Local Search Heuristics

LocalSearch(ProblemInstance *x*)
y := feasible solution to *x*;
while ∃ *z* ϵ *N*(*y*): *v*(*z*)<*v*(*y*) **do**
    y := *z*;
od;
return *y*;

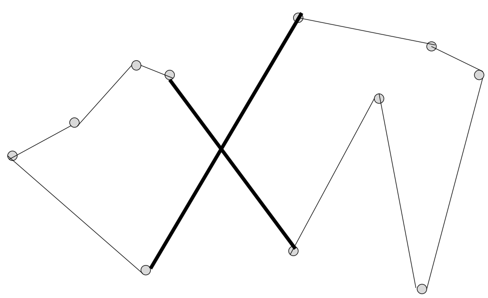
1

To do list

- How do we find the first feasible solution?
- Neighborhood design?
- Which neighbor to choose?
- Partial correctness? **Never Mind!**
- Termination? **Stop when tired! (but optimize the time of each iteration).**
- Complexity?

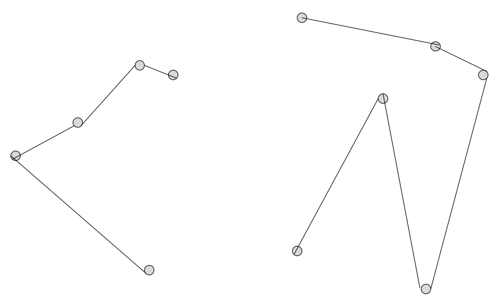
2

2-opt neighborhood



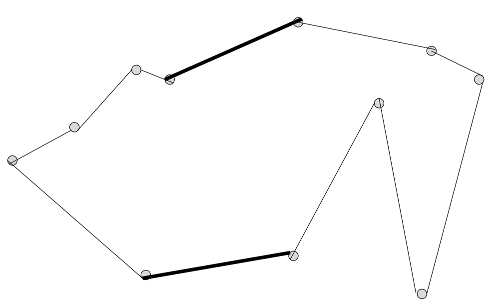
3

2-opt neighborhood



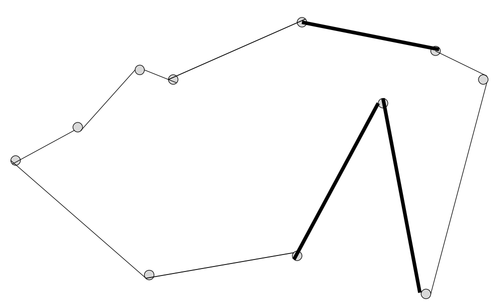
4

2-optimal solution



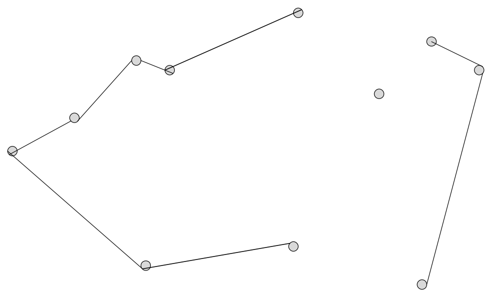
5

3-opt neighborhood



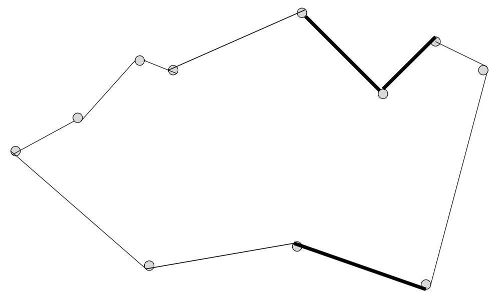
6

3-opt neighborhood



7

3-opt neighborhood



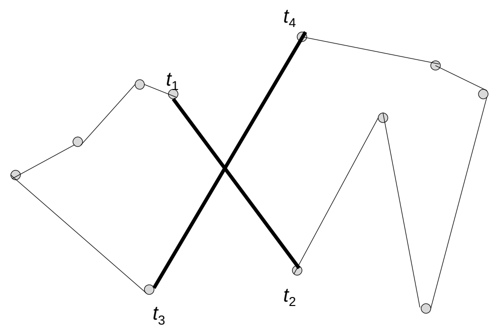
8

Table 8.3 Tour quality for 2-Opt and 3-Opt, plus selected tour generation heuristics: average percent excess over the Held–Karp lower bound

$N =$	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5	$10^{5.5}$	10^6
				Random Euclidean instances					
GR	19.5	18.8	17.0	16.8	16.6	14.7	14.9	14.5	14.2
CW	9.2	10.7	11.3	11.8	11.9	12.0	12.1	12.1	12.2
CHR	9.5	9.9	9.7	9.8	9.9	9.8	9.9	–	–
2-opt	4.5	4.8	4.9	4.9	5.0	4.8	4.9	4.8	4.9
3-opt	2.5	2.5	3.1	3.0	3.0	2.9	3.0	2.9	3.0
				Random distance matrices					
GR	100	160	170	200	250	280	–	–	–
2-opt	34	51	70	87	125	150	–	–	–
3-opt	10	20	33	46	63	80	–	–	–

9

2-opt neighborhood



10

Boosting local search

- Taboo search
- Simulated annealing
- Evolutionary algorithms

Theme: Avoiding local optima.

11

Taboo search

- When the local search reaches a local minimum, **keep searching.**

12

Local Search

LocalSearch(ProblemInstance *x*)
y := feasible solution to *x*;
while $\exists z \in N(y)$: $v(z)<v(y)$ **do**
    y := *z*;
od;
return *y*;

13

Taboo search, attempt 1

LocalSearch(ProblemInstance *x*)
y := feasible solution to *x*;
while not tired **do**
    y := best neighbor of y;
od;
return best solution seen;

14

Serious Problem

- The modified local search will typically enter a cycle of length 2.
- As soon as we leave a local optimum, the next move will typically bring us back there.

15

Attempt at avoiding cycling

- Keep a list of already seen solutions.
- Make it illegal (“taboo”) to enter any of them.
- Not very practical – list becomes long. Also, search tend to circle around local optima.

16

Taboo search

- After a certain “move” has been made, it is declared ***taboo*** and may not be used for a while.
- “Move” should be defined so that it becomes taboo to go right back to the local optimum just seen.

17

MAXSAT

- Given a formula *f* in CNF, find an assignment *a* to the variables of *f*, satisfying as many clauses as possible.

18

Solving MAXSAT using GSAT

- Plain local search method: GSAT.
- GSAT Neighborhood structure: Flip the value of one of the variables.
- Do steepest descent.

19

Taboo search for MAXSAT

- As in GSAT, flip the value of one of the variables and choose the steepest descent.
- When a certain variable has been flipped, it cannot be flipped for, say, $n/4$ iterations.We say the variable is **taboo**. When in a local optimum, make the "least bad" move.

20

TruthAssignment TabooGSAT(CNFformula *f*)
  t := 0; *T* :=Ø; *a*,*best* := some truth assignment;
  repeat
    Remove all variables from *T* with time stamp < *t*-*n*/4;
.
    For each variable *x* *not* in *T*, compute the number of clauses satisfied by the assignment obtained from *a* by flipping the value of *x*. Let *x* be the best choice and let *a'* be the corresponding assignment.
    a = *a*'; Put *x* in *T* with time stamp *t*;
    if *a* is better than *best* **then** *best* = *a*;
    t := *t* +1
  until tired
return *best;*

21

TSP

- No variant of "pure" taboo search works very well for TSP.
- Johnson og McGeoch: Running time 12000 as slow as 3opt on instances of size 1000 with no significant improvements.
- **General remark:** Heuristics should be compared on a time-equalized basis.

22

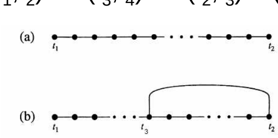
Lin-Kernighan

- Very successful classical heuristic for TSP.
- Similar to Taboo search: Boost 3-opt by sometimes considering "uphill" (2-opt) moves.
- When and how these moves are considered is more "planned" and "structured" than in taboo search, but also involves a "taboo criterion".
- ***Often misrepresented in the literature!***

23

Looking for 3opt moves

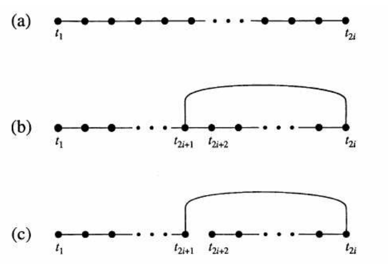
- WLOG look for t_1, t_2, t_3, t_4,t_5,t_6 so that $d(t_1,t_2) > d(t_2, t_3)$ and $d(t_1,t_2) + d(t_3,t_4) > d(t_2,t_3)+d(t_4, t_5)$.



- The weight of the one-tree smaller than length of original tour.

24

Lin-Kernighan move



Lin-Kernighan moves

- A 2opt move can be viewed as LK-move.
- A 3opt move can be viewed as two LK-moves.
- The inequalities that can be assumed WLOG for legal 3-opt (2-opt) moves state than the one-trees involved are shorter than the length of the original tour.

Lin-Kernighan search

- 3opt search with "intensification".
- Whenever a 3opt move is being made, we view it as two LK-moves and see if we **in addition** can perform a number of LK-moves (an LK-search) that gives an even better improvement.
- During the LK-search, we never delete an edge we have added by an LK-move, so we consider at most *n*-2 additonal LK-moves ("taboo criterion"). We keep track of the $\leq n$ solutions and take the best one.
- The next move we consider is the best LK-move we can make. **It could be an uphill move.**
- We only allow one-trees lighter than the current tour. Thus, we can use neighbor lists to speed up finding the next move.

Table 8.7 Tour quality for Lin–Kernighan in comparison to 3-Opt: average percent excess over the Held–Karp lower bound

$N =$	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5	$10^{5.5}$	10^6
Random Euclidean instances									
3-Opt	2.5	2.5	3.1	3.0	3.0	2.9	3.0	2.9	3.0
LK	1.5	1.7	2.0	1.9	2.0	1.9	2.0	1.9	2.0
Random distance matrices									
3-Opt	10.0	20.0	33.0	46.0	63.0	80.0	–	–	–
LK	1.4	2.5	3.5	4.6	5.8	6.9	–	–	–

Table 8.8 Running times for Lin–Kernighan in comparison to 3-Opt: seconds on a 150 MHz SGI Challenge

$N =$	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5	$10^{5.5}$	10^6
Random Euclidean instances									
3-Opt	0.04	0.11	0.41	1.40	4.7	17.5	69	280	1 080
LK	0.06	0.20	0.77	2.46	9.8	39.4	151	646	2 650
Random distance matrices									
3-Opt	0.02	0.16	1.14	9.8	110	1 410	–	–	–
LK	0.05	0.35	1.90	13.6	139	1 620	–	–	–

What if we have more CPU time?

- We could repeat the search, with ***different starting point.***
- Seems better not to throw away result of previous search.

Iterated Lin-Kernighan

- After having completed a Lin-Kernighan run (i.e., 3opt, boosted with LK-searches), make a **random** 4-opt move and do a new Lin-Kernighan run.
- Repeat for as long as you have time. Keep track of the best solution seen.
- The 4-opt moves are restricted to **double bridge** moves (turning A_1 A_2 A_3 A_4 into A_2 A_1 A_4 A_3.)

Table 8.15 Comparison of iterated Lin–Kernighan with independent LK runs

	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5
	Average percent excess over the Held–Karp lower bound						
Independent iterations							
1	1.52	1.68	2.01	1.89	1.96	1.91	1.95
$N/10$	0.99	1.10	1.41	1.62	1.71	–	–
$N/10^{0.5}$	0.92	1.00	1.35	1.59	1.68	–	–
N	0.91	0.93	1.29	1.57	1.65	–	–
ILK iterations							
$N/10$	1.06	1.08	1.25	1.21	1.26	1.25	1.31
$N/10^{0.5}$	0.96	0.90	0.99	1.01	1.04	1.04	1.08
N	0.92	0.79	0.91	0.88	0.89	0.91	–
	Running time in seconds on a 150 MHz SGI Challenge						
Independent iterations							
1	0.06	0.2	0.8	3	10	40	150
$N/10$	0.42	4.7	48.1	554	7 250	–	–
$N/10^{0.5}$	1.31	14.5	151.3	1 750	22 900	–	–
N	4.07	45.6	478.1	5 540	72 400	–	–
ILK iterations							
$N/10$	0.14	0.9	5.1	27	189	1 330	10 200
$N/10^{0.5}$	0.34	2.4	13.6	76	524	3 810	30 700
N	0.96	6.5	39.7	219	1 570	11 500	–

31