Unless P=NP, there is no polynomial time algorithm for

SAT, MAXSAT, MIN NODE COVER, MAX INDEPENDENT SET, MAX CLIQUE, MIN SET COVER, TSP, ….

But we have to solve (instances of) these problems anyway – what do we do?

Two approaches

- Exponential-time algorithms finding the exact solution:
  - Branch-and-bound,
  - Branch-and-cut,
  - Branch-and-reduce.
- Approximation algorithms and heuristics finding "good" solutions in polynomial time.

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*;

Examples of algorithms using local search

- Ford-Fulkerson algorithm for Max Flow
- Klein's algorithm for Min Cost Flow
- Simplex Algorithm

Local search **heuristics** - 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?

TSP

- Johnson and McGeoch. The traveling salesman problem: A case study (from Local Search in Combinatorial Optimization).
- Covers plain local search as well as concrete instantiations of popular **metaheuristics** such as tabu search, simulated annealing and evolutionary algorithms.
- ***An example of good experimental methodology!***

TSP

- Branch-and-cut method gives a practical way of solving TSP instances of up to ~ 1000 cities to optimality.
- Instances considered by Johnson and McGeoch: ***Random*** Euclidean instances and ***random*** distance matrix instances of several thousands cities.

7

Local search design tasks

- Finding an initial solution
- Neighborhood structure

8

The initial tour

- Christofides
- Greedy heuristic
- Nearest neighbor heuristic
- Clarke-Wright

9

Table 8.1 Tour quality for 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					
CHR	9.5	9.9	9.7	9.8	9.9	9.8	9.9	–	–
CW	9.2	10.7	11.3	11.8	11.9	12.0	12.1	12.1	12.2
GR	19.5	18.8	17.0	16.8	16.6	14.7	14.9	14.5	14.2
NN	25.6	26.2	26.0	25.5	24.3	24.0	23.6	23.4	23.3
				Random distance matrices					
GR	100	160	170	200	250	280	–	–	–
NN	130	180	240	300	360	410	–	–	–
CW	270	520	980	1800	3200	5620	–	–	–

10

Held-Karp lower bound

- Value of certain LP-relaxation of the TSP-problem.
- Guaranteed to be at least 2/3 of the true value for metric instances.
- Empirically usually within 0.01% (!)

11

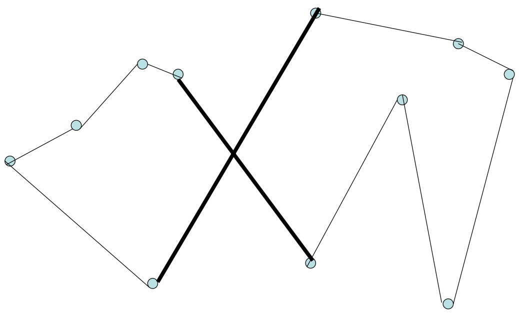
Neighborhood design

Natural neighborhood structures:

2-opt, 3-opt, 4-opt,…

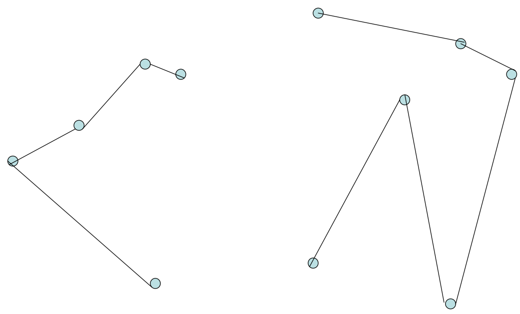
12

2-opt neighborhood



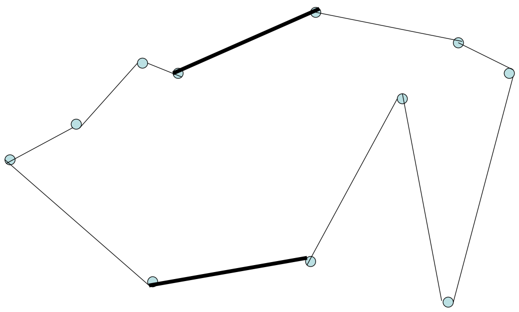
13

2-opt neighborhood



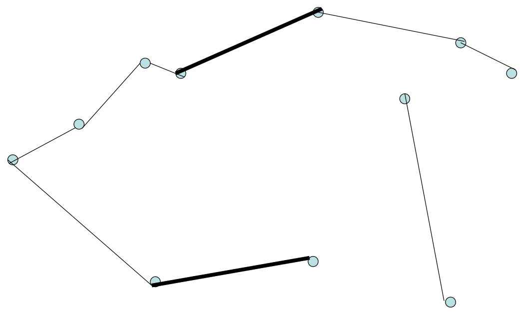
14

2-opt neighborhood



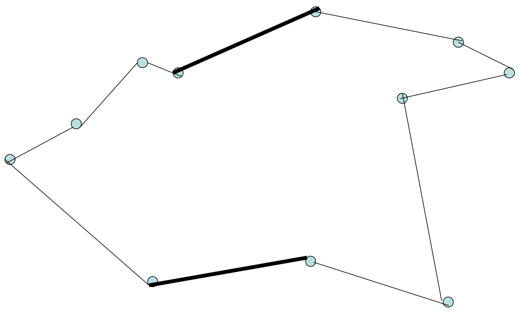
15

2-opt neighborhood



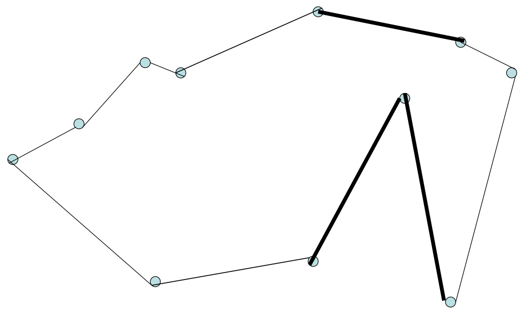
16

2-optimal solution



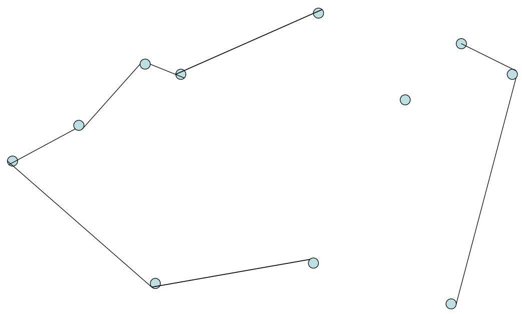
17

3-opt neighborhood



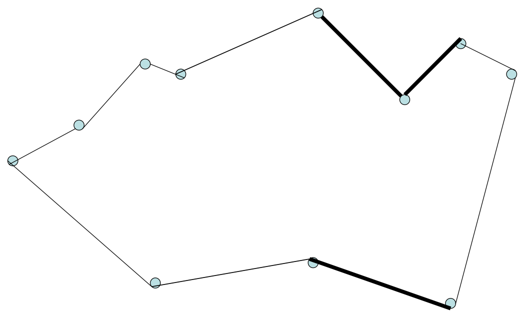
18

3-opt neighborhood



19

3-opt neighborhood



20

Neighborhood Properties

- Size of *k*-opt neighborhood: $O(n^k)$
- Non-trivial algorithmics are applied for implementation..

21

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	–	–	–

22

Table 8.4 Tour quality for 2-Opt and 3-Opt using different starting heuristics: average percent, excess over the Held–Karp lower bound, $N = 1000$

	Random Euclidean instances			Random distance matrices		
	Start	2-Opt	3-Opt	Start	2-Opt	3-Opt
Random	2 150.0	7.9	3.8	24 500	290	55
NN	25.9	6.6	3.6	240	96	39
Greedy	17.6	4.9	3.1	170	70	33
CW	11.4	8.5	5.0	980	380	56

23

Table 8.6 Running times for 2-Opt and 3-Opt, plus selected tour construction heuristics: seconds on a 150 MHz SGI Challenge

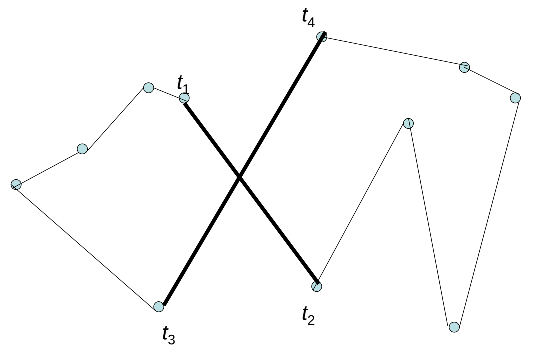
$N =$	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5	$10^{5.5}$	10^6
				Random Euclidean instances					
CHR	0.03	0.12	0.53	3.57	41.9	801.9	23 009	–	–
CW	0.00	0.03	0.11	0.35	1.4	6.5	31	173	670
GR	0.00	0.02	0.08	0.29	1.1	5.5	23	90	380
Preprocessing	0.02	0.07	0.27	0.94	2.8	10.3	41	168	673
Starting tour	0.01	0.01	0.04	0.15	0.6	2.7	12	50	181
2-Opt	0.00	0.01	0.03	0.09	0.4	1.5	6	23	87
3-Opt	0.01	0.03	0.09	0.32	1.2	4.5	16	61	230
Total 2-Opt	0.03	0.09	0.34	1.17	3.8	14.5	59	240	940
Total 3-Opt	0.04	0.11	0.41	1.40	4.7	17.5	69	280	1 080
				Random distance matrices					
GR	0.02	0.12	0.98	9.3	107	1 400	–	–	–
Preprocessing	0.01	0.11	0.94	9.0	104	1 370	–	–	–
Starting tour	0.00	0.01	0.05	0.3	3	27	–	–	–
2-Opt	0.00	0.01	0.03	0.1	0	1	–	–	–
3-Opt	0.01	0.04	0.16	0.6	3	15	–	–	–
Total 2-Opt	0.02	0.13	1.02	9.4	108	1 400	–	–	–
Total 3-Opt	0.02	0.16	1.14	9.8	110	1410	–	–	–

24

- One 3OPT move takes time $O(n^3)$. How is it possible to do local optimization on instances of size 10^6 ?????

25

2-opt neighborhood



26

A 2-opt move

- If $d(t_1, t_2) \leq d(t_2, t_3)$ and $d(t_3,t_4) \leq d(t_4,t_1)$, the move is not improving.
- Thus we can restrict searches for tuples where either $d(t_1, t_2) > d(t_2, t_3)$ or $d(t_3, t_4) > d(t_4, t_1)$.
- WLOG, $d(t_1,t_2) > d(t_2, t_3)$.

27

Neighbor lists

- For each city, keep a static list of cities in order of increasing distance.
- When looking for a 2-opt move, for each candidate for t_1 with t_2 being the next city, look in the neighbor list of t_2 for t_3 candidate, searching “inwards” from t_1.
- For random Euclidean instance, expected time to for finding 2-opt move is linear.

28

Problem

- Neighbor lists are very big.
- It is very rare that one looks at an item at position > 20.
- Solution: Prune lists to 20 elements.

29

- Still not fast enough……

30

Don't-look bits.

- If a candidate for t_1 was unsuccessful in previous iteration, and its successor and predecessor has not changed, ignore the candidate in current iteration.

31

Variant for 3opt

- 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 $d(t_1,t_2)+d(t_3,t_4) > d(t_2,t_3)+d(t_4, t_5)$.

32

Boosting local search

- Theme: How to escape local optima
  - Taboo search, Lin-Kernighan
  - Simulated annealing
  - Evolutionary algorithms

33

Taboo search

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

34

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*;

35

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;

36

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.

37

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.

38

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.

39

MAXSAT

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

40

Solving MAXSAT using GSAT

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

41

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.

42

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;*

TSP

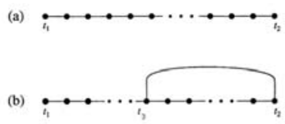
- No variant of "pure" taboo search works very well for TSP.
- Johnson og McGeoch: Running time 12000 as slow as 3opt on instances of size 1000 with no significant improvements.
- **General remark:** Heuristics should be compared on a time-equalized basis.

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!***

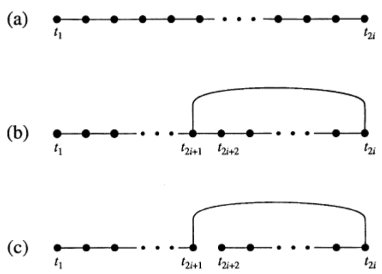
Looking for 3opt moves

- 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 (b) smaller than length of original tour.

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-tree"s 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 additional LK-moves (“taboo criterion”). We keep track of the $\leq n$ solutions and take the best one.
- During the LK-search, 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.

49

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	–	–	–

50

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.

51

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.)

52

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	–

53

Boosting local search

- **Simulated annealing**
  (inspired by physics)
- **Evolutionary algorithms**
  (inspired by biology)

54

Metropolis algorithm and simulated annealing

- Inspired by physical systems (described by statistical physics).
- Escape local minima by allowing move to worse solution with a certain probability.
- The probability is regulated by a parameter, the *temperature* of the system.
- High temperature means high probability of allowing move to worse solution.

55

Metropolis Minimization

FeasibleSolution Metropolis(ProblemInstance *x,* Real *T*)
  y := feasible solution to *x*;
  repeat
    Pick a random member *z* of *N*(*y*);
    with probability min($e^{(v(y)-v(z))/T}$, 1) let *y*:=*z;*
  until tired;
  return the best *y* found;

56

Why $\min(e^{(v(y)-v(z))/T},1)$?

- Improving moves are always accepted, bad moves are accepted with probability ***de***creasing with badness but ***in***creasing with temperature.
- Theory of Markov chains: As number of moves goes to infinity, the probability that *y* is some value *a* becomes proportional to exp(-*v*(*a*)/*T*)
- This convergence is in general slow (an exponential number of moves must be made). **Thus, in practice, one should feel free to use other expressions**.

57

What should *T* be?

Intuition:

T large: Convergence towards limit distribution fast, but limit distribution does not favor good solutions very much (if T is infinity, the search is random).

T close to 0 : Limit distribution favor good solution, but convergence slow.

T = 0: Plain local search.

One should pick "optimal" *T.*

58

Simulated annealing

- As Metropolis, but *T* is changed during the execution of the algorithm.
- *T* starts out high, but is gradually lowered.
- **Hope:** *T* stays at near-optimal value sufficiently long.
- Analogous to models of crystal formation.

59

Simulated Annealing

FeasibleSolution Metropolis(ProblemInstance *x)*
  y := feasible solution to *x*; T:=big;
  repeat
    T := 0.99 *T* ;
    Pick a random member *z* of *N*(*y*);
    with probability min($e^{(v(y)-v(z))/T}$, 1) let *y*:=*z*
  until tired;
  return the best *y* found;

60

Simulated annealing

- **THM:** If *T* is lowered sufficiently slowly (exponentially many moves must be made), **the final solution will with high probability be optimal**!
- In practice *T* must be lowered faster.

61

TSP

- Johnson and McGeoch: Simulated annealing with 2opt neighborhood is promising but neighborhood must be pruned to make it efficient.
- Still, not competitive with LK or ILK on a time-equalized basis (for any amount of time).

62

Table 8.10 Results for variants of simulated annealing, with temperature length set to α times the total length of the initial neighbor lists, compared to results for one or more runs of 2-Opt, 3-Opt, and Lin–Kernighan random Euclidean instances

Variant		Average percent excess			Running time in seconds		
		10^2	$10^{2.5}$	10^3	10^2	$10^{2.5}$	10^3
SA_1 (baseline annealing)	$\alpha = 1$	3.4	3.7	4.0	12.40	188.00	3170.00
SA_1 + pruning	$\alpha = 1$	2.7	3.2	3.8	3.20	18.00	81.00
SA_1 + pruning	$\alpha = 10$	1.7	1.9	2.2	32.00	155.00	758.00
SA_2 (pruning + low temp.)	$\alpha = 10$	1.6	1.8	2.0	14.30	50.30	229.00
SA_2	$\alpha = 40$	1.3	1.5	1.7	58.00	204.00	805.00
SA_2	$\alpha = 100$	1.1	1.3	1.6	141.00	655.00	1910.00
2-Opt		4.5	4.8	4.9	0.03	0.09	0.34
Best of 1000 2-Opts		1.9	2.8	3.6	6.60	16.20	52.00
Best of 10000 2-Opts		1.7	2.6	3.4	66.00	161.00	517.00
3-Opt		2.5	2.5	3.1	0.04	0.11	0.41
Best of 1000 3-Opts		1.0	1.3	2.1	11.30	33.00	104.00
Best of 10000 3-Opts		0.9	1.2	1.9	113.00	326.00	1040.00
Lin–Kernighan		1.5	1.7	2.0	0.06	0.20	0.77
Best of 100 LKs		0.9	1.0	1.4	4.10	14.50	48.00

63

Boosting local search using metaheuristics

- Theme: How to escape local optima
  – Taboo search, Lin-Kernighan
  – Simulated annealing
  – Evolutionary algorithms

64

Local Search – interpreted biologically

FeasibleSolution LocalSearch(ProblemInstance *x*)
    y := feasible solution to *x*;
    while Improve(*y*) != *y* and !tired **do**
        y := Improve(*y*);
    od;
    return *y*;

Improve(y) is an **offspring** of *y.* The **fitter** of the two will survive

Maybe y should be allowed to have other children?

Maybe the “genetic material” of y should be combined with the “genetic material” of others?

65

Evolutionary/Genetic algorithms

- Inspired by biological systems (evolution and adaptation)
- Maintain a *population* of solution
- *Mutate* solutions, obtaining new ones.
- *Recombine* solutions, obtaining new ones.
- *Kill* solutions randomly, with better (more *fit*) solutions having lower probability of dying.

66

Evolutionary Algorithm

FeasibleSolution EvolSearch(ProblemInstance *x*)
  P := initial population of size *m* of feasible solutions to *x*;
  while !tired **do**
    Expand(*P)*;
    Selection(*P)*
  od;
  return best solution obtained at some point;

67

Expansion of Population

Expand(Population *P*)
  for *i*:=1 to *m* **do**
    with probability *p* **do**
      ExpandByMutation(*P*)
    else (i.e., with probability 1-*p*)
      ExpandByCombination(*P*)
    htiw
  od

68

Expand Population by Mutation

ExpandByMutation(Population *P*)
  Pick random *x* in *P;*
  Pick random *y* in *N*(x);
  P := *P* U {*y*};

69

Expand Population by Combination

ExpandByCombination(Population *P)*
  Pick random *x* in *P;*
  Pick random *y* in *P;*
  z := Combine(*x*,*y*);
  P := *P* U {*z*};

70

Selection

Selection(Population *P*)
  while |*P*| > *m* **do**
    Randomly select a member *x* of *P but* select each particular *x* with probability monotonically increasing with *v(x);*
    P := *P* – {*x*};
  od

71

How to combine?

- Problem specific decision.
- There is a “Generic way”: Base it on the way biological recombination of genetic material is done.

72

Biological Recombination

- Each feasible solution (the *phenotype*) is represented by a string over a finite alphabet (the *genotype*).
- String x is combined with string y by splitting x in x_1x_2 and y in y_1y_2 with $|x_1|=|y_1|$ and $|x_2|=|y_2|$ and returning x_1y_2.

73

Evolutionary algorithms

- Many additional biological features can be incorporated.
- Dozens of decisions to make and knobs to turn.
- One option: Let the decisions be decided by evolution as well!

74

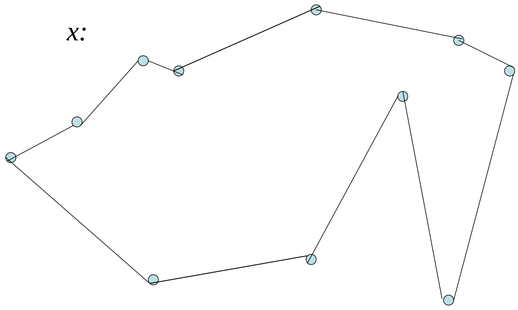
Conclusions of McGeoch and Johnson

Best known heuristics for TSP:

- Small CPU time: Lin-Kernighan.
- Medium CPU time: Iterated Lin-Kernighan (Lin-Kernighan + Random 4opt moves).
- Very Large CPU time: An evolutionary algorithm.

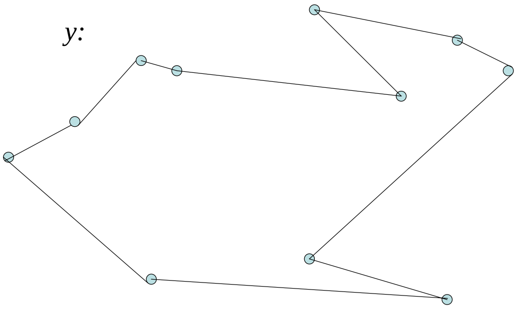
75

Combine operation in winning approach for large CPU time



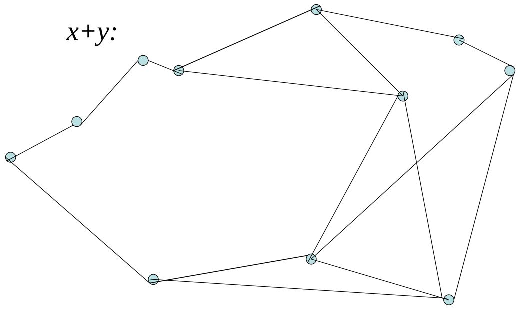
76

Combine operation in winning approach for large CPU time



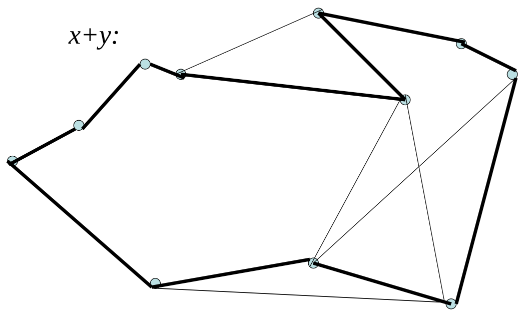
77

Take union of *x* and *y*



78

Solve to optimality, using only edges from *x* + *y*



79

Combine(*x*,*y*)

- Combine(*x*,*y*): Take the graph consisting of edges of *x* and *y.* Find the optimal TSP solution using only edges from that graph.
- Finding the optimal TSP tour in a graph which is the union of two Hamiltonian paths can be done efficiently in practice.
- More “obvious” versions of combine (like the generic combine) yield evolutionary algorithms which are not competitive with simpler methods.

80