
- Local search (optimization)
- Lin–Kernighan heuristic

References

1. Blazinskas, Andrius; Misevicius, Alfonsas (2011). "Combining 2-OPT, 3-OPT and 4-OPT with K-SWAP-KICK perturbations for the traveling salesman problem". S2CID 15324387 (https://api.semanticscholar.org/CorpusID:15324387).

- F. BOCK (1965). *An algorithm for solving traveling-salesman and related network optimization problems*. unpublished manuscript associated with talk presented at the 14th ORSA National Meeting.
- S. LIN (1965). *Computer solutions of the traveling salesman problem*. Bell Syst. Tech. J. 44, 2245-2269. Available as PDF (http://bstj.bell-labs.com/BSTJ/images/Vol44/bstj44-10-2245.pdf)
- S. LIN AND B. W. KERNIGHAN (1973). *An Effective Heuristic Algorithm for the Traveling-Salesman Problem*. Operations Res. 21, 498-516. Available as PDF (http://www.dti.unimi.it/~righini/Didattica/Algoritmi%20Euristici/MaterialeAE/Lin%20Kernighan%20TSP.pdf)
- Local Search Heuristics. (n.d.) Retrieved June 16, 2008, from http://www.tmsk.uitm.edu.my/~naimah/csc751/slides/LS.pdf

External links