Evolutionary Algorithms and Combinatorial Optimization

Presenter:

Peter Ross

P.Ross@napier.ac.uk

The brief outline

- *Combinatorial optimization:* some types of problems
- *Non-evolutionary methods:* a quick tour
- *Evolutionary methods:* a range of ideas
- *Evolutionary methods:* do they work well?
- *What next?* Some developing ideas and directions for research

What is combinatorial optimization?

In short: optimization involving **discrete-valued variables** – usually integer-valued.

Practical examples: crew scheduling; vehicle routing problems; facility (and other) layout problems; packing problems and many more.

Textbook topics: network flows; shortest-path problems; matching; graph coloring and satisfiability; graph partitioning and many more.

A very simple example

A five-month project needs, in each month:

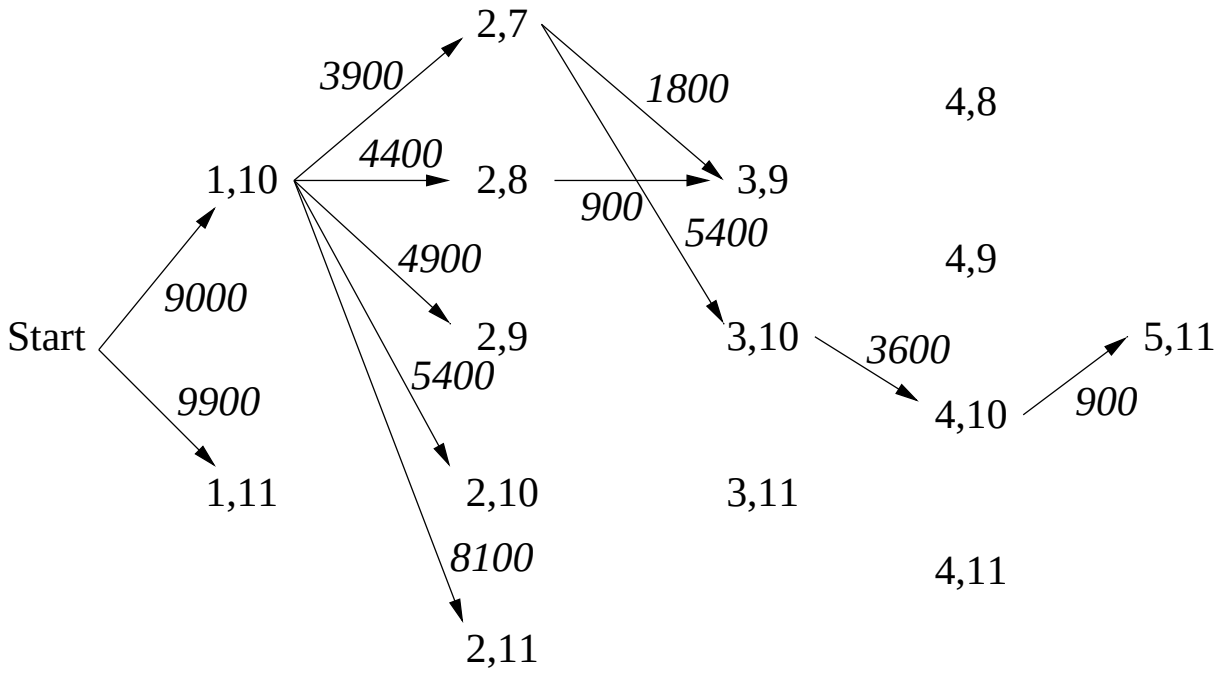
Month	1	2	3	4	5
People	10	7	9	8	11

Extra costs: \$900 to recruit/train an employee; \$1300 to get rid of an employee; \$1800 to keep each superfluous employee in a given month.

Minimise the total extra costs.

A very simple example, continued

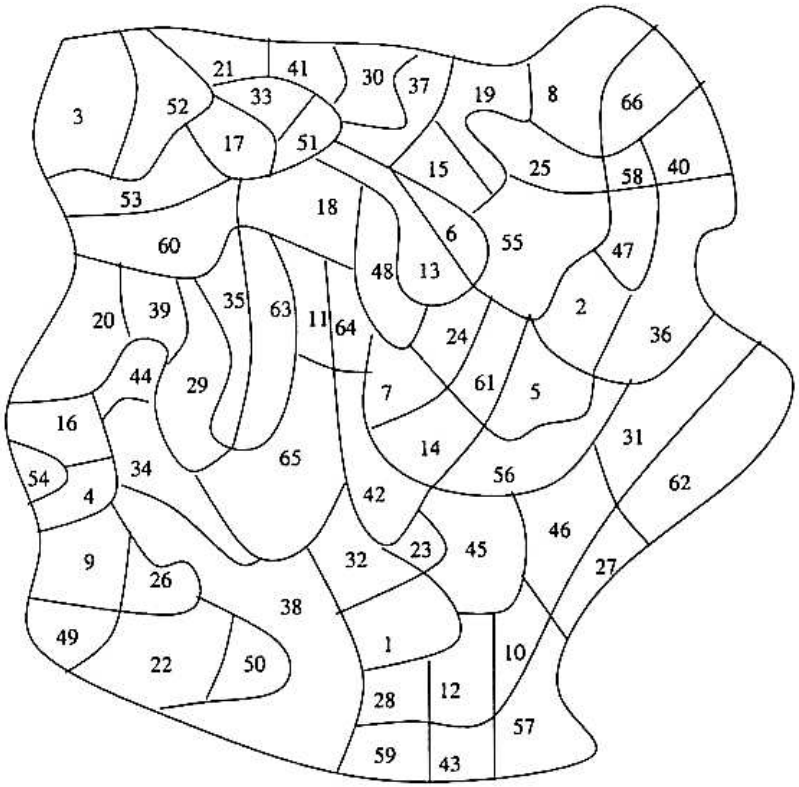
$M, N = M$ employees in month N Many edges omitted:



It becomes a shortest-path problem. Brute force works.

Another example

Combine these 66 districts into just 14, about 12,000 each:



1	2992	21	2008	41	2020	61	2009
2	2032	22	1991	42	2983	62	3004
3	3021	23	2987	43	3004	63	3026
4	1973	24	2031	44	1970	64	2984
5	2020	25	2969	45	3023	65	3011
6	2977	26	3028	46	2973	66	2993
7	2003	27	2991	47	2024		
8	3004	28	3001	48	5976		
9	2985	29	1993	49	1975		
10	3024	30	2010	50	1976		
11	3032	31	2995	51	1969		
12	3004	32	2979	52	2978		
13	3020	33	2008	53	2030		
14	1980	34	2010	54	2002		
15	2026	35	3027	55	2971		
16	2008	36	2991	56	1991		
17	1990	37	1974	57	2997		
18	1984	38	2019	58	2024		
19	1978	39	1979	59	2990		
20	2008	40	3028	60	1993		

(Dennis Shasha)

Combinatorial optimization methods

Many kinds, no universally agreed taxonomy:

Classical exhaustive, guaranteed: eg, linear programming, integer linear programming, branch-and-bound, constraint satisfaction, . . .

Heuristic instance-based given one/several candidates, look for better: eg, simulated annealing, genetic algorithms, GRASP, . . . plus metaheuristics such as tabu search

Heuristic model-based reasoning about distributions/landscapes: eg, PBIL, ant systems, simulated entropy methods, nested partitioning, . . .

A note about simulated entropy (SE) methods

Example: a shortest-path problem:

- start with a random Markov chain of probabilities of taking allowed transitions (source and target are absorbing states)
- use Boltzman sampling and K-L cross-entropy to move towards a Markov chain in which there is a single 1 and otherwise 0 in each row, that defines (with high prob) the shortest path

(See eg `http://iew3.technion.ac.il/Home/Users/ierrr01.phtml?YF`)

A note about nested partitioning (NP)

A generic strategy – many possible variations:

- start with whole space: compute its *promise index*
- partition into M subregions (and one more: the rest); compute promise index of each; choose most promising
- repeat. Can permit backtracking etc.

Eg (Chen 00) a product design problem: 10 hours ($\times$ 75 machines) of FATCOP branch-and-bound, vs. 2 minutes NP/GA hybrid; see `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/00-03.ps`

About linear programming (LP)

Many combinatorial optimization problems can be expressed as an *integer linear programming* problem such as:

$$\begin{aligned} \text{Minimize} \quad & \textstyle\sum_i c_i x_i & & (1) \\ \text{subject to} \quad & \textstyle\sum_k a_{jk} x_k & = b_j (\text{various } j) & (2) \\ \text{and} \quad & x_i \text{ integer} & \text{(often limited to a set)} & (3) \end{aligned}$$

Simplex and interior-point algorithms: can be very costly

Example: TSP

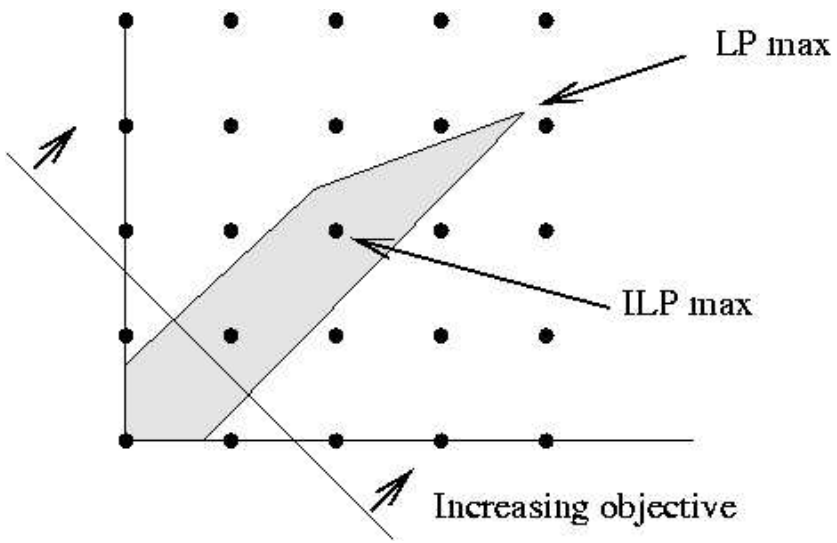
Let $x_{pq} = 1$ if $p \to q$ is a step on the tour, else 0

$$\begin{aligned} \text{Minimize} \quad & \textstyle\sum_{p,q} d_{pq} x_{pq} && && (4) \\ \text{subject to} \quad & \textstyle\sum_{j} x_{ij} = 1 && \text{an edge leaves i} && (5) \\ \text{and} \quad & \textstyle\sum_{k} x_{ki} = 1 && \text{an edge enters i} && (6) \\ & && && (7) \end{aligned}$$

... but messy: far too many variables

LP and ILP

Solve the LP and then round to integer? Not a good idea:



Cutting-plane methods

1. Solve the continuous-valued version

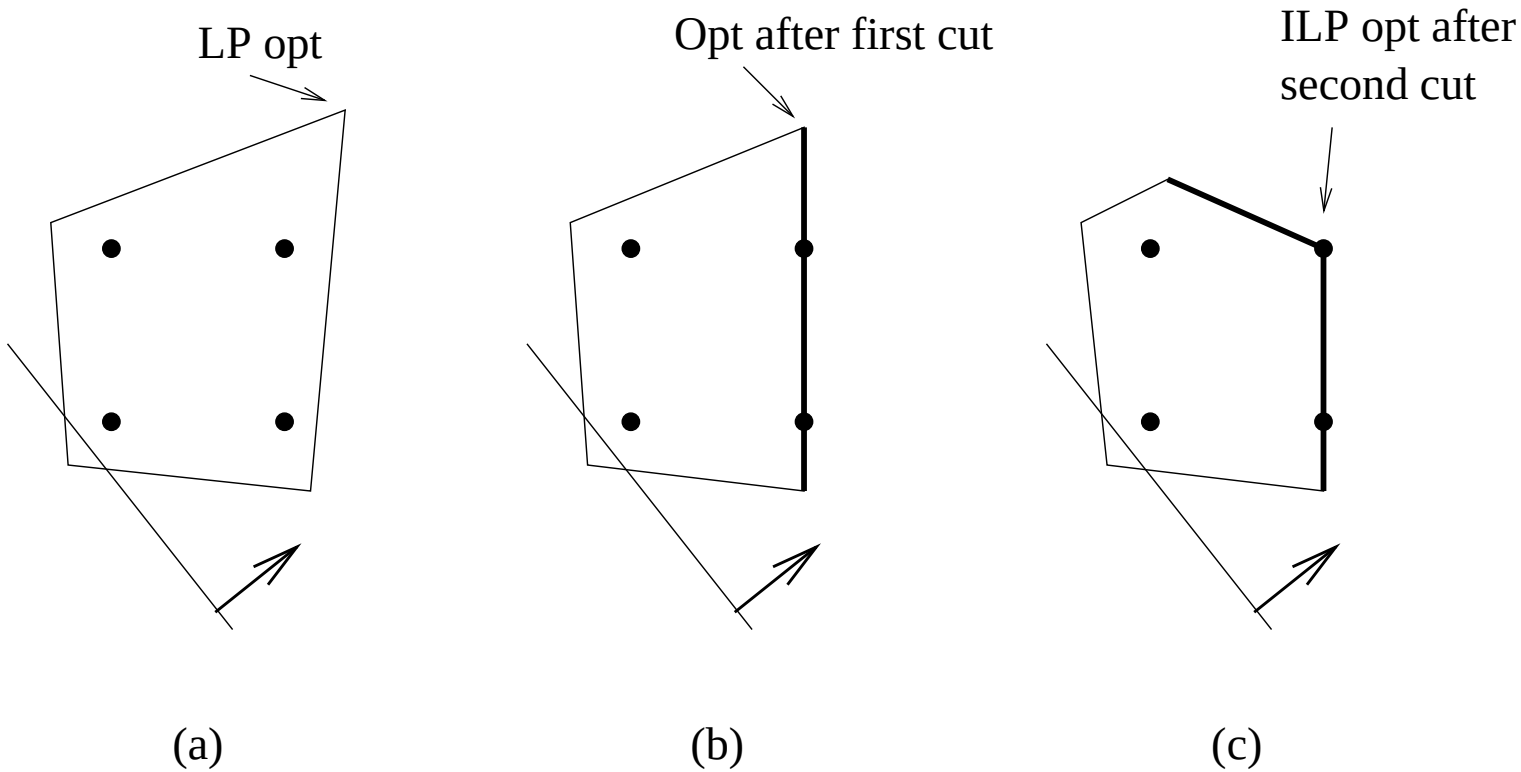
2. No solution? or unbounded? $\Rightarrow$ same for integer-valued problem

3. Integer solution? $\Rightarrow$ solved

4. Facet-identification: find a linear inequality that slices off the non-integer solution but leaves all feasible integer solutions

.. leads to branch-and-cut search methods. Can be stopped early to get good solutions with error bounds.

Cutting-plane methods, continued



EA operators for combinatorics

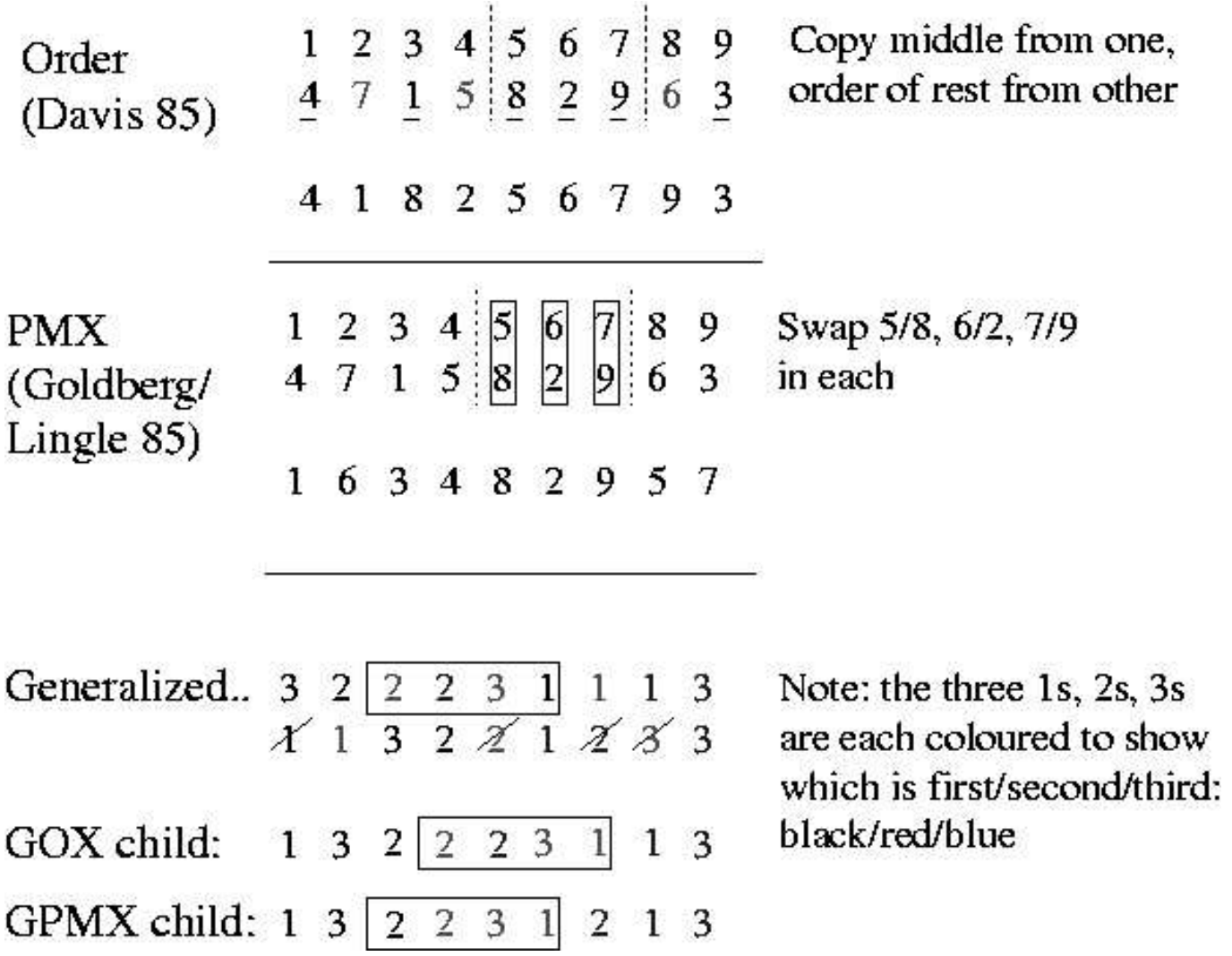
Often, the task is "hunt the permutation":

TSP: permutation = order of visiting nodes (order matters)

QAP: permutation = assignment of resource to task (position matters)

Special EA operators are needed to preserve permutations . . .

Some EA operators for permutations



EA operators for permutations, continued

Mutation: swap a pair; or shift one along; etc

Pick (or design) operators to suit the application – eg, is it position or is it ordering that matters? E.g. in TSP it is ordering that matters, not position

But these are *textbook* recommendations..

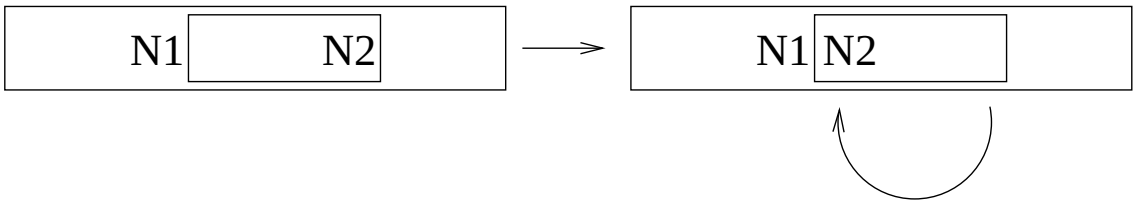
TSP: inver-over (Tao & Michalewicz PPSN V 1998)

For each member M of the population:

1. pick a node N_1 at random

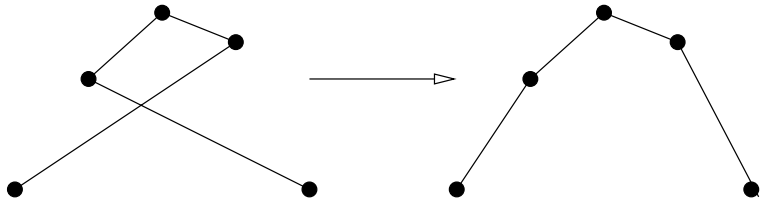
2. choose another member: find the node N_2 that follows N_1 in that member

3. in M, invert a segment to make N_2 be the successor of N_1



TSP

Inver-over is OK for (say) 1000 nodes, but non-EA methods still win:

2-opt step: 

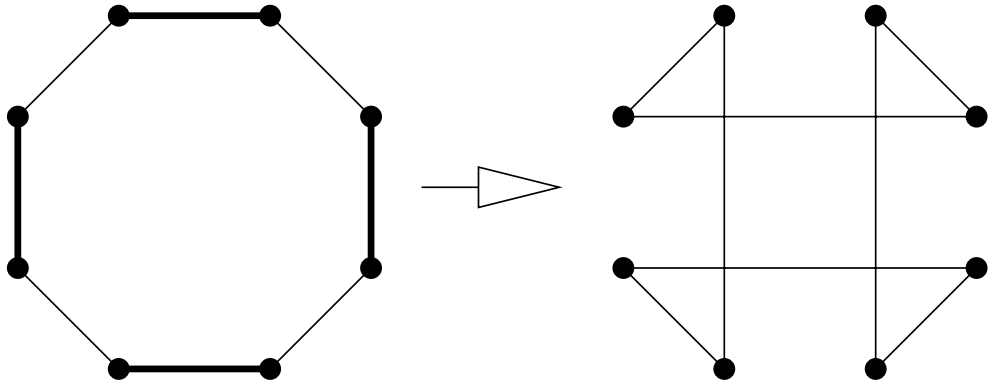
Note: not all two-opt steps are improving ones

Lin-Kernighan: find a sequence of (say) 25 2-opt steps, maybe not each improving, and consider several choices for the first few steps. Repeat.

Iterated Lin-Kernighan: simply re-run L-K a few times

TSP, continued

Chained Lin-Kernighan: like iterated L-K, but new start formed by finding an improving 4-exchange:



Fast and good: has been applied to problems with 25,000,000 nodes (Martin/Otto/Felten 91; Applegate et al 99)

TSP: hybridizing

See Jung and Moon, GECCO 2000, for an EA that uses 4-exchange in mutation and crossover

See Baraglia et al, LNCS 2037, 2001 that uses chained L-K for seeking tour improvements

EAs combined with local search

In general: including plenty of local search helps a lot.

Local search is often the most expensive part by far, but all the ingredients seem to matter.

Permutation-based representations are not always best.

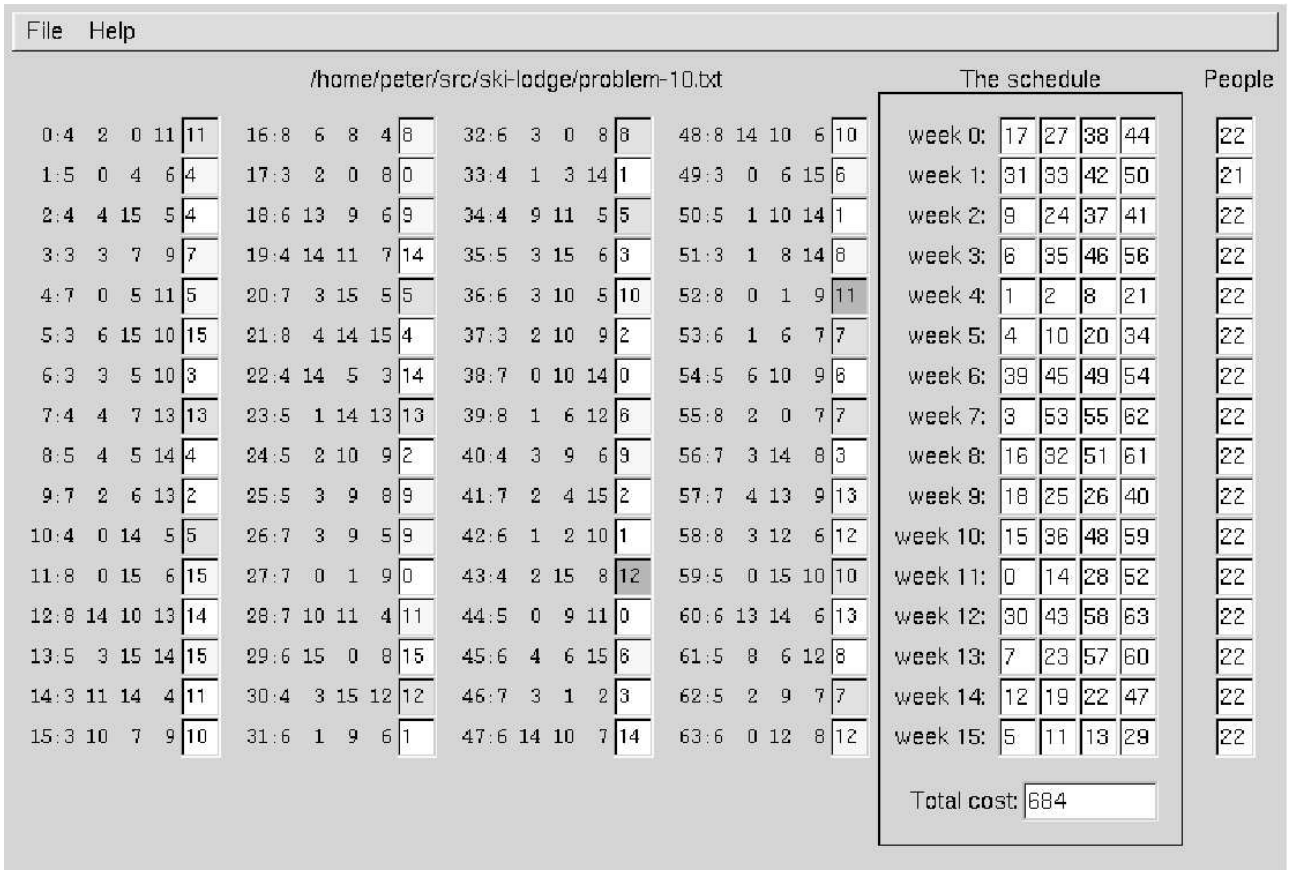
An example: the ski-lodge problem

A four-apartment time-shared lodge:

- 8 beds per apartment, but max 22 people in the building (safety)
- 16-week season: 5 of the 16 are popular choices
- owner states $1^{\text{st}}, 2^{\text{nd}}, 3^{\text{rd}}$ preferences
- compensation for not getting first choice:
  - if 2^{nd}: 2 day-passes per person
  - if 3^{rd}: 4 day-passes per person
  - if other: 7 day-passes per person, $+$ 50 cash-equivalent
  - if none: 1000 day-passes cash-equivalent

The ski-lodge, continued

The task: assign owners to weeks to minimise compensation payout:



The ski-lodge, continued: an EA

Representation: 64 integers: c_i = week for owner i
Initialisation: give each owner one of his three preferences
Repair procedure:

- for each owner in turn, unassign if necessary
- for each week, check if there are free appartments:
  - if just one, find the best-fitting unssigned owner
  - if just two, find the best-fitting pair of owners
  - if more, let crossover and mutation deal with it
  - for 1000 tries, try swapping two assignments

The ski-lodge, continued

The GA used, in Java:

- population size $= 100$
- tournament selection, size 2
- one iteration: choose two parents; create one child by one-point crossover; mutate two genes to be a random one of the given owner's choices; apply repair procedure; child overwrites higher-cost parent if of $\leq$ cost
- run for up to 50,000 iterations (about 25 seconds on modest PC)

The ski-lodge: some results

Problem	Size	Min	Max	Average 25 runs
01	344	641	707	667.48
02	337	404	457	415.88
03	338	450	502	479.92
04	351	732	1616	1362.68
05	315	304	308	305.76
06	328	360	392	373.84
07	347	730	842	787.76
08	326	481	493	484.12
09	316	404	412	406.00
10	351	684	1604	1164.20
11	320	386	408	393.04

The ski-lodge EA: some observations

- more variation on "tight" problems (size close to $16 \times 22 = 352$
- crossover matters - off: $\Rightarrow$ worse
- two-point crossover worse than one-point
- larger tournament size: $\Rightarrow$ worse
- two children per mating: $\Rightarrow$ worse
- child overwrites if of $<$ cost: $\Rightarrow$ worse
- mutating 1 gene, or 3 genes: $\Rightarrow$ worse
- popsize 50 or 150: $\Rightarrow$ worse

The ski-lodge: more observations

A well-tuned simulated-annealing algorithm does a little worse than the EA - on every problem!

Results can sometimes be improved: a different EA does better on the "tight" problems, worse on the others

75 students each implemented an EA: permutation-based ones performed somewhat worse

Source code, problem generators, results etc at:

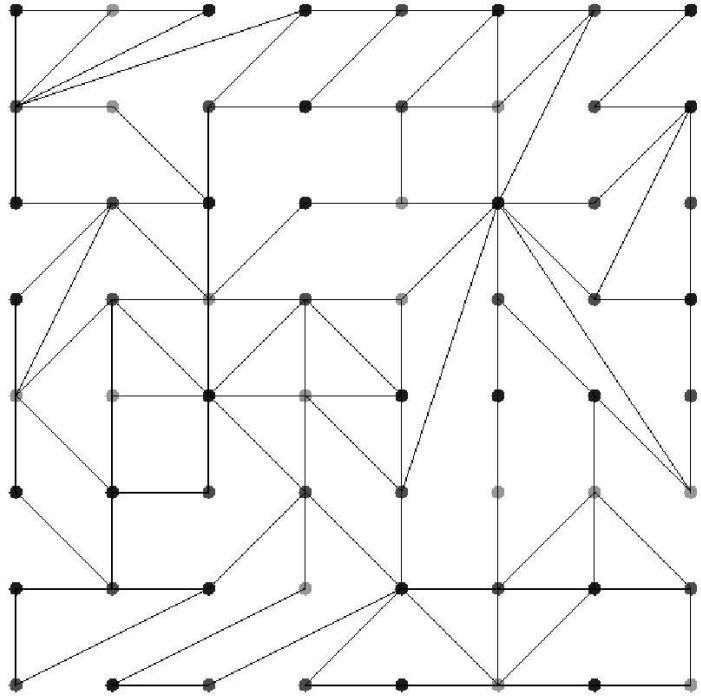
`http://www.dcs.napier.ac.uk/~peter/ski-lodge/`

A good EA is important – for setting compensation rates!

EAs and constraint satisfaction

Example: graph coloring

- color the nodes using as few colors as possible
- edge ends must have different colors
- eg: nodes = events, colors = timeslots
- or: nodes = radio stations, colors = frequencies



`http://www-unix.mcs.anl.gov/~benson/dsdp/`

Constraint satisfaction generally

Given: a set of variables $x_1, x_2, \ldots x_n$
- the possible values for each x_i
- constraints (disallowed value combinations)

Find: an assignment that obeys all constraints (or maybe the best assignment)

Common non-EA techniques:

- chronological backtracking with forward-checking
- forward-checking with constraint-directed backjumping (Prosser 93)

EA approaches to constraint satisfaction

n variables $\Rightarrow$ chromosome $c[]$ of length n

Integer representation: $c[i]$ is the value of variable x_i

Order representation: $c[]$ is a permutation of $1 \cdots n$

To decode: consider each variable in turn, in the order given

- find an allowed value for it
- backtrack as necessary
- no legal assignment $\Rightarrow$ lousy fitness

Paredis's Co-evolutionary EA (94-95)

Two populations:

assignment:	integer representation; population is evolved; fitness based on how well the member solves a sampling of constraints
constraint:	all the constraints; population not evolved; member fitness depends on the assigments it manages to defeat

Fitnesses continually updated by *encounters* between members of the two populations, chosen by linear ranking

Stepwise adaptation of weights (Eiben et al 95-98)

Adds a vector of weights, one per constraint. All weights initially 1.

Let $v_{ij} = 1$ if there is a violation caused by x_i and x_j, else 0

Fitness $= \sum_{i,j} w_{ij} v_{ij}$ – big means bad

Every ΔT iterations (typically 250):

- find best (lowest fitness) member
- for each constraint violated by it: $w_{ij} = w_{ij} + 1$
- re-evaluate every member

Zooming adaptation (van Hemert 02): like stepwise adaptation, but there is a separate weight for each disallowed pair of values.

Some other EAs

Falkenauer's grouping GA (92-94):

- mainly for graph-coloring and grouping
- chromosome: eg nodes=ABBACBA : groups=BAC
  group part used to handle violations
- fancy crossover, mutation and inversion

Dozier's Microgenetic Iterative Descent (93-95):

- elaborate representation, tracks violations by variables
  so that it "can know when to quit"
- tracks which variables are most troublesome
- weights 'nogoods' (disallowed value combinations)
- fitness somewhat like stepwise adaptation

Performance (thanks to Jano van Hemert)

On 1000-variable problems of varying constraint density and tightness (randomly created, not all solvable):

- stepwise adaptation is fast and pretty good
- Dozier's MID is slower, a little better
- others are poorer
- .. but non-EA methods still win; and can handle much bigger problems too

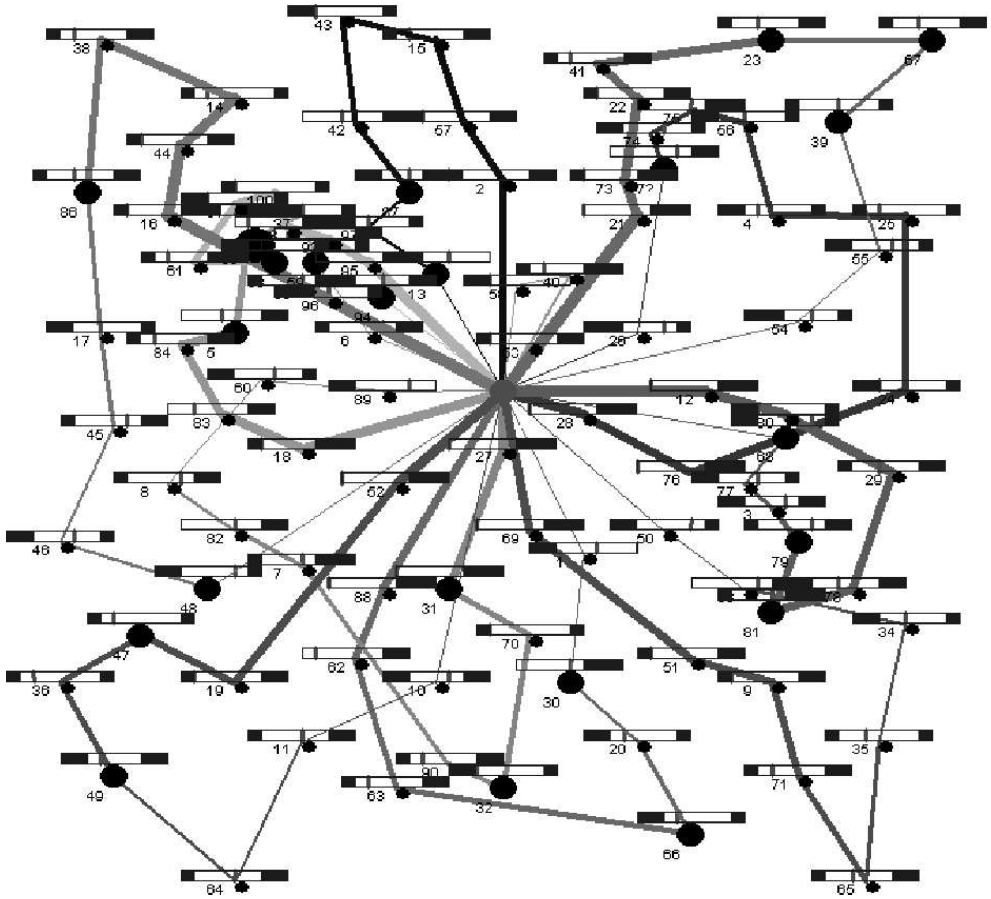
Vehicle routing with time windows (VRPTW)

Typically: one depot; lots of vehicles with known capacity; lots of customers each needing a delivery of some varying amount; each customer has a time-windows within which delivery must happen.

Vehicles must all return to depot by a given deadline.

Aims: *minimise vehicles used (one per trip)*
and *minimise total distance travelled*

VRPTW, continued

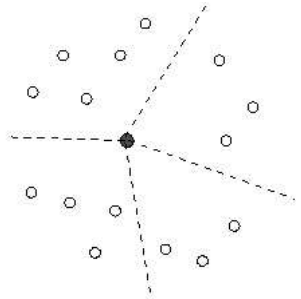


http://www.pratix.hu/OptOnline/Vrp/FlexGrid/VRPDescription.asp

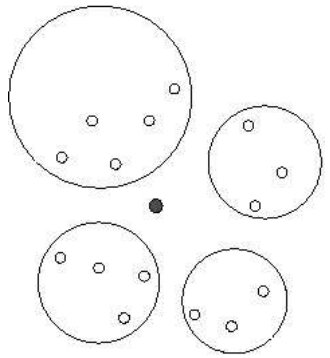
VRPTW, continued

Early ideas (Thangiah, 93-94):

- GA chooses sector angles only



- GA chooses cluster centres/sizes



VRPTW: the Solomon problems

Six sets: R1 (12), R2 (11), RC1 (8), RC2 (8), C1 (9), C2 (8)

see `http://www.idsia.ch/~luca/macs-vrptw/problems/welcome.htm`

R* : 100 randomly-placed customers (same for all)
C* : 100 clustered customers (same for all)
RC* : 100 mixed random and clustered (same for all)
*1 : tight time windows, small vehicle capacity
*2 : wide time windows, large capacity

See `http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm`
for problems with up to 1000 customers

VRPTW: Gambardella's ant colonies 99

First: a basic TSP single ant-colony system:

- each ant is assigned to a random node, tries to build a tour:
  - with prob p choose highest-pheromone arc, else choose stochastically according to pheromone level. Level is decreased on chosen branch
  - repeat, until complete tour
  - improve tour by local search
- best solution found is used to strengthen pheromone levels
- restart with new ants, until tired/timed out/stagnant

VRPTW: two ant colonies

ACv aims to reduce vehicles, ACd aims to reduce total distance

```
(bestSoFar, V) = localSearchResult();
repeat {
  start ACv with V-1 vehicles;
  start ACd with V vehicles;
  while ( both running ) {
    watch for improved bestSoFar;
    if ( bestSoFar needs < V vehicles )
    then stop both colonies;
  }
} until ( tired );
```

VRPTW: two ant colonies

ACv may build an incomplete tour – 'better' if it visits more customers. But it only anounces tours that visit all customers.

In ACv, best feasible and best incomplete tours are used to update pheromones.

Both ACv and ACd use a lot of local search, eg: for each customer there is a precompiled list of the 20 nearby customers.

Local search tries swaps and moves of customers between routes.

VRPTW: ant colonies

Results in 99 were good, eg R112: 9 vehicles, distance 982.14

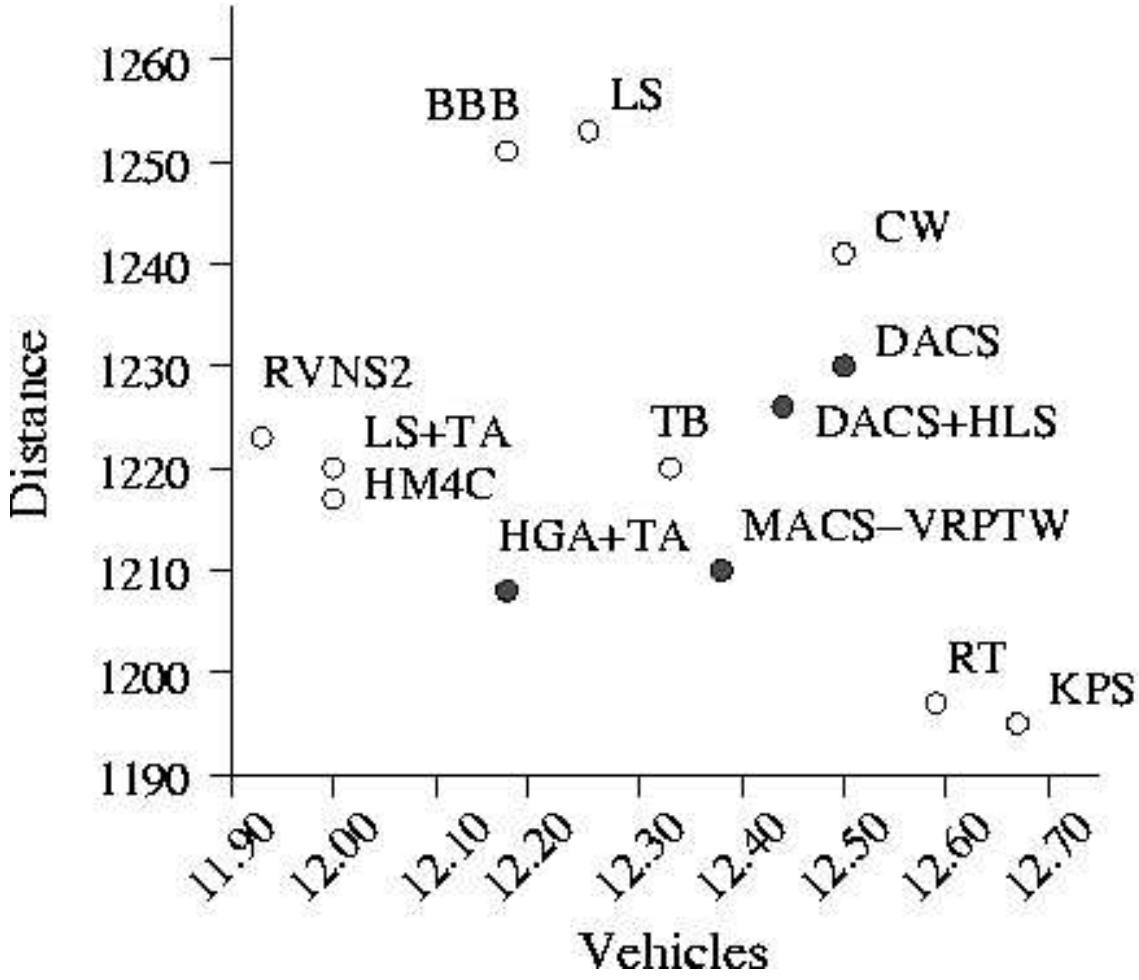
Since then: many more forms of ant colony model – see 2004 book by Marco Dorigo

AntOptima (www.antoptima.com) is marketing solutions

Ant colony methods related to Rubinstein's simulated entropy approach `http://iew3.technion.ac.il/~ierrr01/PAPERS/noisy.ps`

But ... ant colonies not currently best on Solomon problems

VRPTW: comparisons on R1 (averaged)



A recent idea: hyper-heuristics

Some objections to EAs and other heuristic search methods:

- they improve, but no guarantees
- poor understanding of worst-case behaviour
- they are often 'black box'
- lots of parameters, lots of design choices to make
- usually tested on small set of benchmark problems

Real users sometimes like simple heuristics instead

.. but simple heuristics have flaws

Bin-packing: pack 12, 11, 11, 7, 7, 6 into bins of size 20.
Best-fit: $7 + 7 + 6$ exactly fills a bin but $\Rightarrow$ suboptimal

Djang & Finch algorithm:

```
use largest items to fill a bin to > capacity/3;
for(i=0; i <= freeSpace; i++) {
  seek one item of size (freeSpace-i);
  else two items of size (freSpace-i);
  else three items of size (freeSpace-i);
}
```

Good for 'hard' problems, terrible for easy problems

Hyper-heuristics: the concept

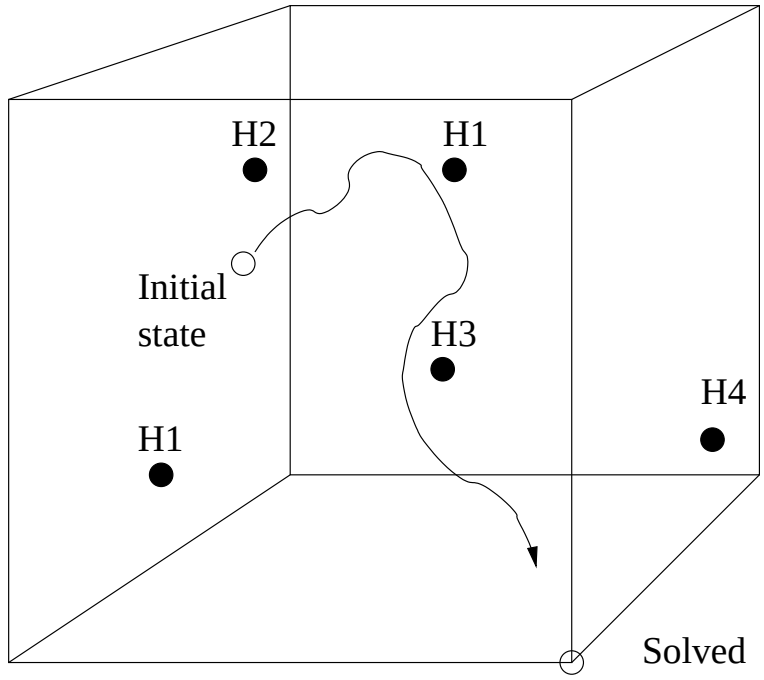
Rather than solving individual problems ...
Use search methods to find an algorithm that combines simple heuristics that offers good, fast performance on a whole set of problems

Can be solution-constructing or solution-improving

Search can also visit 'pure' simple heuristics: losers don't survive

An algorithm framework: solution-constructing

- choose simplified state representation
- choose set of heuristics
- EA searches for a set of labelled points
- label = heuristic for next step
- the algorithm:
  repeat: find closest labelled point, apply its label



Example: timetabling

State: % of 'awkward' events left to place
- % of 'average' events left to place
- % of 'easy' events left to place
- crude resource estimates
- % events remaining

Heuristics: (say) 8 each of event-chooser and slot-chooser

Point labels: either an event-chooser or a slot-chooser

Some results

The EA: a simplified messy GA (see CEC 2004 for details)

The problems: Carter's (real) university exam timetabling problems – up to 30,000 students and 2,400 exams; and the class timetabling problems from the International Timetabling Competition

`http://www.idsia.ch/Files/ttcomp2002/`

Generated algorithm finds good feasible solutions
without backtracking or any search!

Hyper-heuristics, continued

Also works well on a set of 1000+ hard bin-packing problems – excellent worst-case behaviour

Another algorithm framework: a classifier system:
"state $\longrightarrow$ heuristic" rules

See `http://www.asap.cs.nott.ac.uk` for some papers, including solution-improving approaches

Finally. . .

- still hard to decide when/if to use an EA
- the best methods are hybrids: often with *lots* of local search
- past research has focused on doing well on benchmarks, and solving individual problems
- getting good worst-case behaviour is important
- in the future: systems that learn about *your* problems and adapt to get better at them? Development vs. evolution