COMP8620
Lecture 5-6

Neighbourhood Methods, and Local Search

(with special emphasis on TSP)

Assignment

http://users.rsise.anu.edu.au/~pjk/teaching

- "Project 1"

Neighbourhood

- For each solution $S \in \mathcal{S}$, $\mathcal{N}(S) \subseteq \mathcal{S}$ is a neighbourhood
- In some sense each $T \in \mathcal{N}(S)$ is in some sense "close" to S
- Defined in terms of some operation
- Very like the "action" in search

Neighbourhood

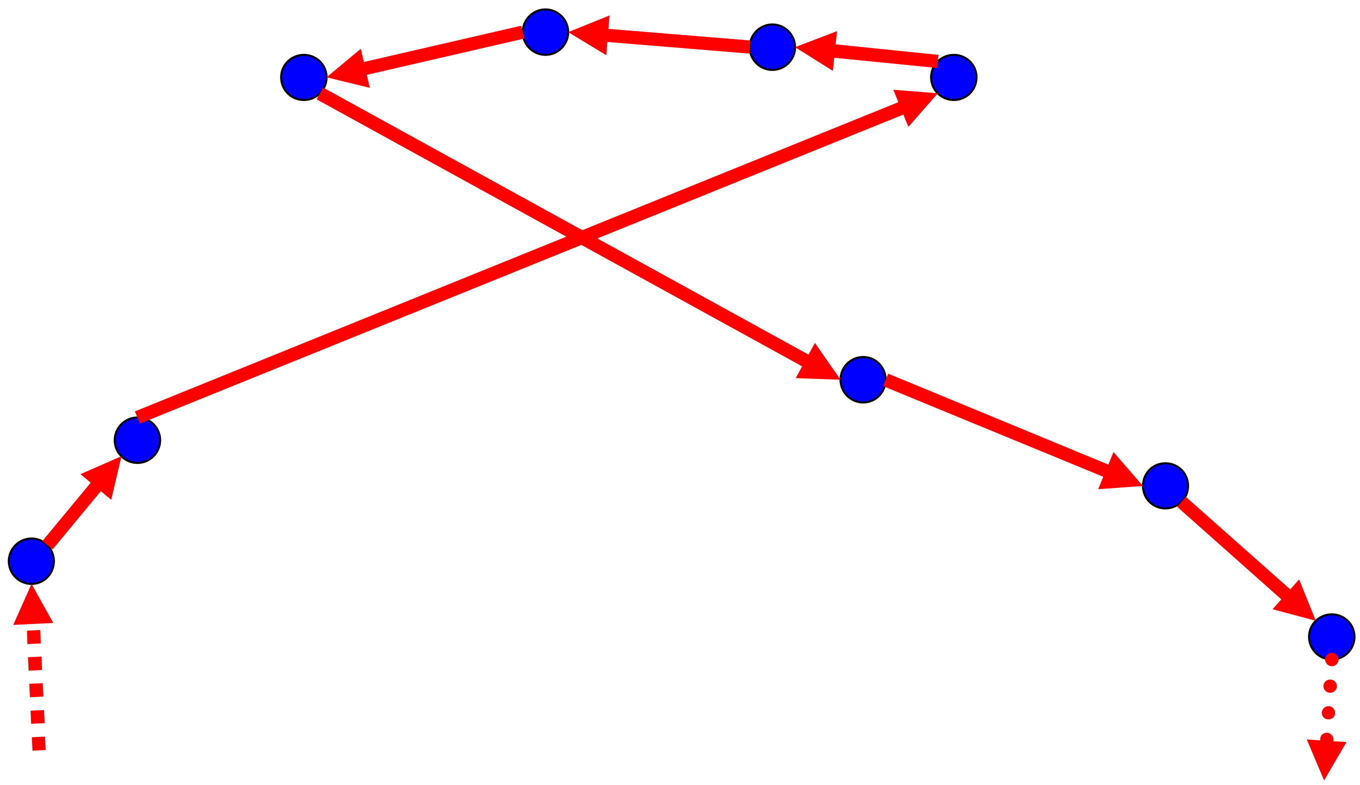
Exchange neighbourhood:

Exchange k things in a sequence or partition

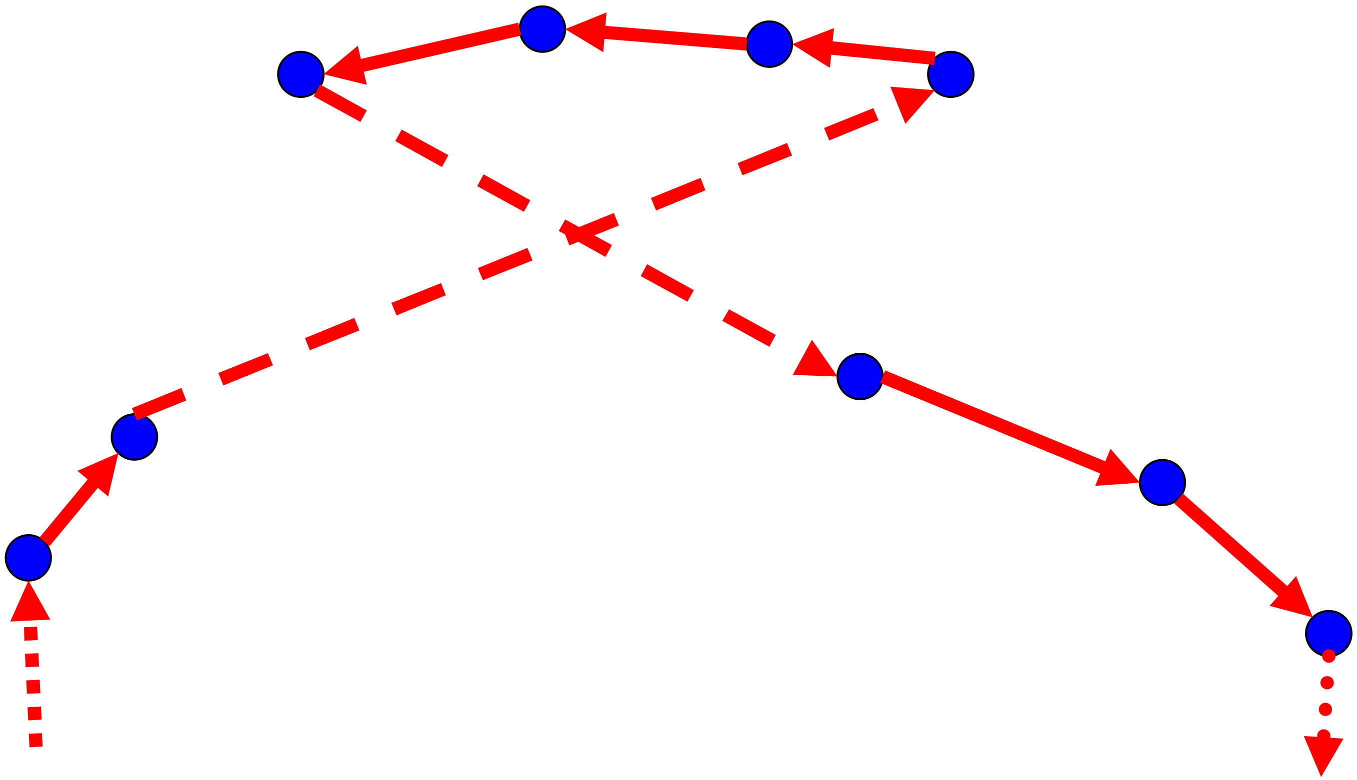
Examples:

- Knapsack problem: exchange k_1 things inside the bag with k_2 not in. (for $k_{i,}$ k_2 = {0, 1, 2, 3})
- Matching problem: exchange one marriage for another

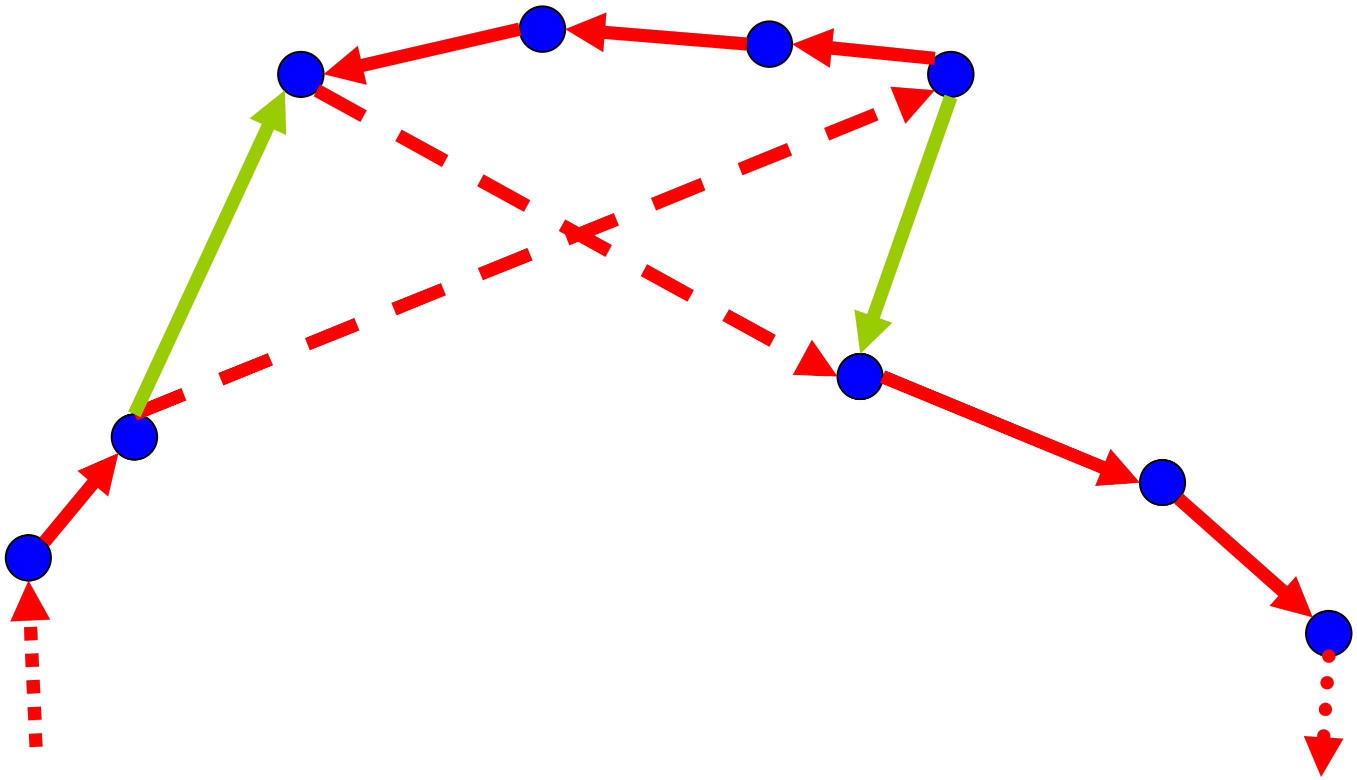
2-opt Exchange



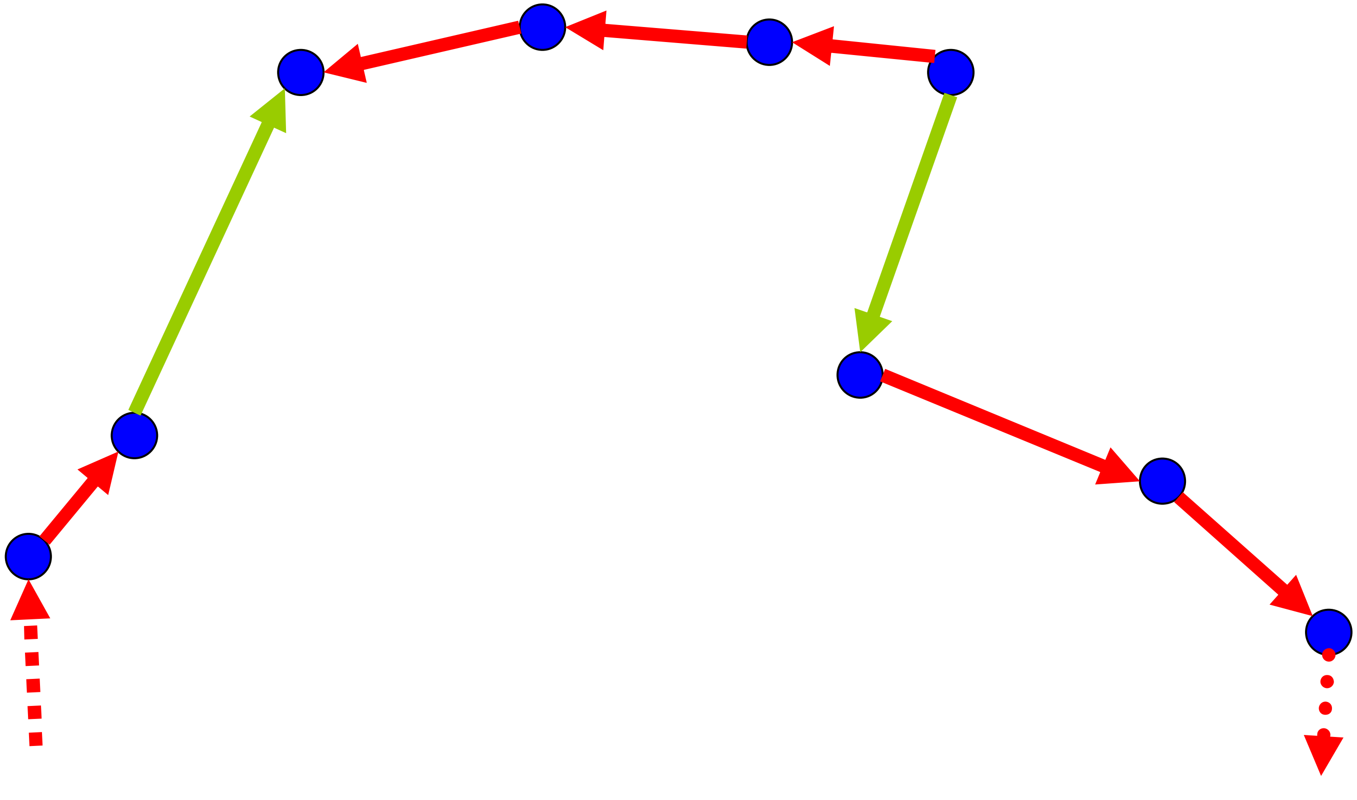
2-opt Exchange



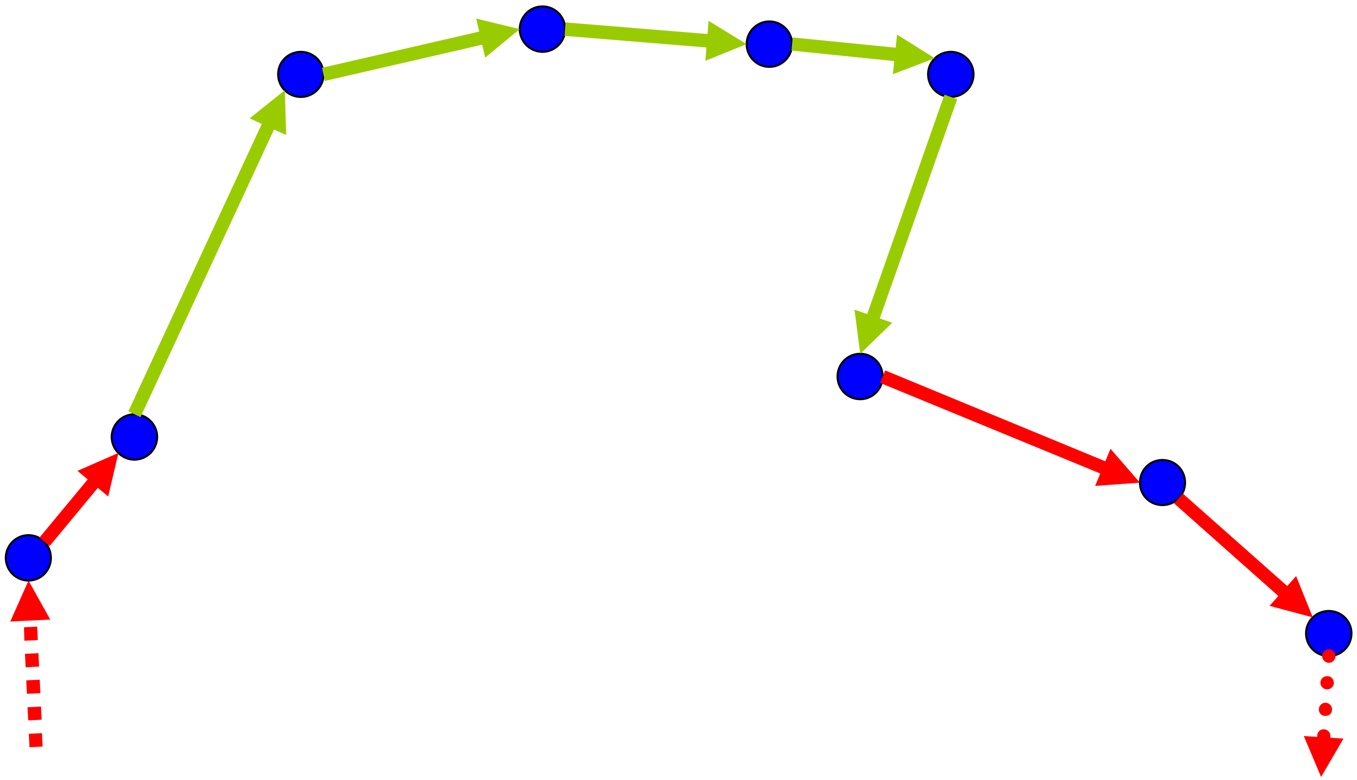
2-opt Exchange



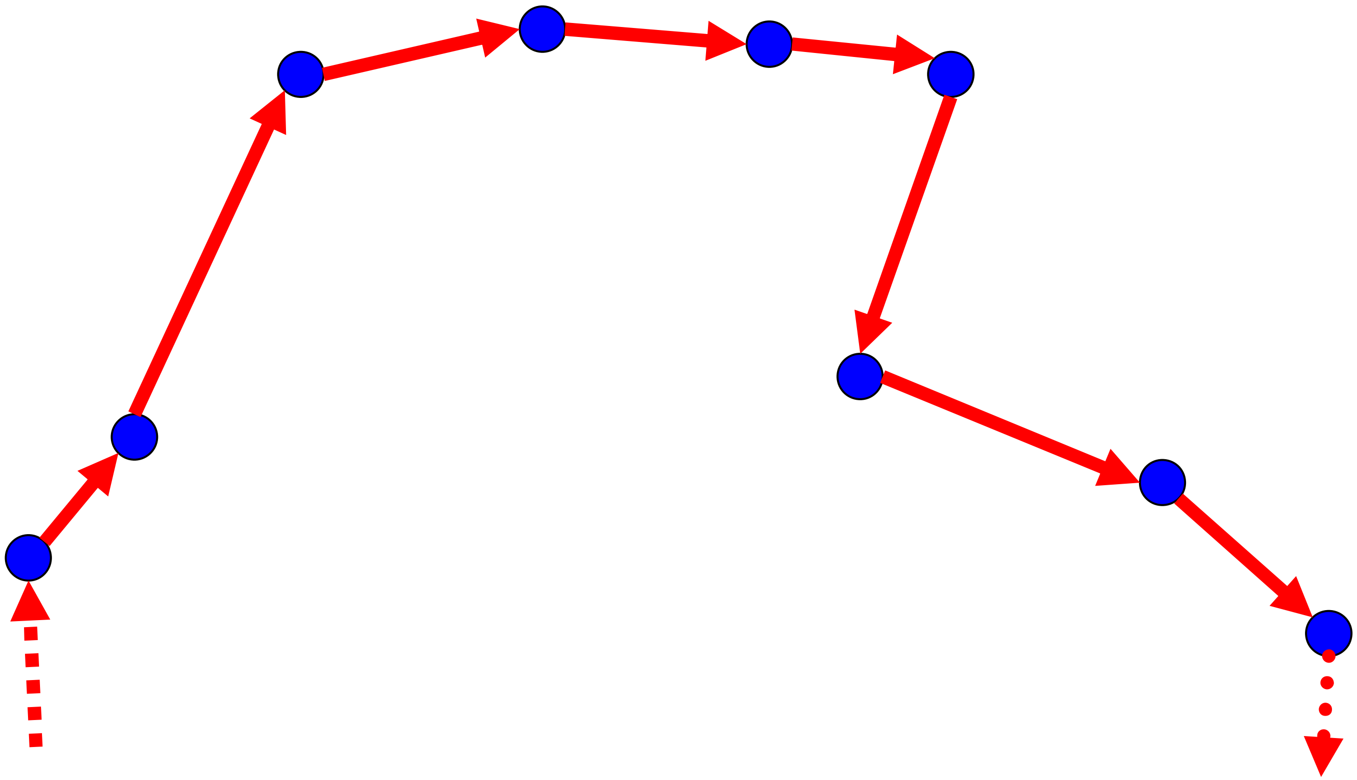
2-opt Exchange



2-opt Exchange



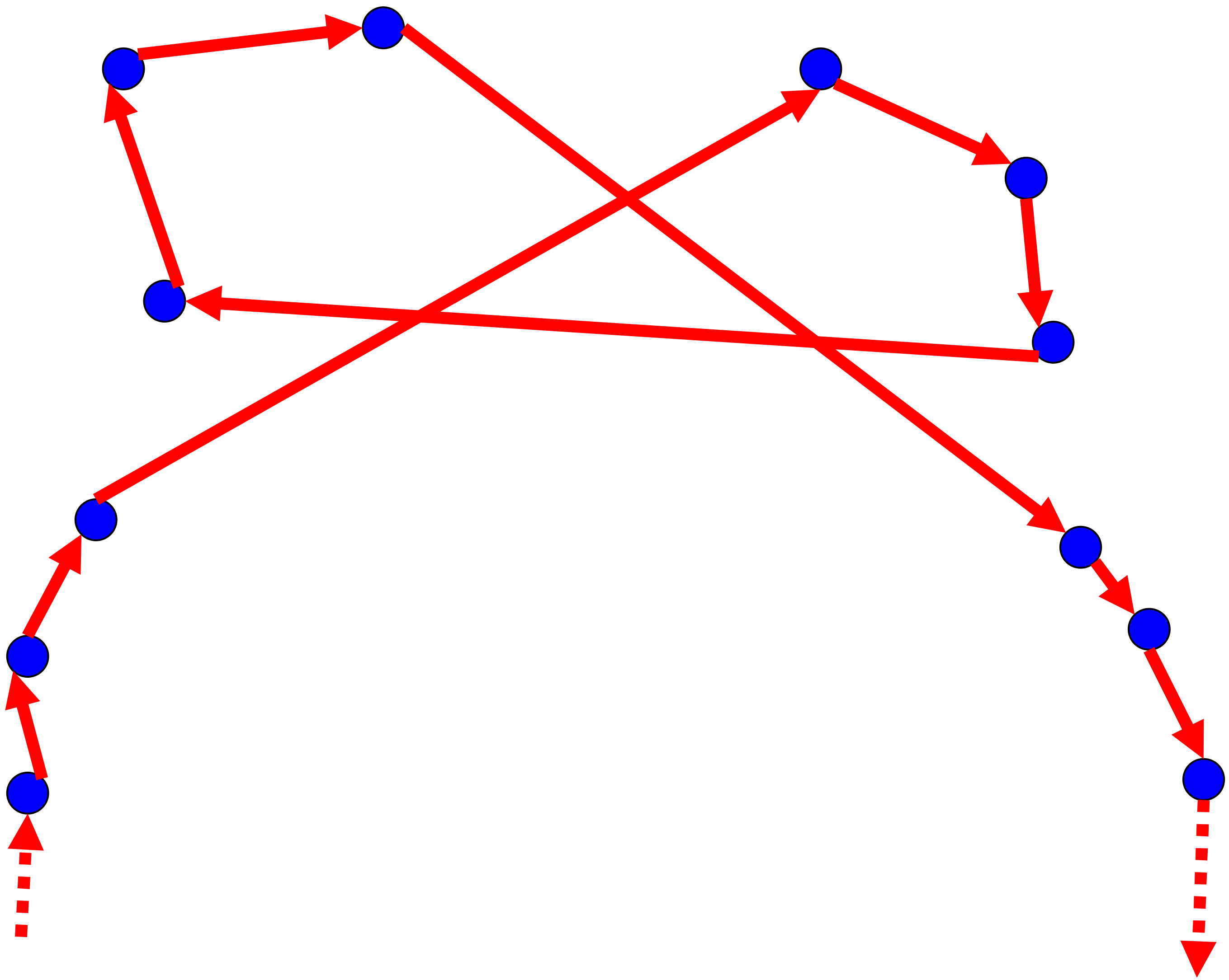
2-opt Exchange



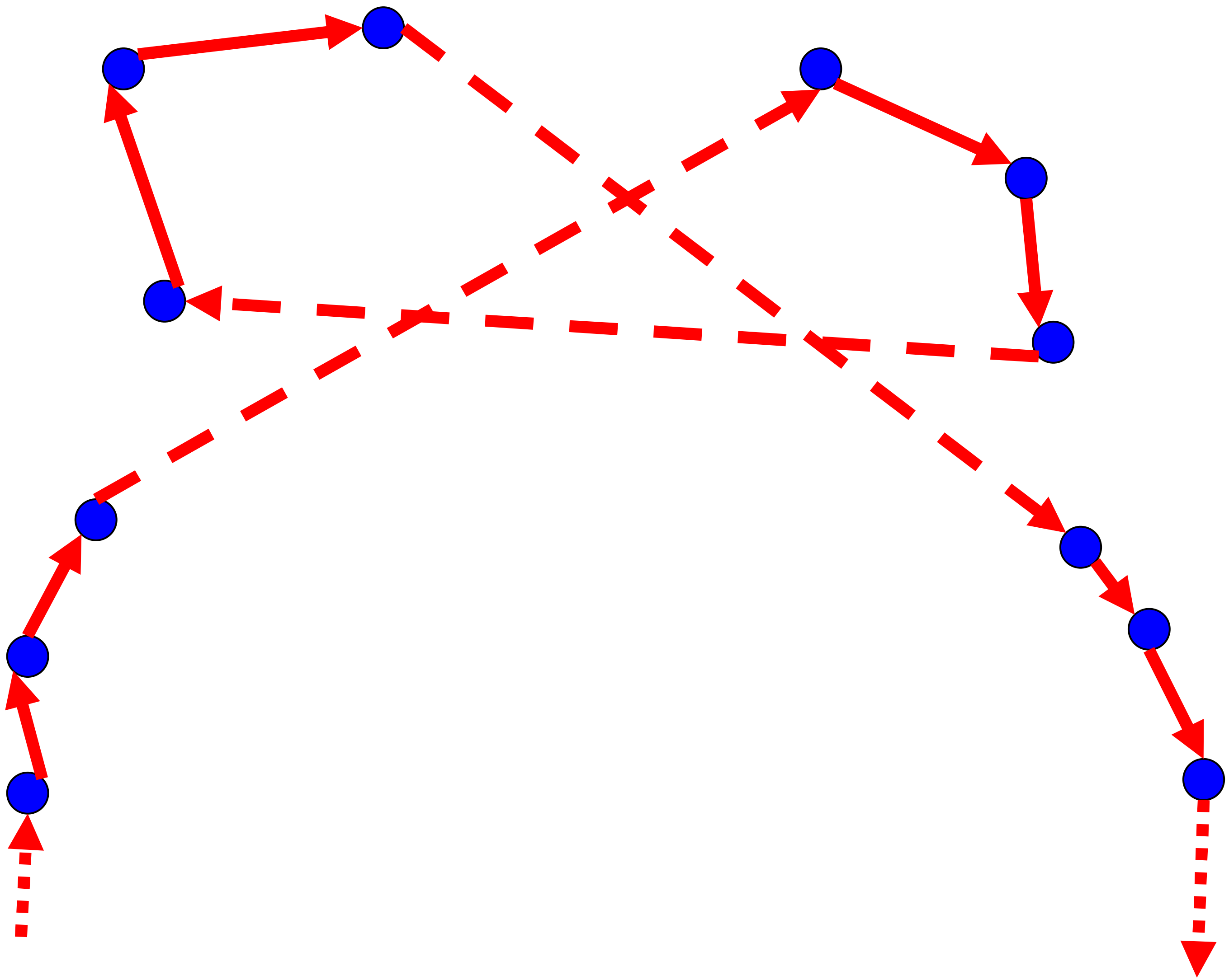
3-opt exchange

- Select three arcs
- Replace with three others
- 2 orientations possible

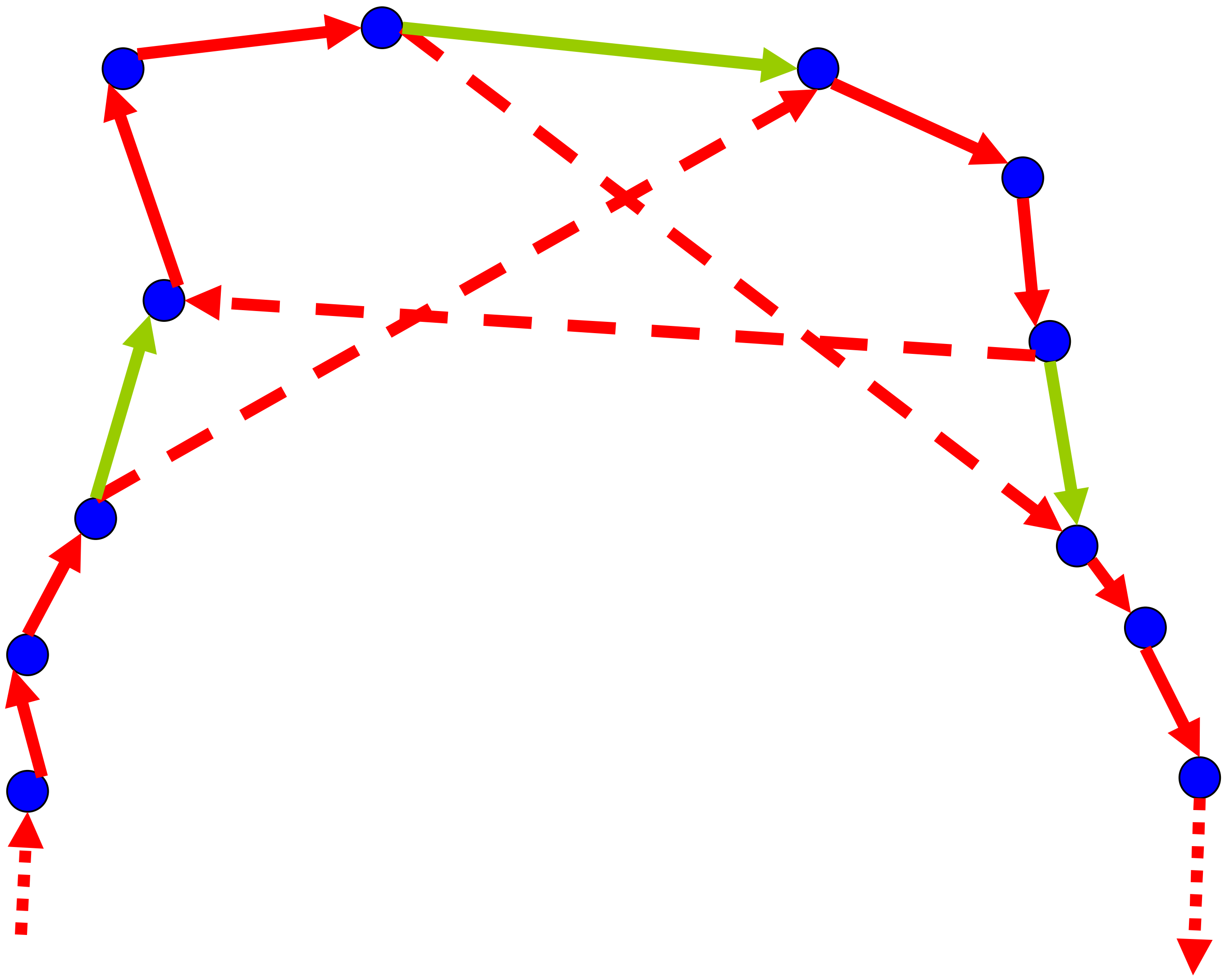
3-opt exchange



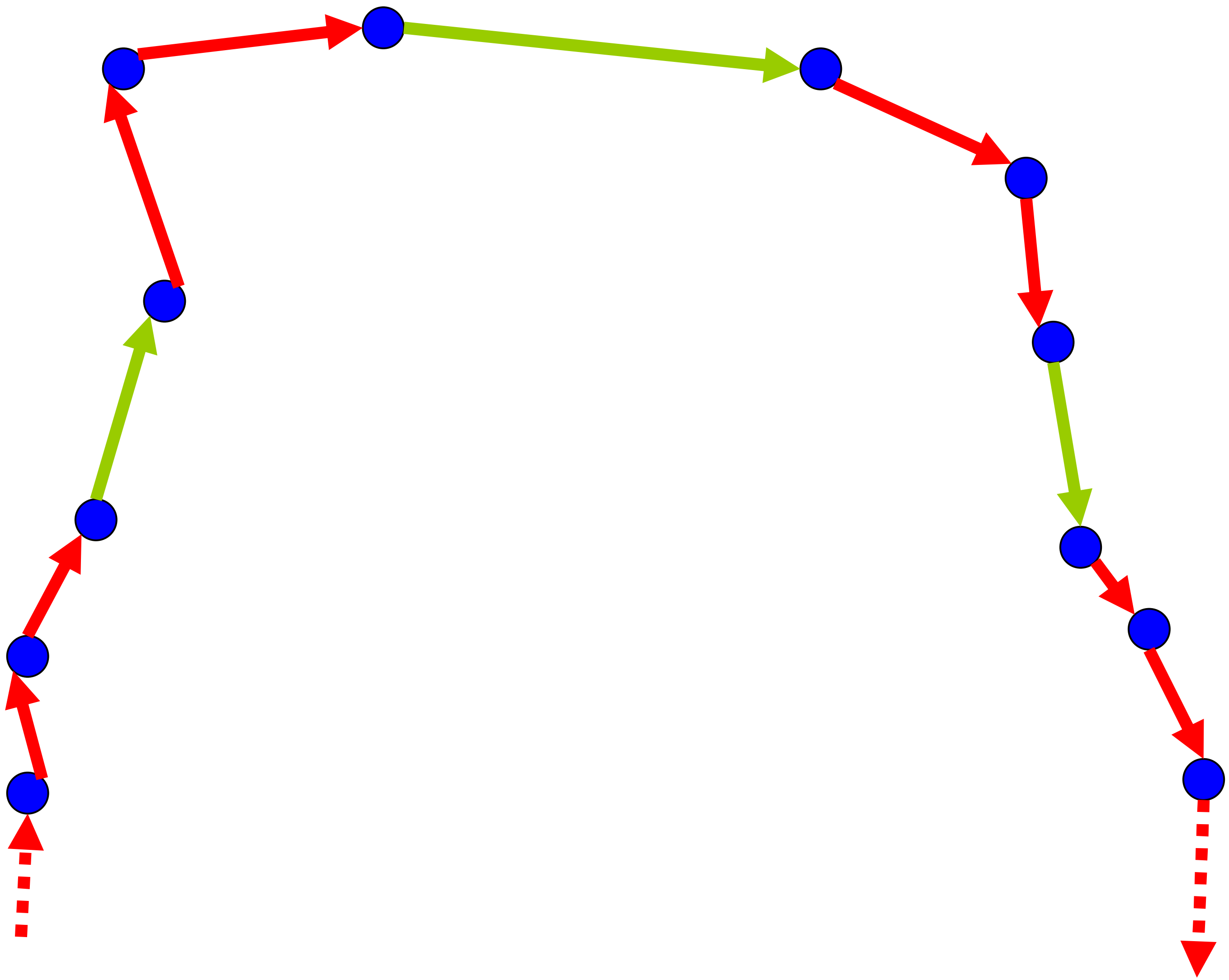
3-opt exchange



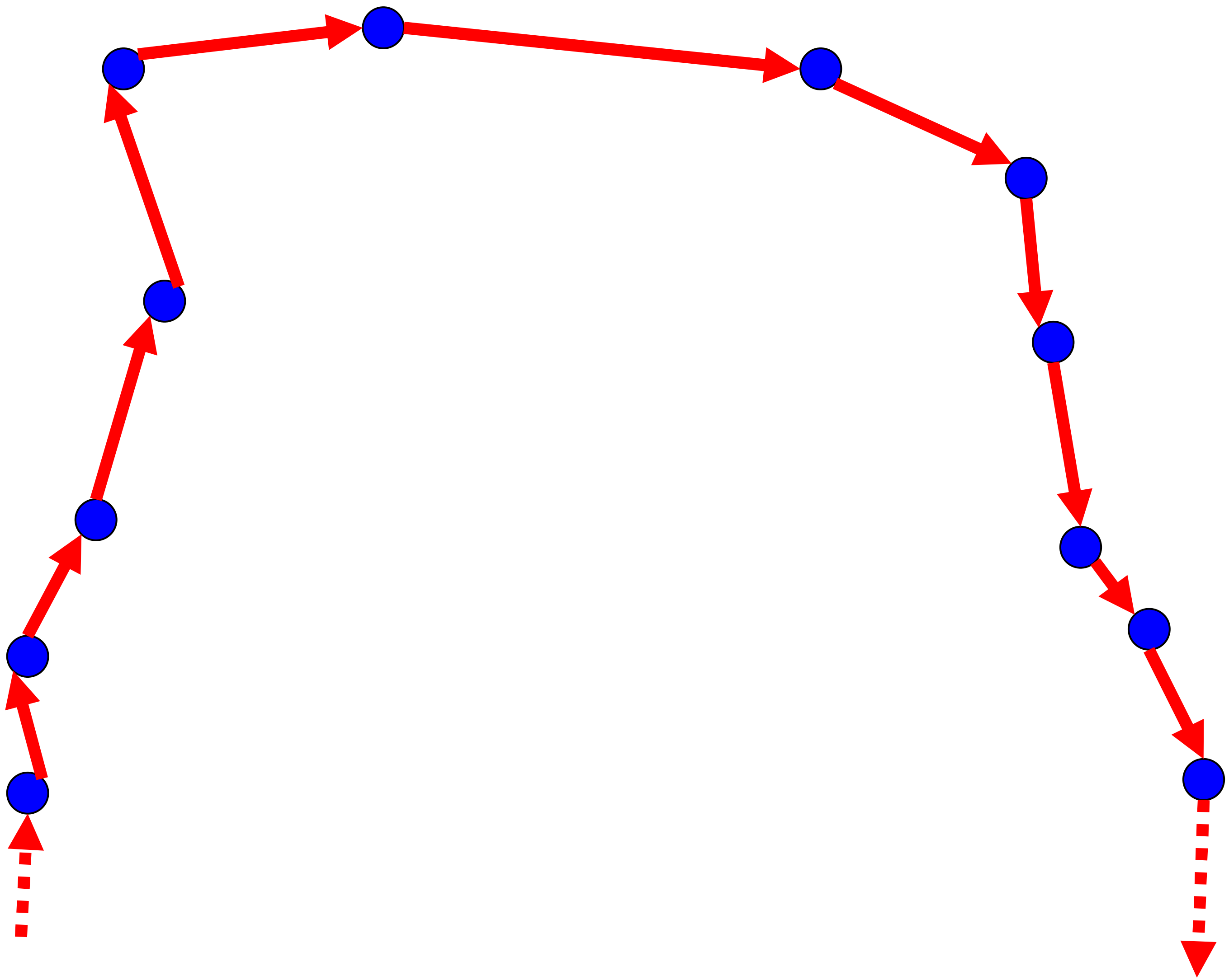
3-opt exchange



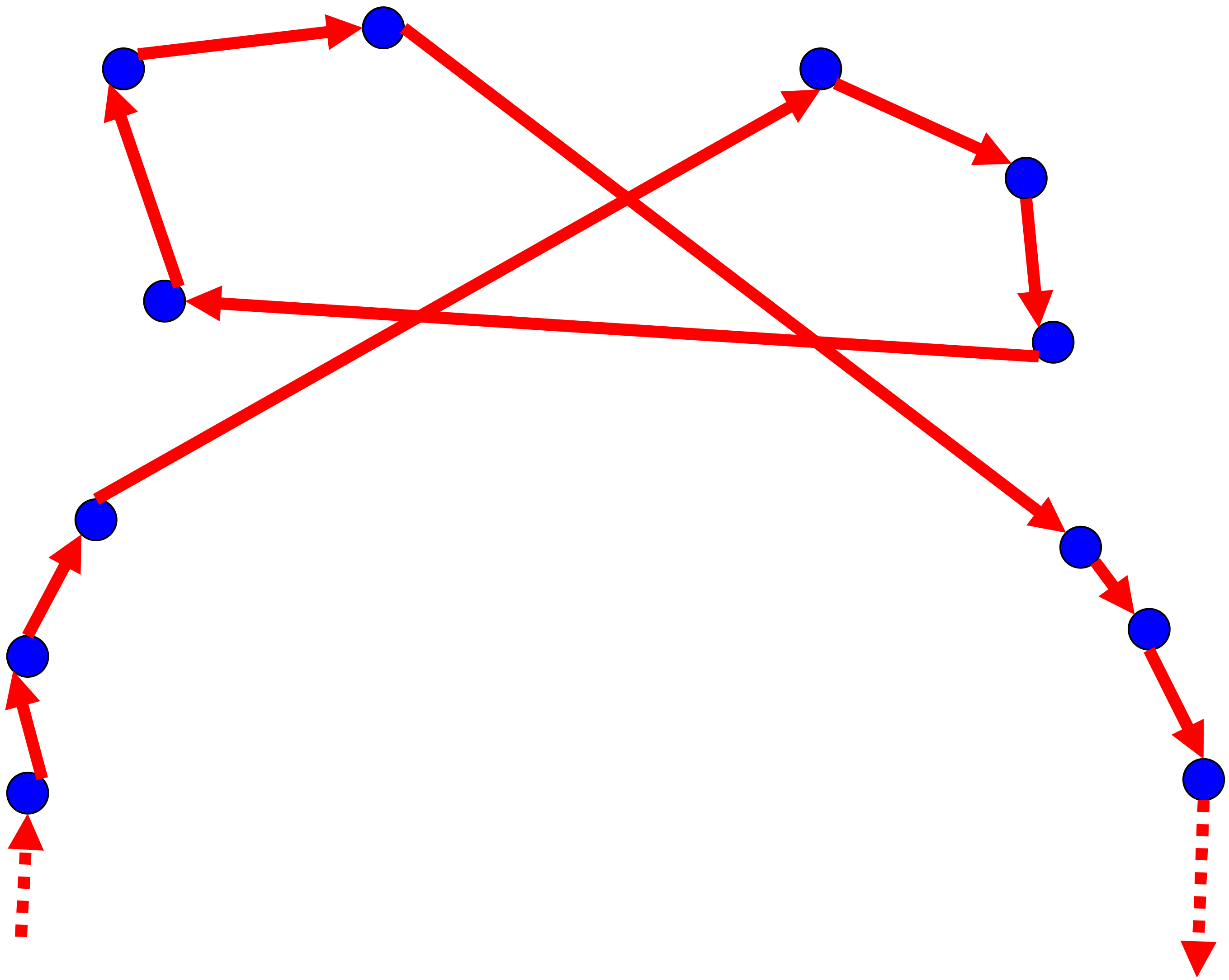
3-opt exchange



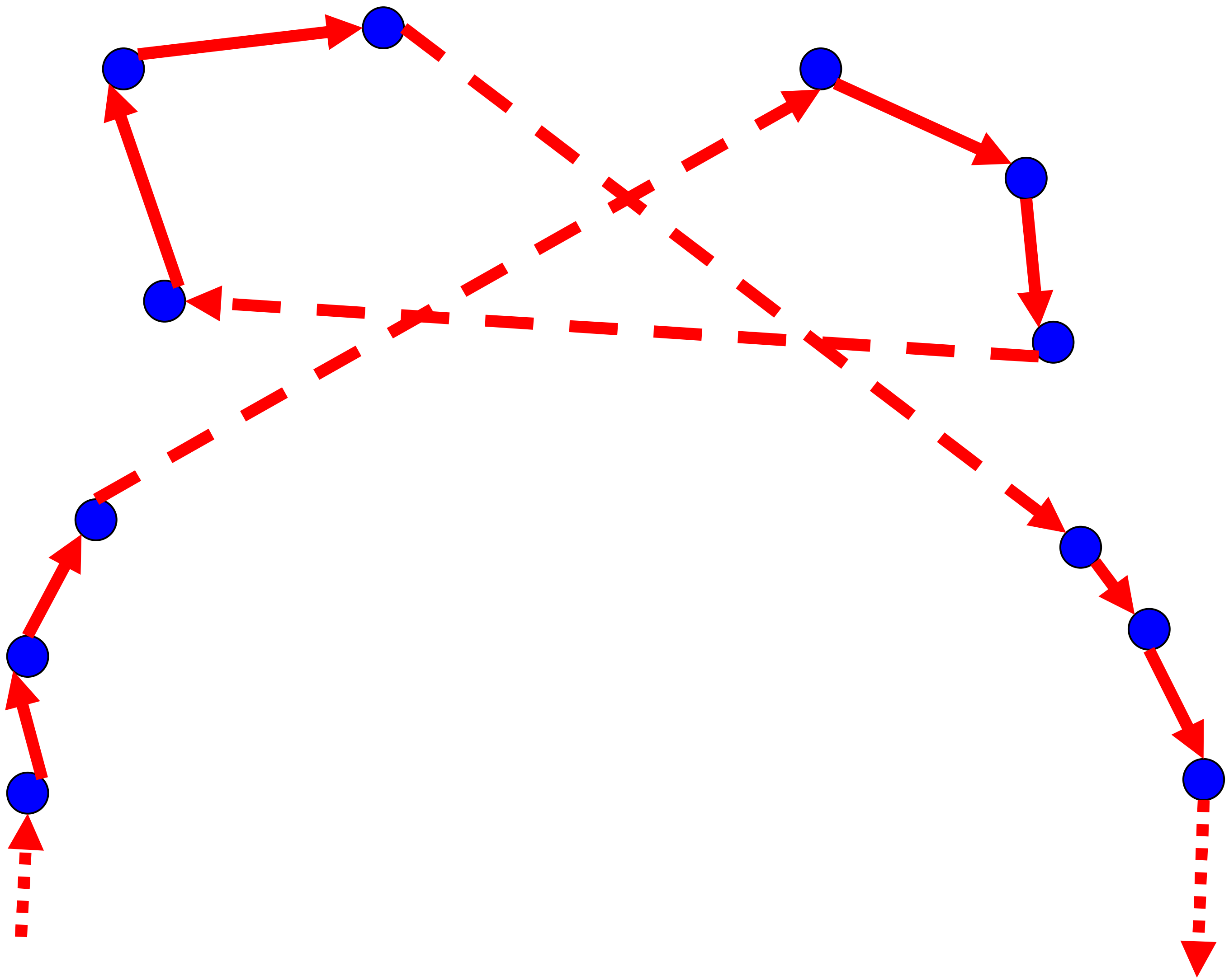
3-opt exchange



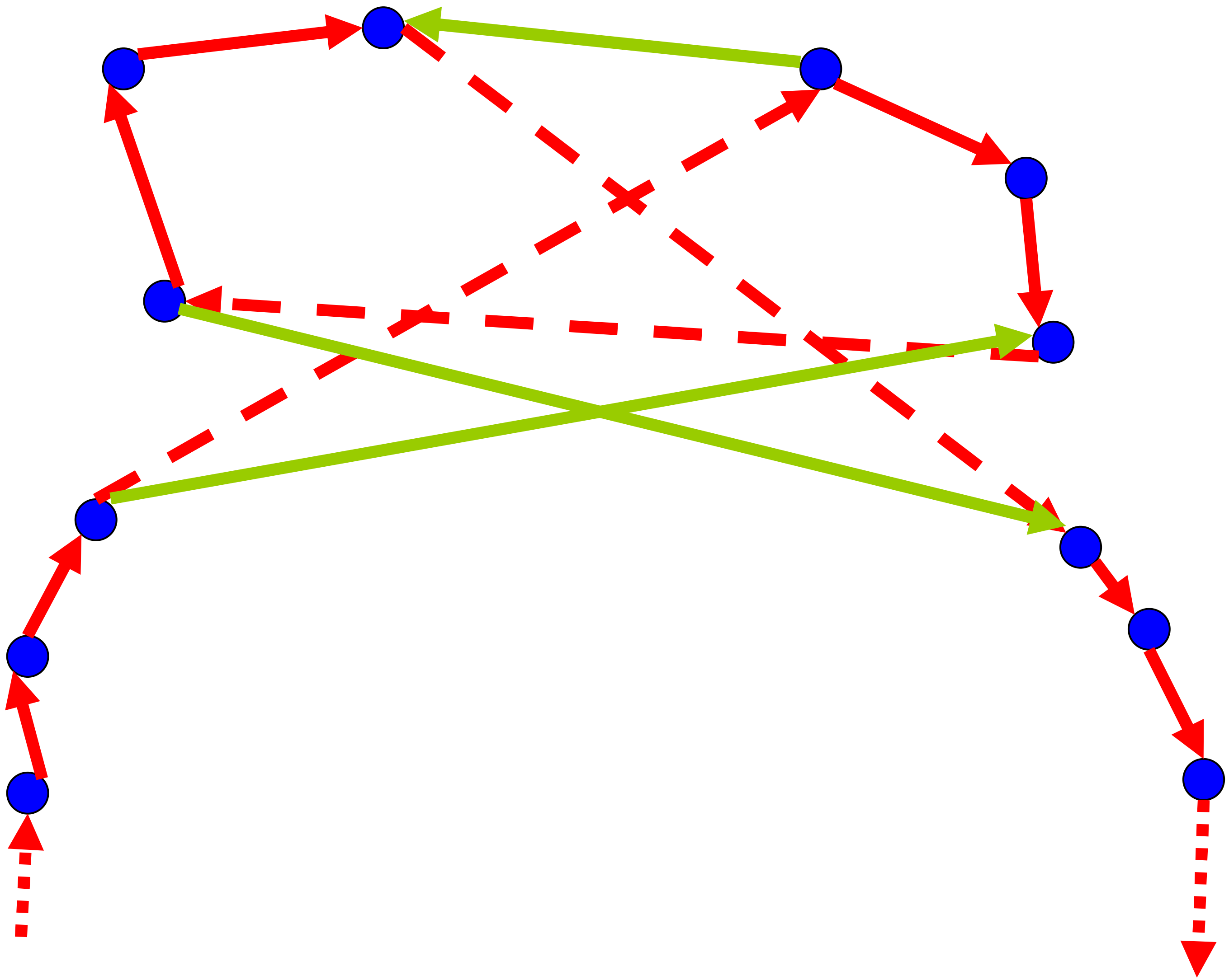
3-opt exchange



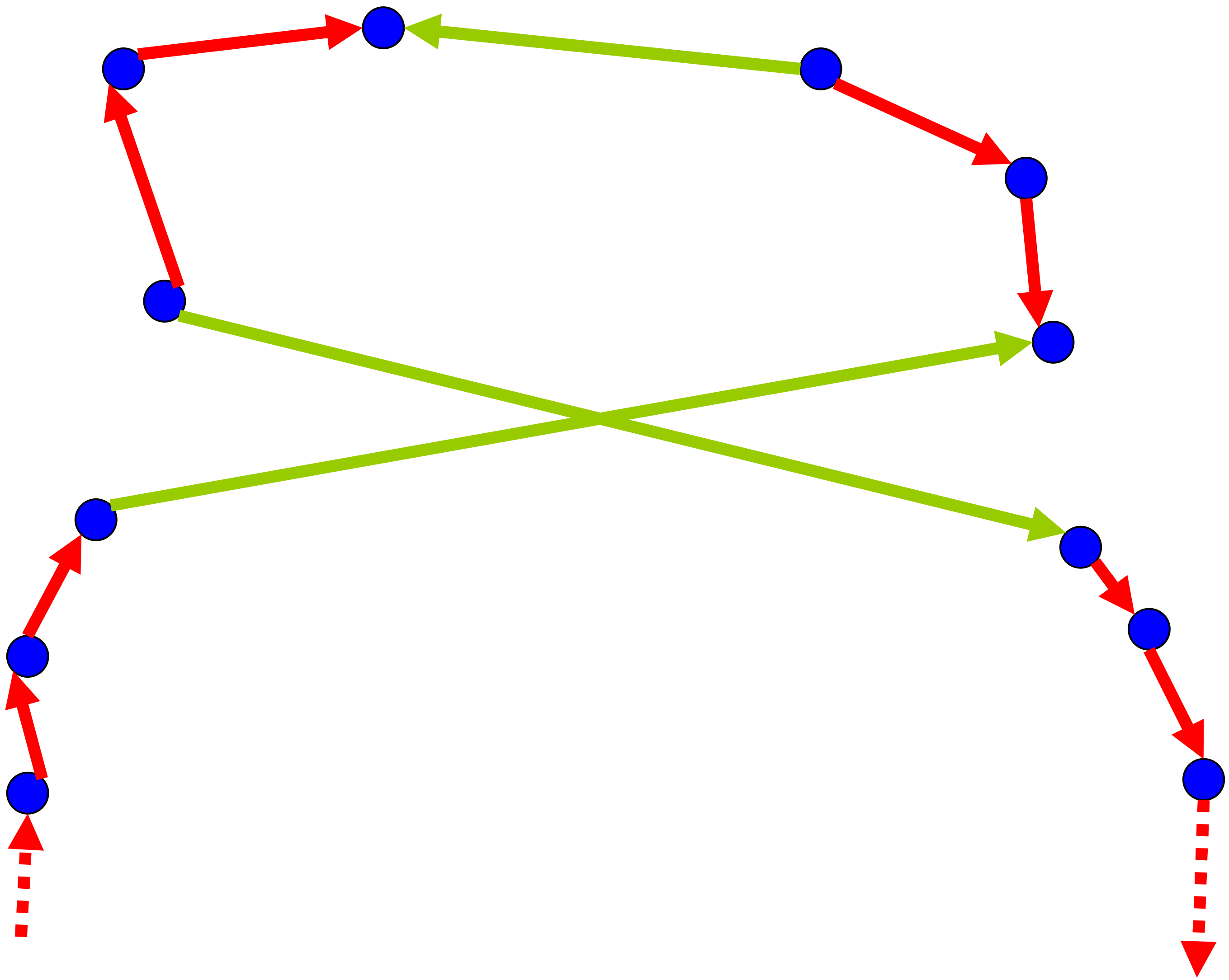
3-opt exchange



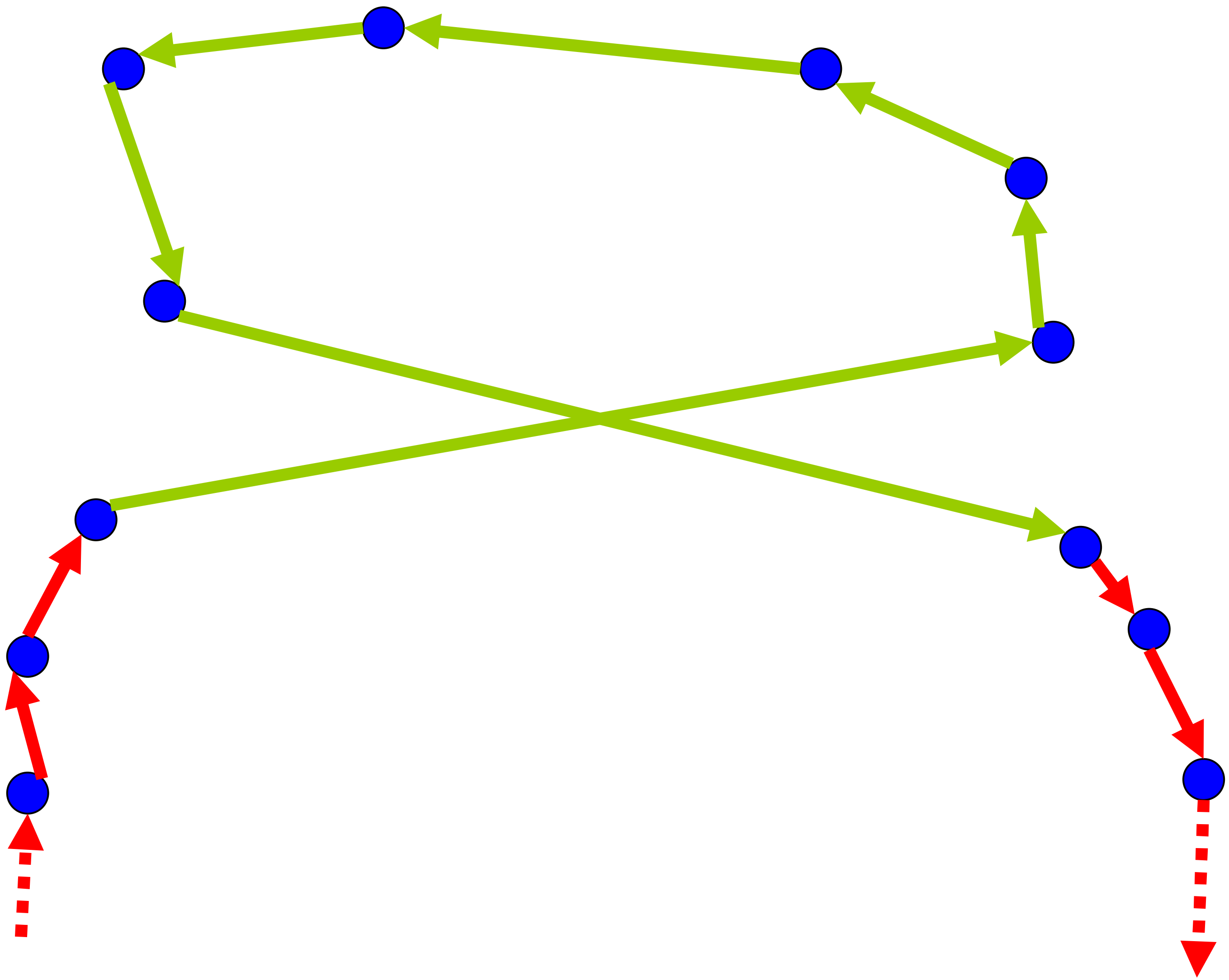
3-opt exchange



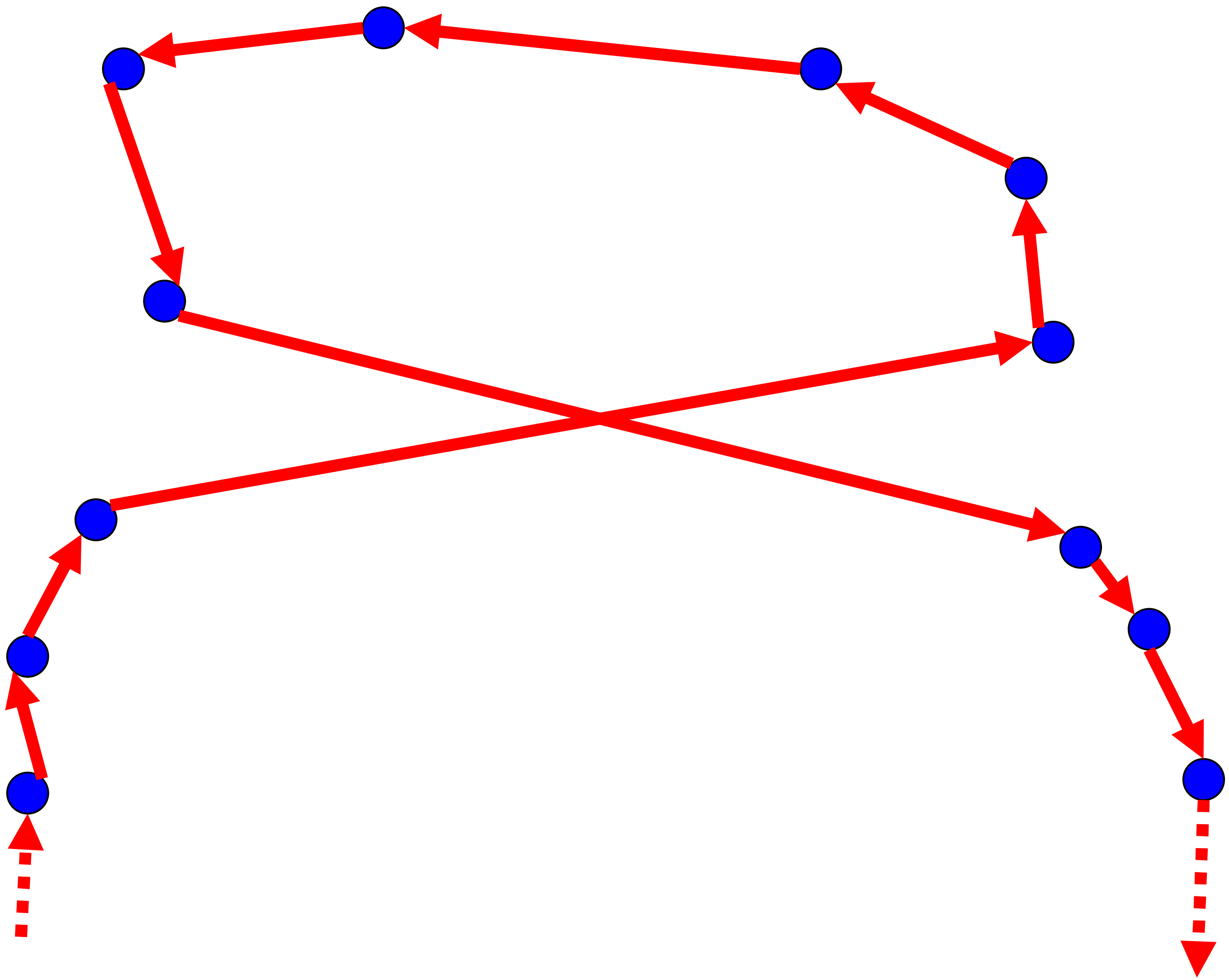
3-opt exchange



3-opt exchange



3-opt exchange



Neighbourhood

Strongly connected:

- Any solution can be reached from any other
  (e.g. 2-opt)

Weakly optimally connected

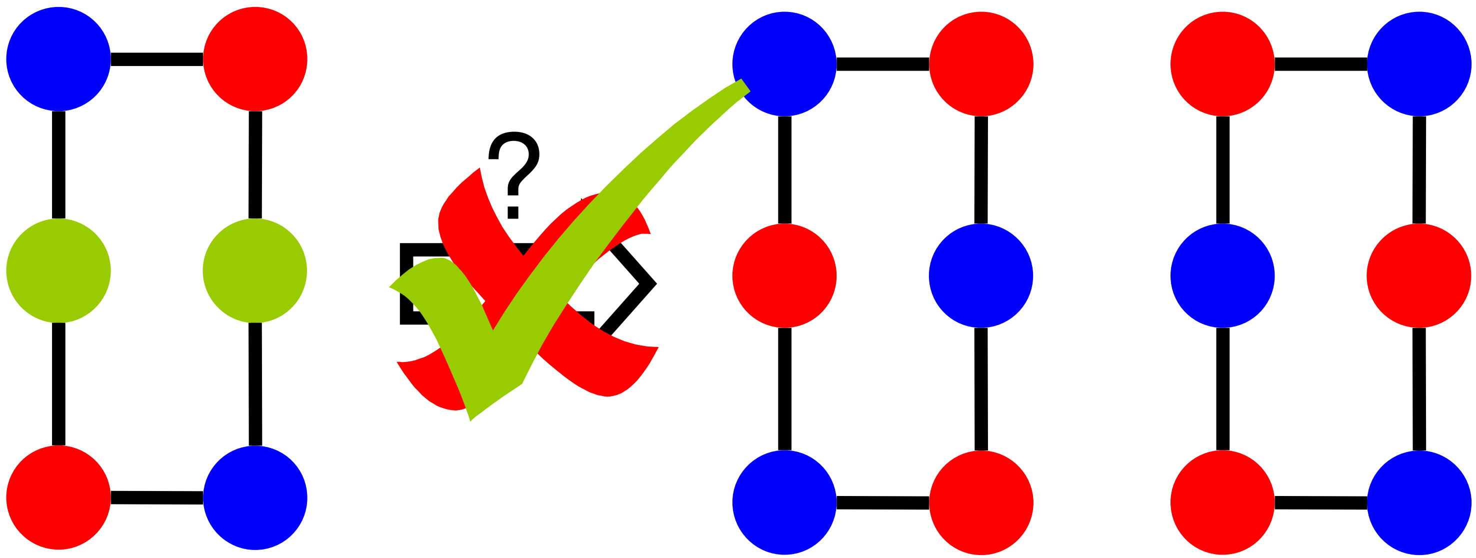
- The optimum can be reached from any starting solution

Neighbourhood

- Hard constraints create solution impenetrable mountain ranges
- Soft constraints allow passes through the mountains

- E.g. Map Colouring (*k*-colouring)
  - Colour a map (graph) so that no two adjacent countries (nodes) are the same colour
  - Use at most *k* colours
  - Minimize number of colours

Map Colouring



Starting sol

Two optimal solutions

Define neighbourhood as:

Change the colour of at most one vertex

Make k-colour constraint soft…

Iterative Improvement

1. Find initial (incumbent) solution S
2. Find $T \in \mathcal{N}(S)$ which minimises objective
3. If $z(T) \geq z(S)$

   Stop

   Else

   S = T

   Goto 2

Iterative Improvement

- Best First (a.k.a Greedy Hill-climbing, Discrete Gradient Ascent)
  - Requires entire neighbourhood to be evaluated
  - Often uses randomness to split ties
- First Found
  - Randomise neighbourhood exploration
  - Implement first improving change

Local Minimum

- Iterative improvement will stop at a local minimum
- Local minimum is not necessarily a global minimum

How do you escape a local minimum?

Restart

- Find initial solution using (random) procedure
- Perform Iterative Improvement
- Repeat, saving best

- Remarkably effective
- Used in conjunction with many other meta-heuristics

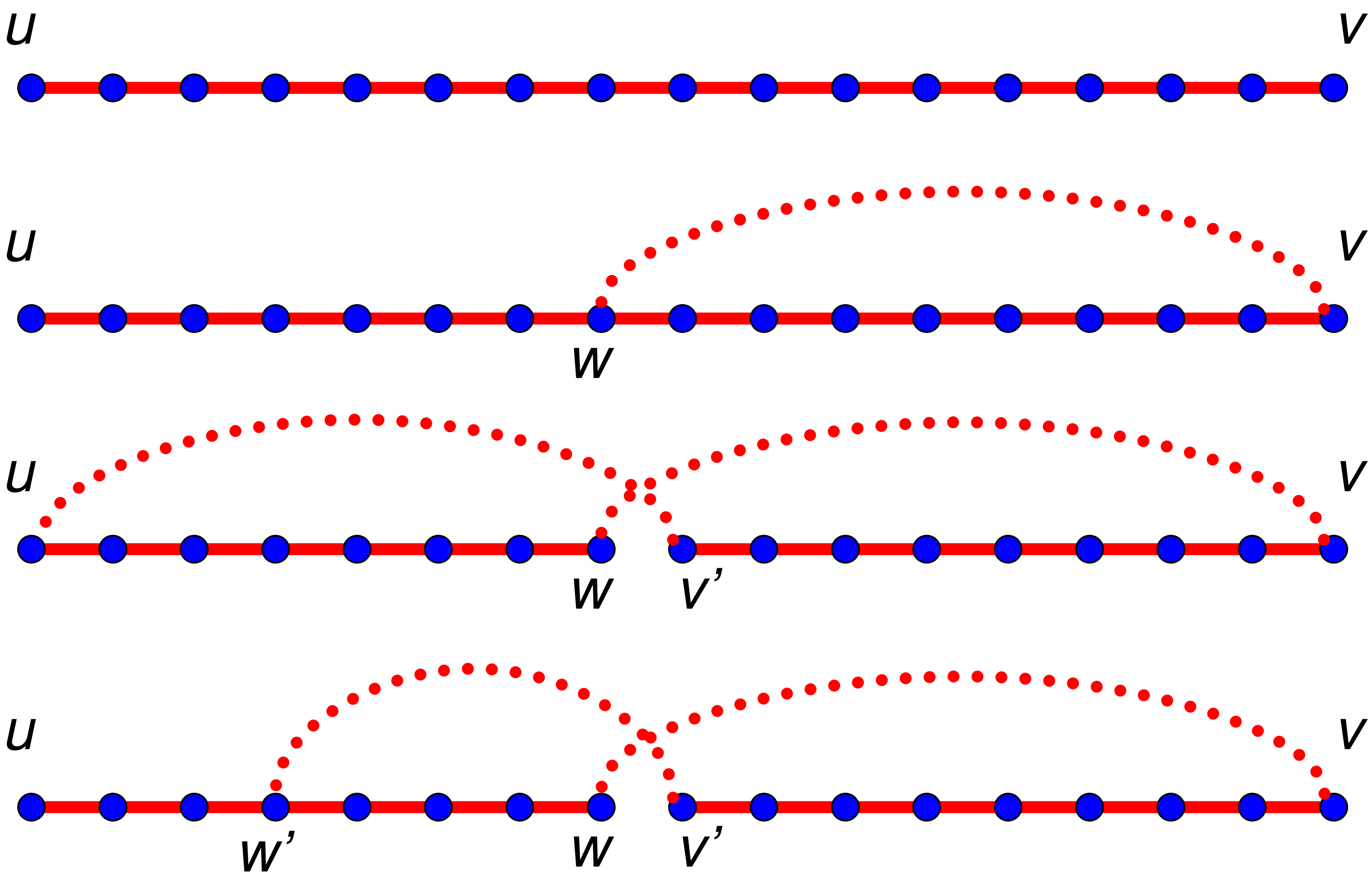
- Results from SAT

Variable Depth Search

- Make a series of moves
- Not all moves are cost-decreasing
- Ensure that a move does not reverse previous move

- Very successful VDS: Lin-Kernighan algorithm for TSP (1973)
(Originally proposed for Graph Partitioning Problem (1970))

Lin-Kernighan (1973) – δ-path



Lin-Kernighan (1973)

- Essentially a series of 2-opt style moves

- Find best δ-path
- Partially implement the path
- Repeat until no more paths can be constructed
- If arc has been added (deleted) it cannot be deleted (added)
- Implement best if cost is less than original

Dynasearch

- Requires all changes to be independent
- Requires objective change to be cummulative

- e.g. A set of 2-opt changes were no two swaps touched the same section of tour

- Finds best combination of exchanges
  – Exponential in worst case

Variable Neighbourhood Search

- Large Neighbourhoods are expensive
- Small neighbourhoods are less effective

Only search larger neighbourhood when smaller is exhausted

Variable Neighbourhood Search

- *m* Neighbourhoods N_i
- $|N_1| < |N_2| < |N_3| < \ldots < |N_m|$

1. Find initial sol *S* ; best = z (*S*)
2. k = 1;
3. Search $N_k(S)$ to find best sol *T*
4. If $z(T) < z(S)$

   S = T

   k = 1

   else

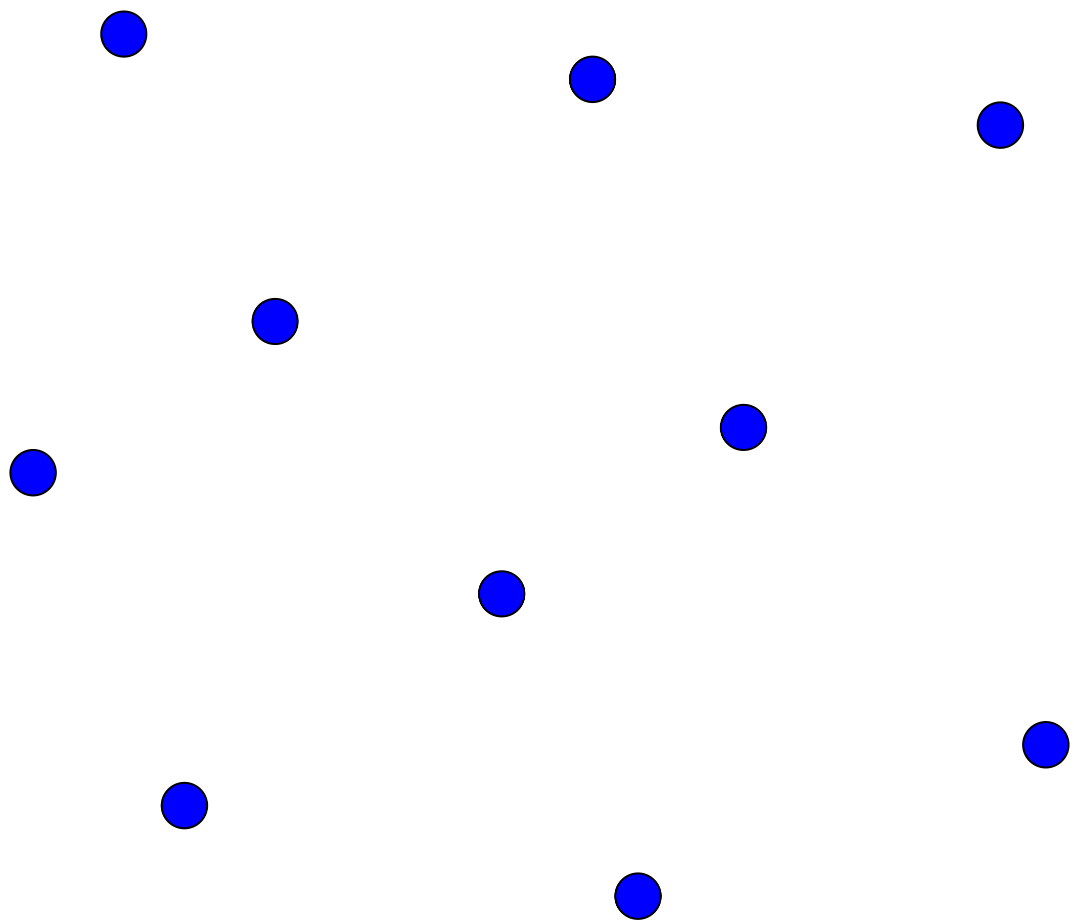
   k = k+1

Large Neighbourhood Search

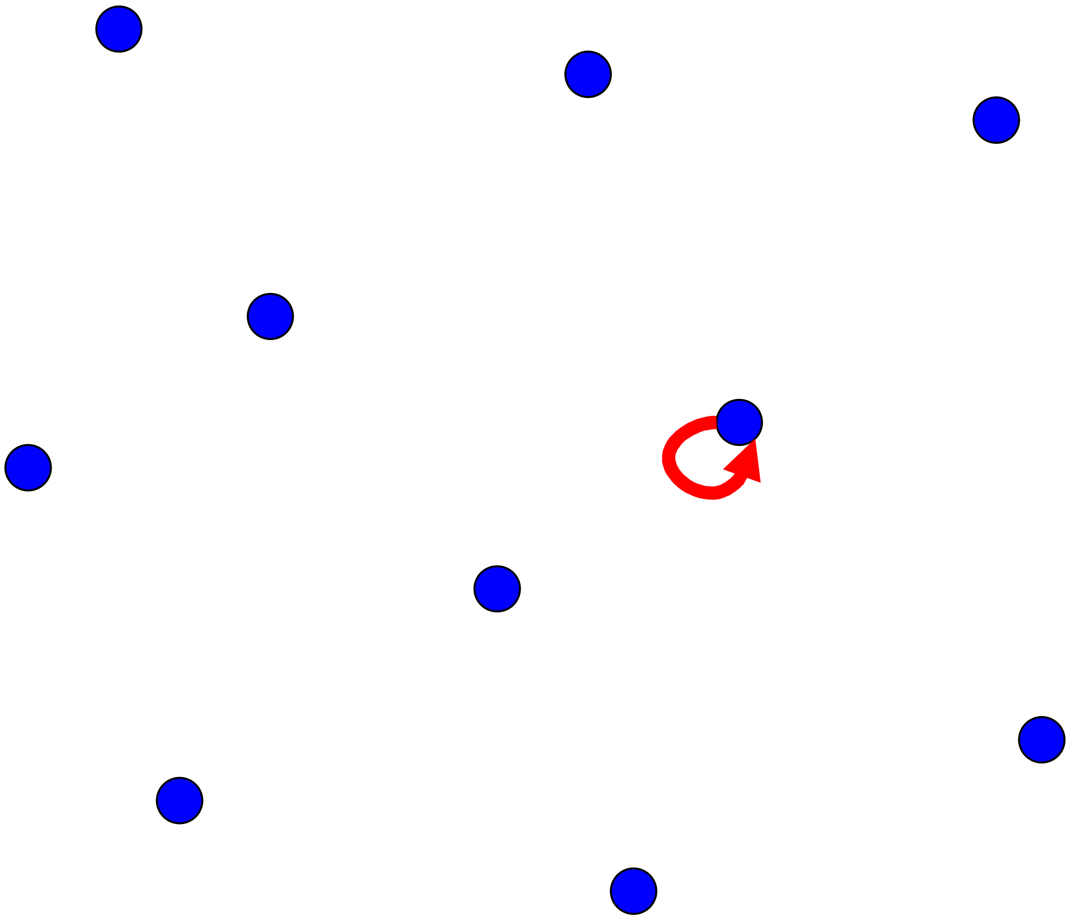
- Partial restart heuristic

1. Create initial solution
2. Remove a part of the solution
3. Complete the solution as per step 1
4. Repeat, saving best

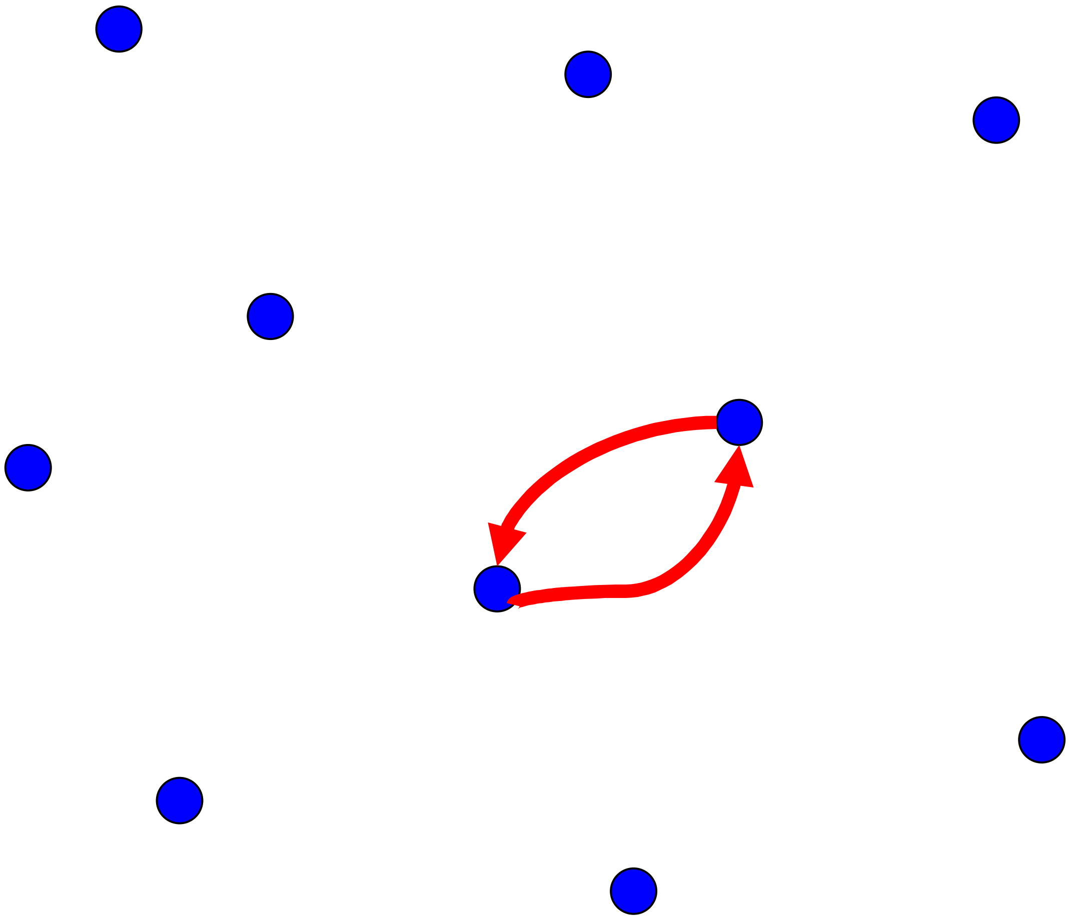
LNS – Construct



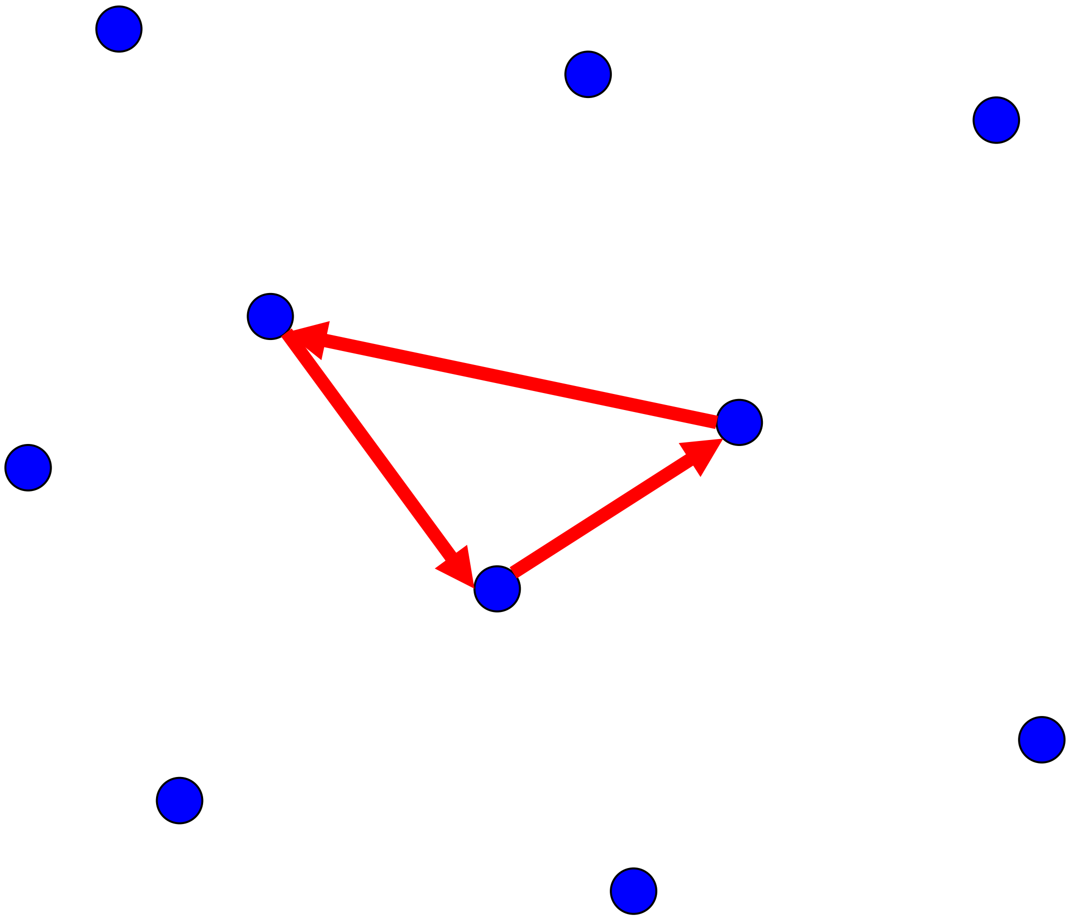
LNS – Construct



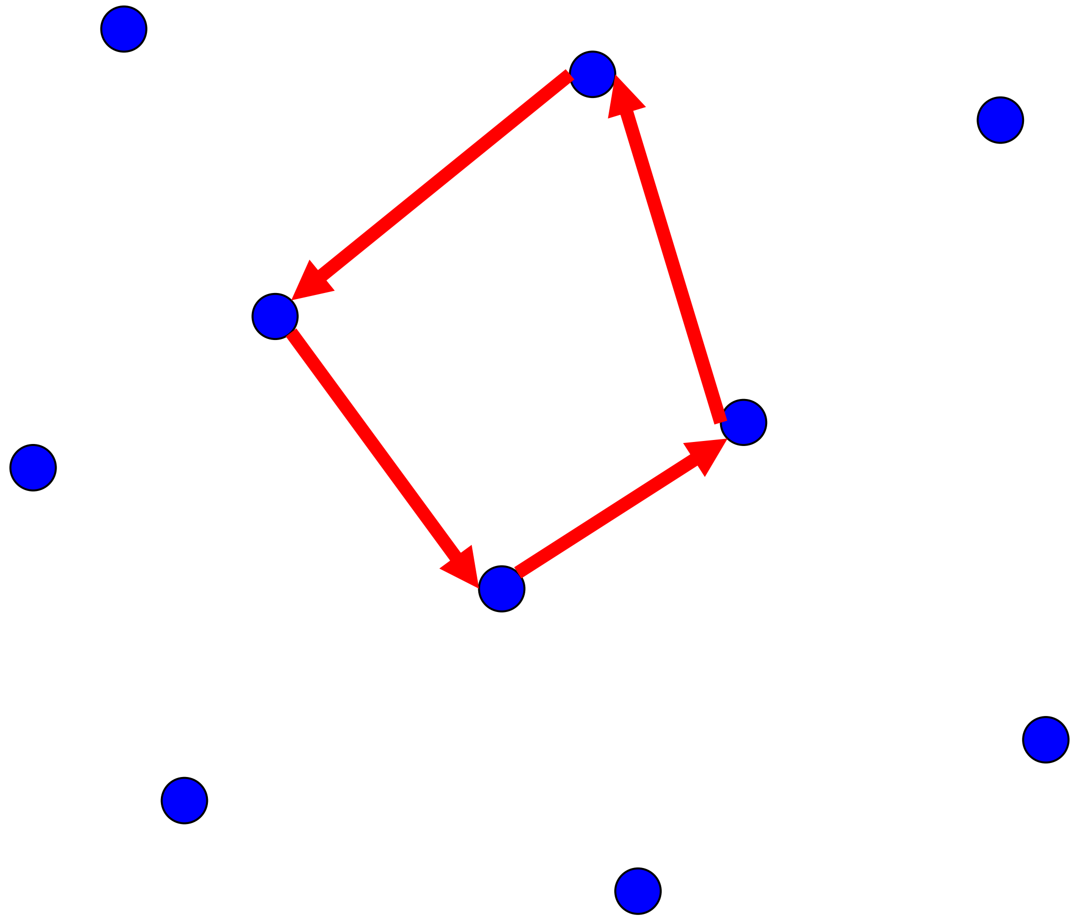
LNS – Construct



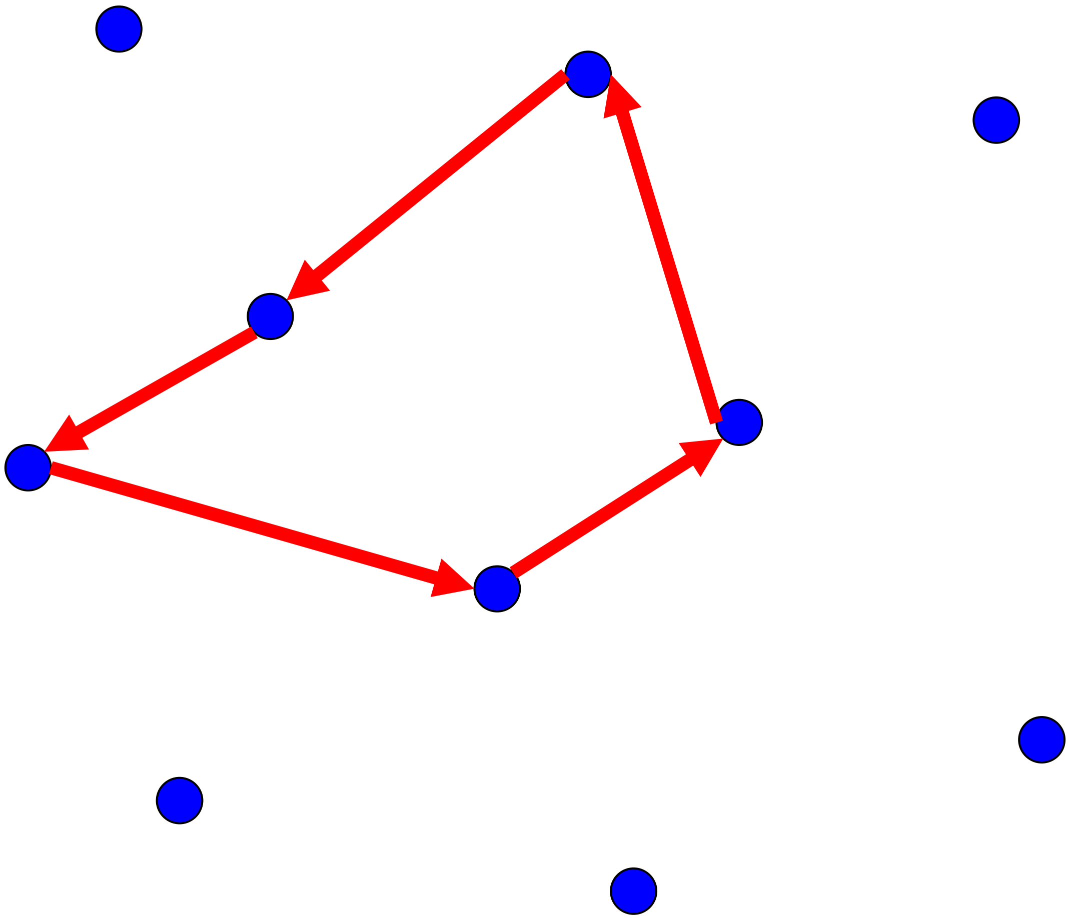
LNS – Construct



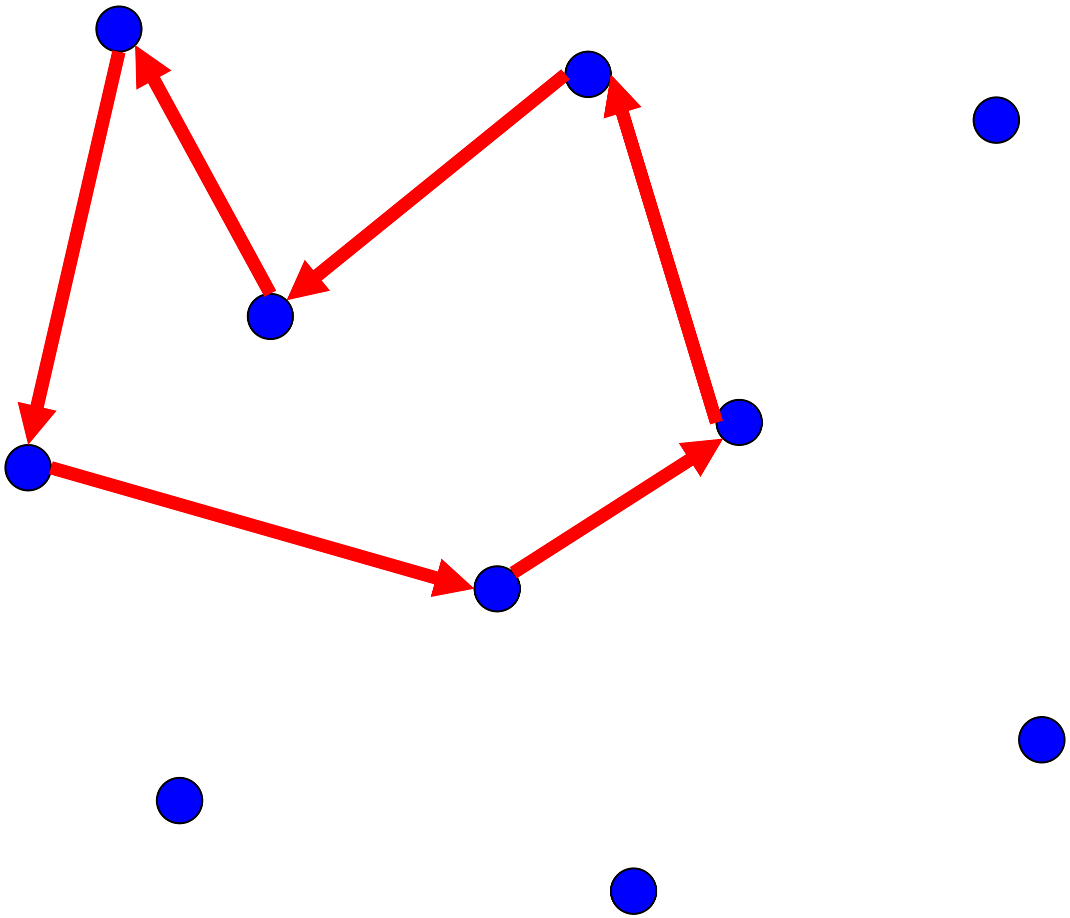
LNS – Construct



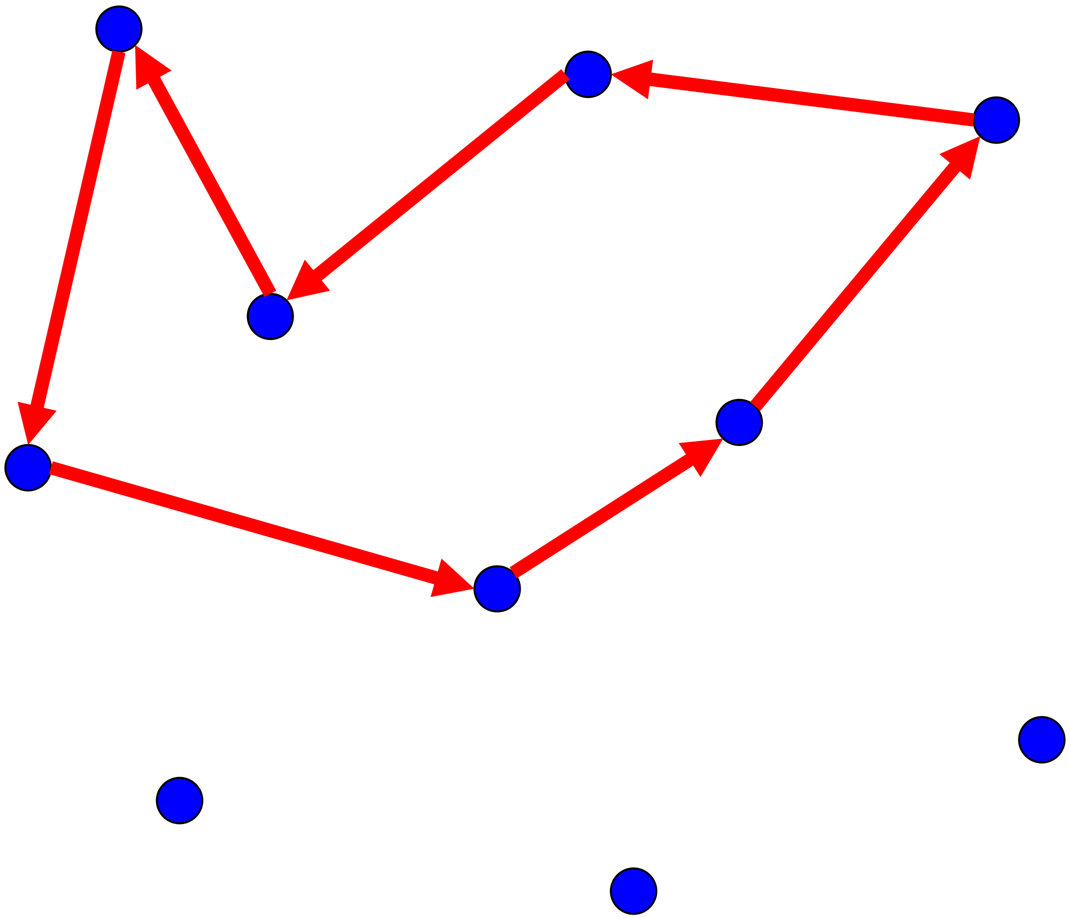
LNS – Construct



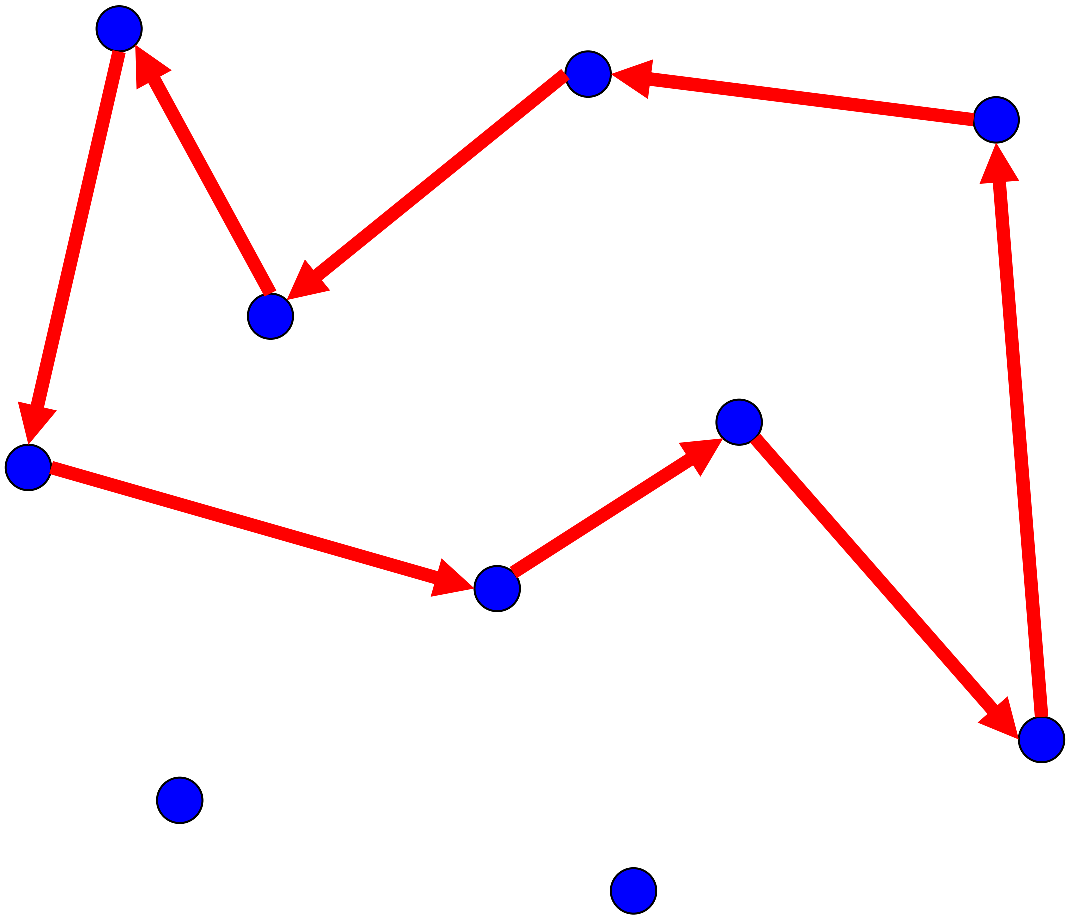
LNS – Construct



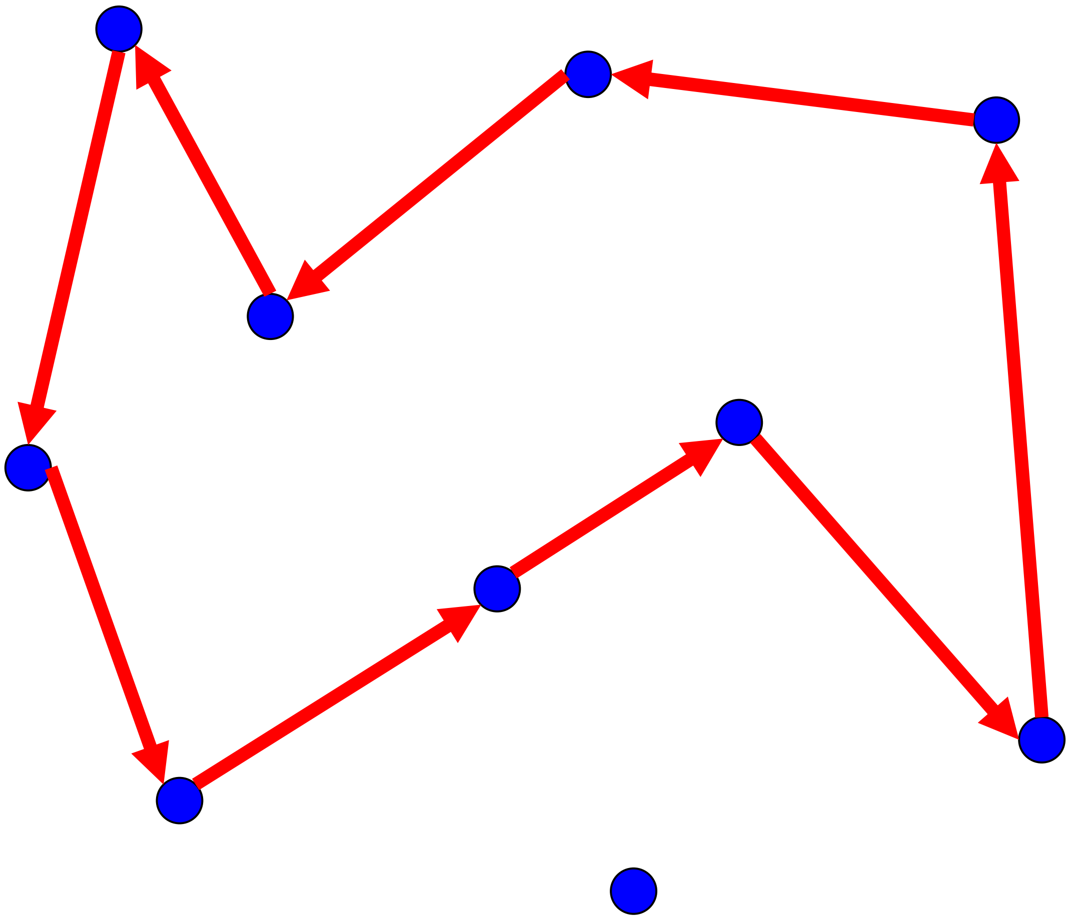
LNS – Construct



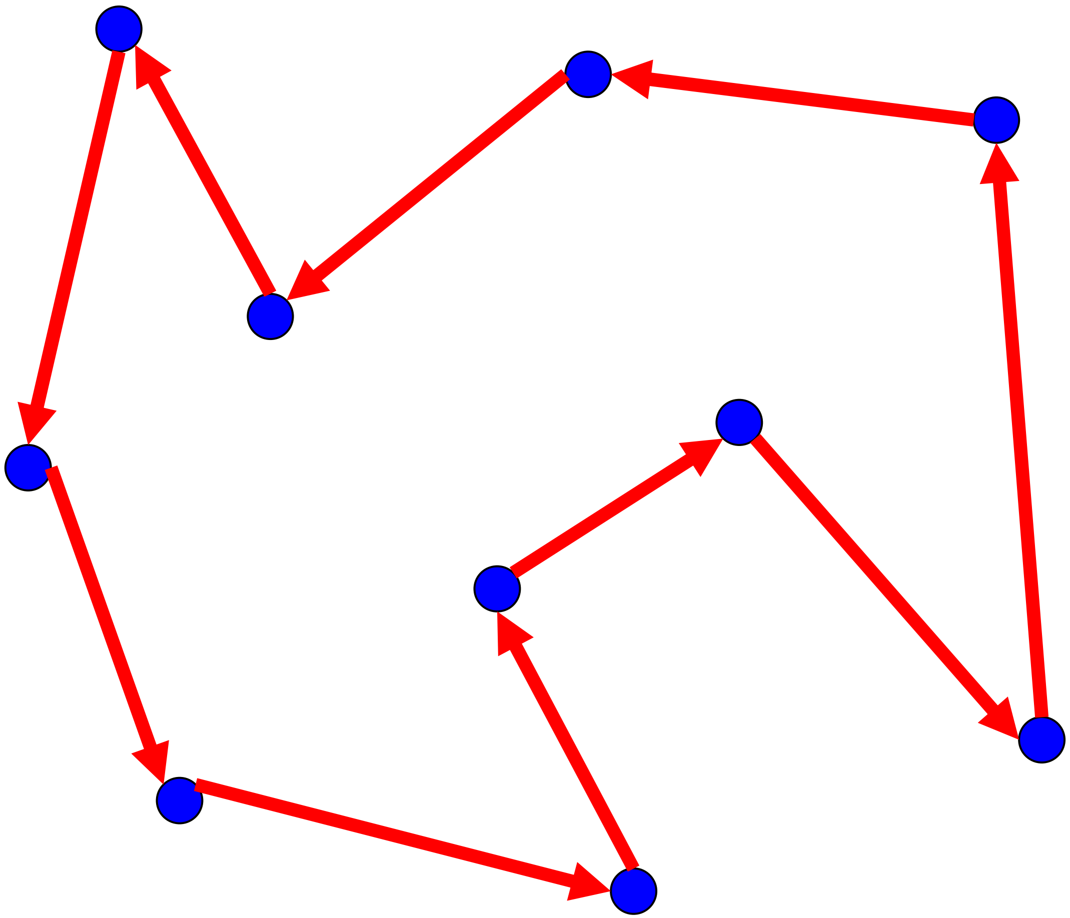
LNS – Construct



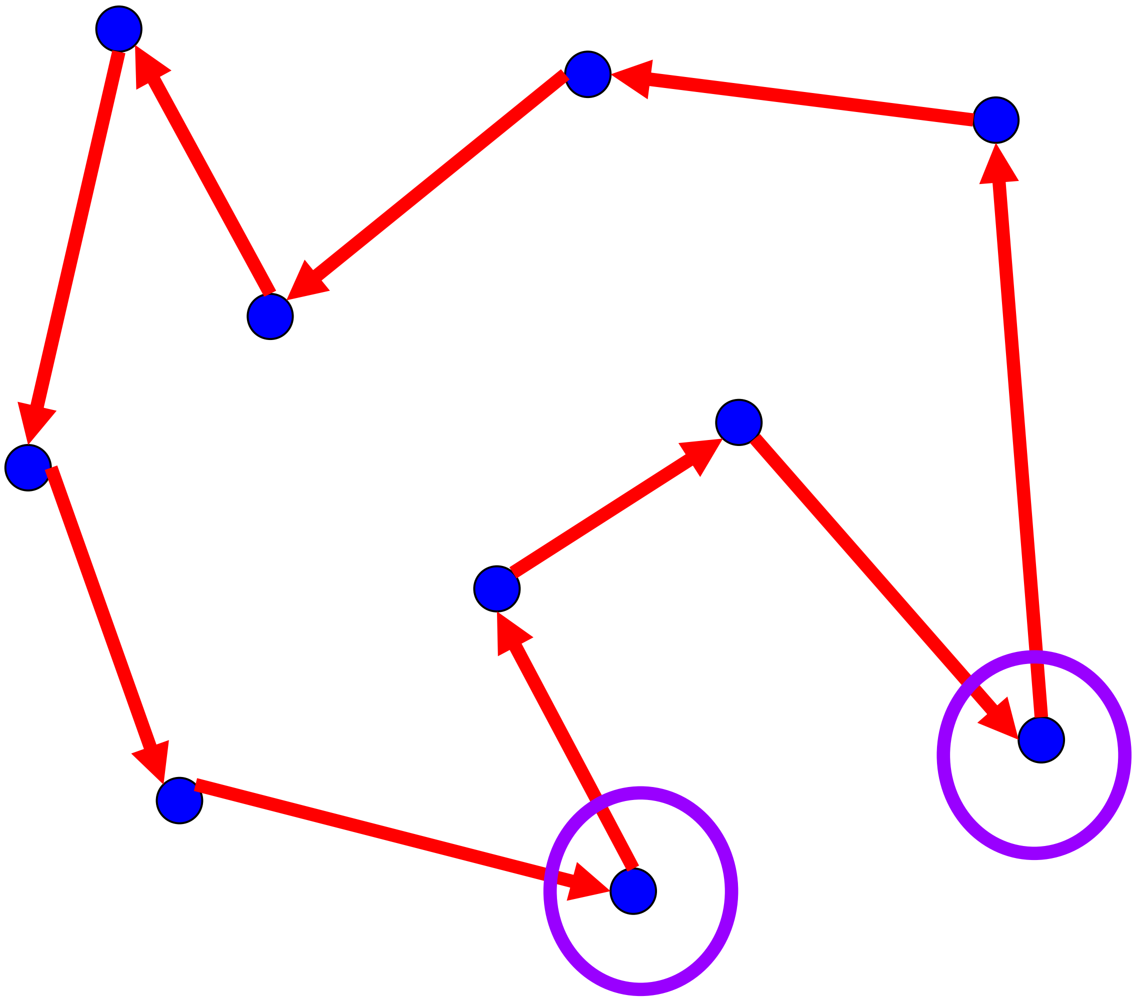
LNS – Construct



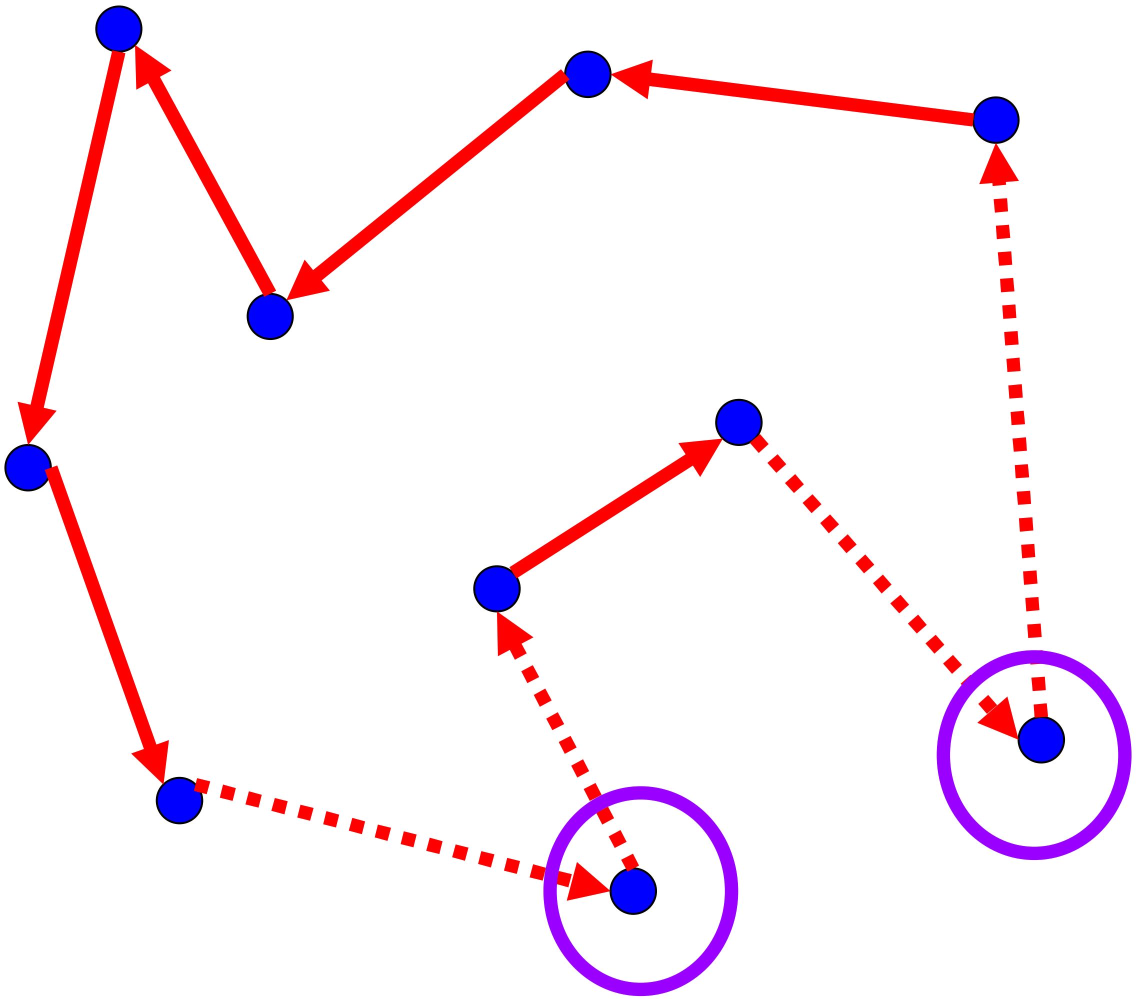
LNS – Construct



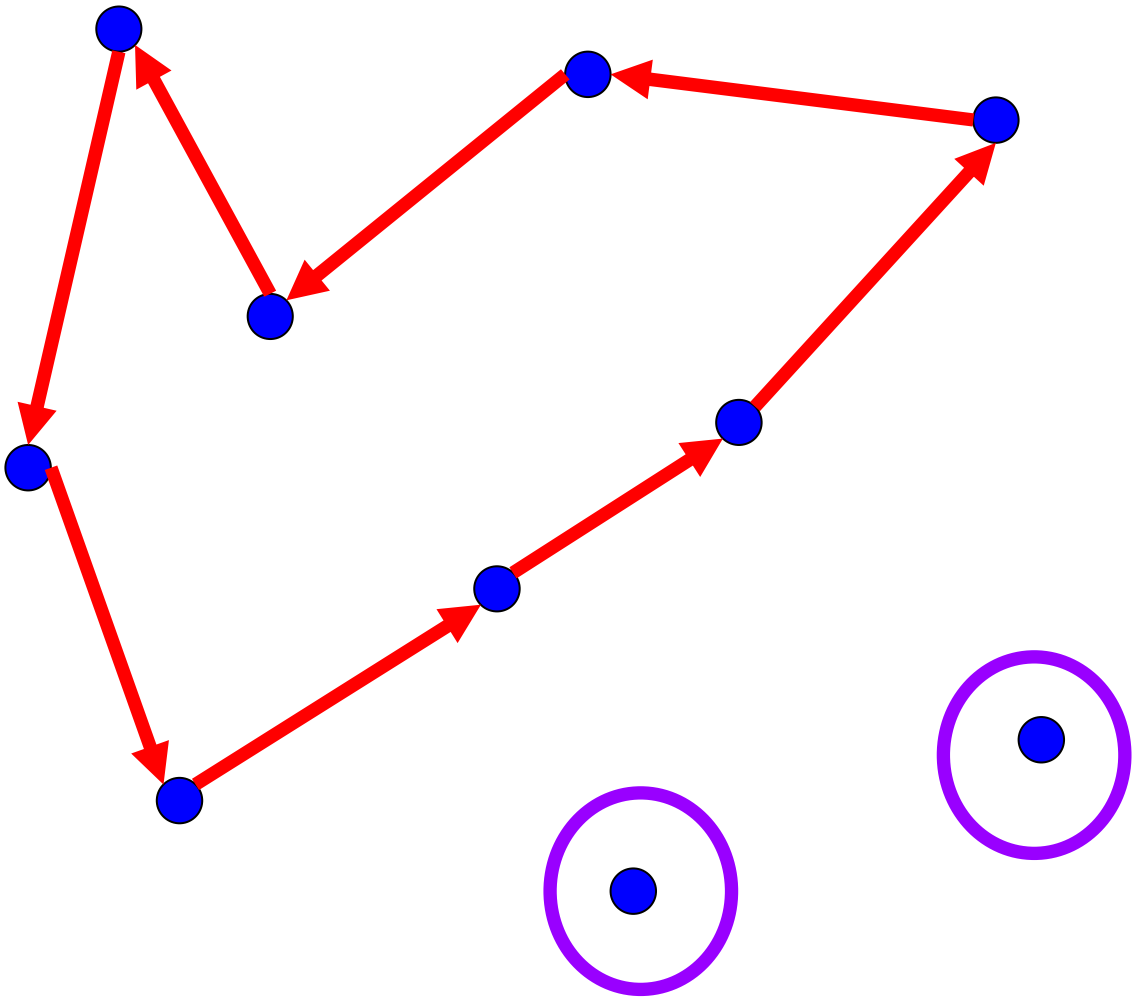
LNS – Destroy



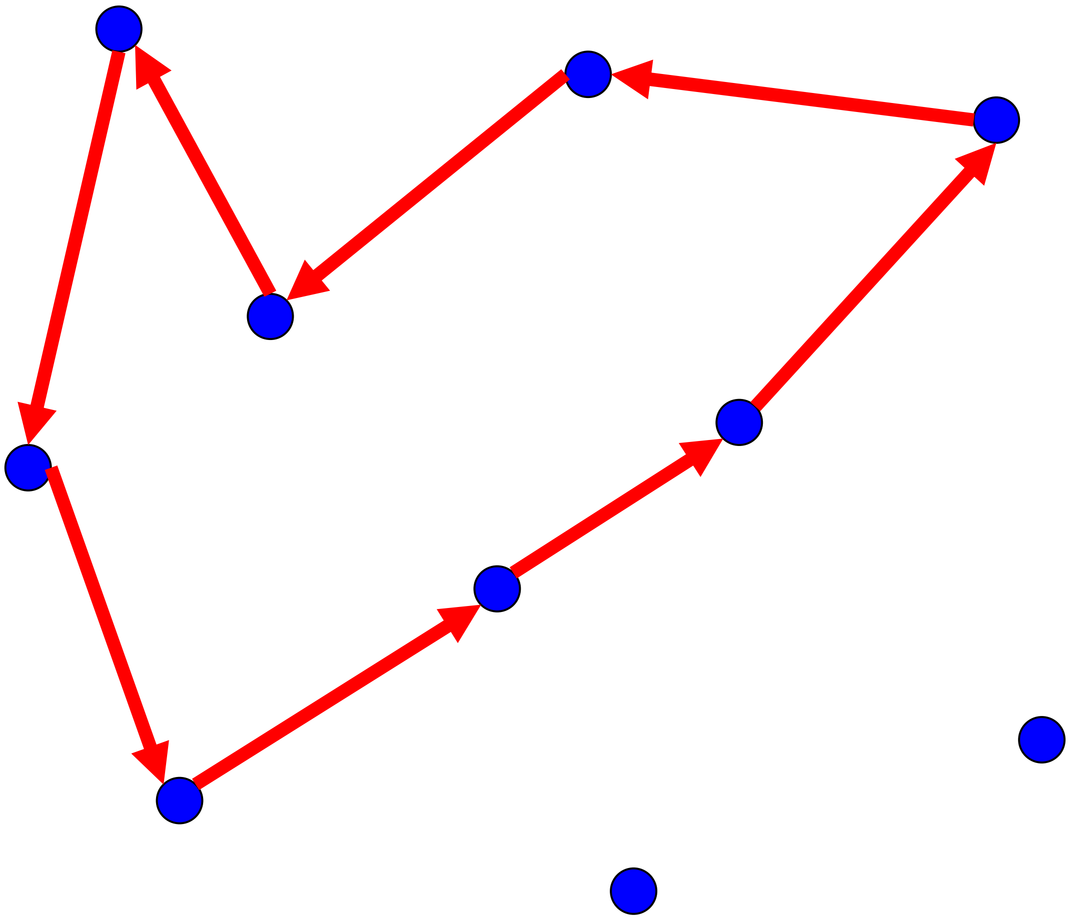
LNS – Destroy



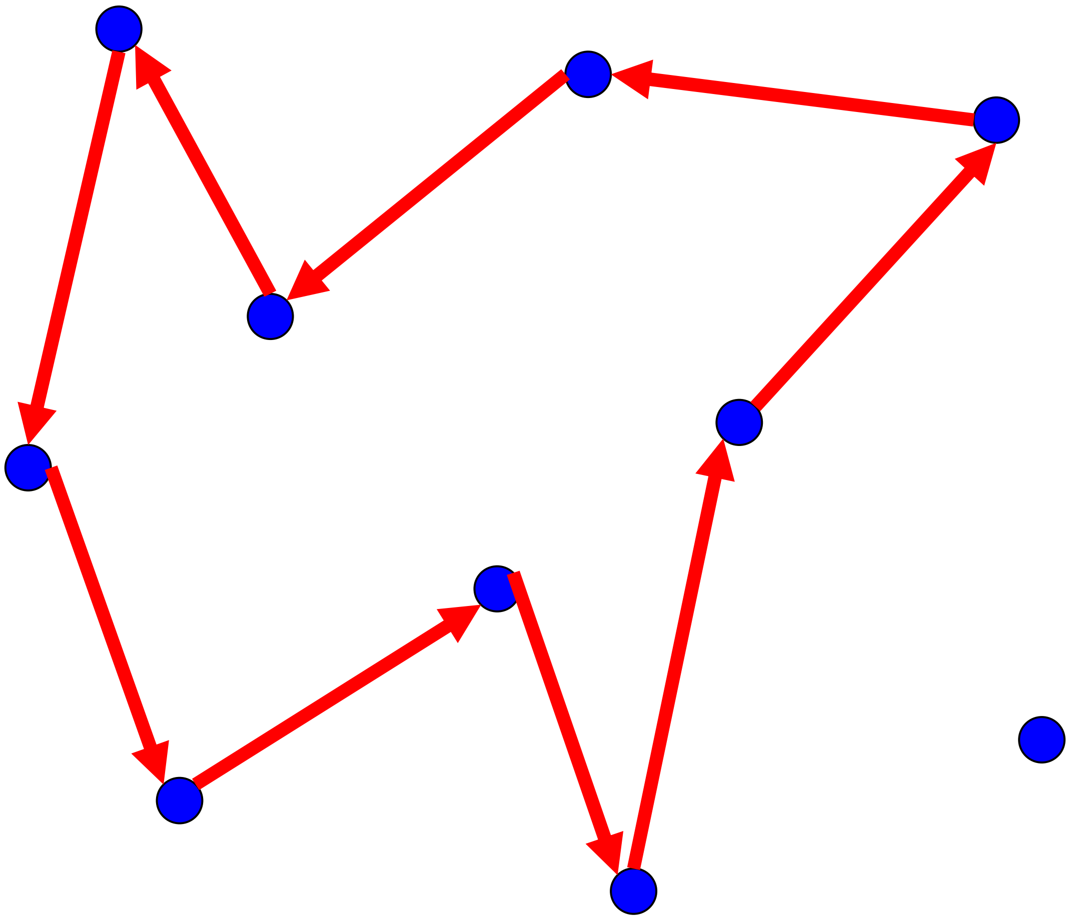
LNS – Destroy



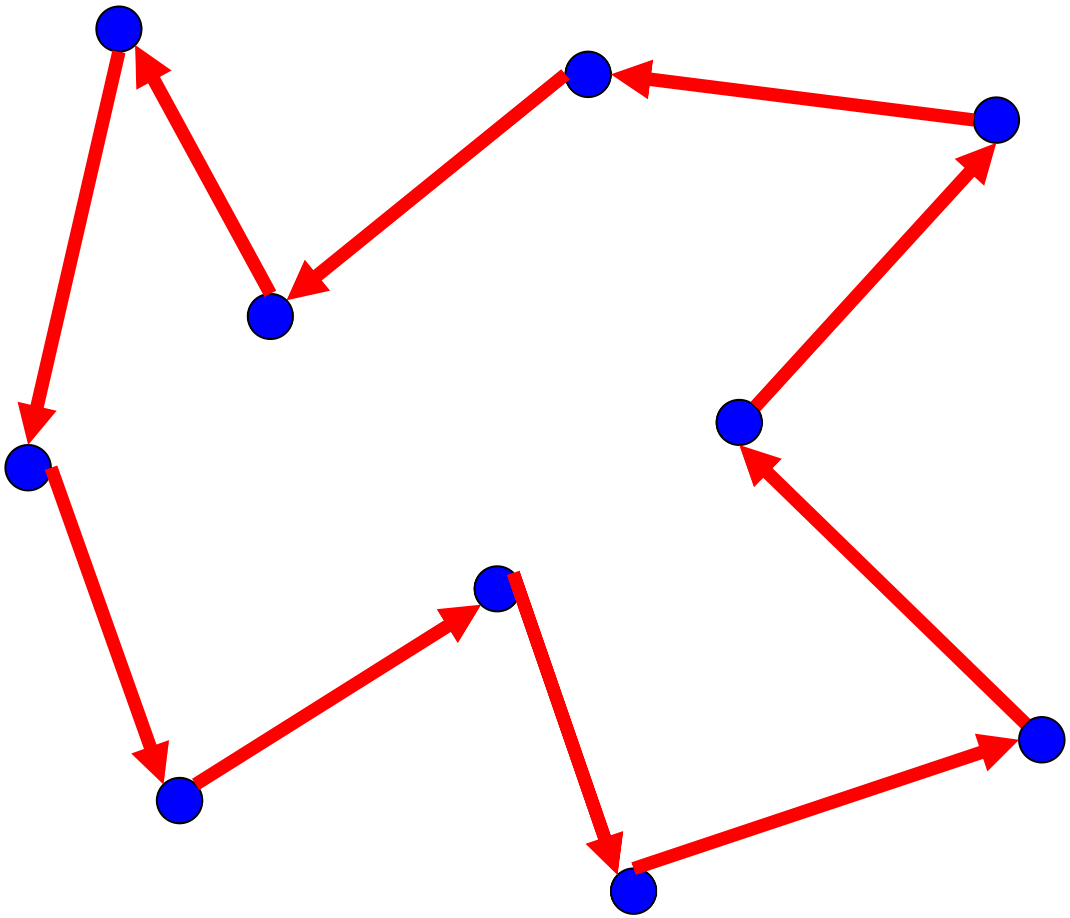
LNS – Destroy



LNS – Construct



LNS – Construct



LNS

- The magic is choosing which part of the solution to destroy
- Different problems (and different instances) need different heuristic

Speeding Up 2/3-opt

- For each node, store *k* nearest neighbours
- Only link nodes if they appear on list

- *k* = 20 does not hurt performance much
- *k* = 40 0.2% better
- *k* = 80 was *worse*

- FD-trees to help initialise

Advanced Stochastic Local Search

- Simulated Annealing
- Tabu Search
- Genetic algorithms
- Ant Colony optimization

Simulated Annealing

- Kirkpatrick, Gelatt & Vecchi [1983]

- Always accept improvement in obj
- Sometimes accept increase in obj

$$P(\text{accept increase of } \Delta) = e^{\Delta/T}$$

- *T* is temperature of system
- Update *T* according to *"cooling schedule"*
- *(T = 0) ==* Greedy Iterative Improvement

Simulated Annealing

- Nice theoretical result:

  As number of iters → ∞, probability of finding the optimal solution → 1

- Experimental confirmation: On many problem, long runs yield good results
- Weak optimal connection required

Simulated Annealing

1. Generate initial S
2. Generate random $T \in \mathcal{N}(S)$
3. $\Delta = z(T) - z(S)$
4. if $\Delta < 0$

   S = T ; goto 2
5. if rand() < $e^{\Delta/T}$

   S = T ; goto 2

Simulated Annealing

Initial T

- Set equal to max [acceptable] Δ

Updating T

- Geometric update: $T_{k+1} = \alpha \, T_k$
- α usually in [0.9, 0.999]

Don't want too many changes at one temperature (too hot):

If (*numChangesThisT* > *maxChangesThisT*)
    updateT()

Simulated Annealing

Updating T

- Many other update schemes
- Sophisticated ones look at mean, std-dev of Δ

Re-boil (== Restart)

- Re-initialise T

0-cost changes

- Handle randomly

Adaptive parameters

- If you keep falling into the same local minimum,
  *maxChangesThisT *= 2, or*
  *initialT *= 2*

Tabu Search

- Glover [1986]
- Some similarities with VDS
- Allow cost-increasing moves
- Selects best move in neighbourhood
- Ensure that solutions don't cycle by making return to previous solution "tabu"
- Effectively a modified neighbourhood
- Maintains more memory than just best sol

Tabu Search

Theoretical result (also applies to SA):

- As $k \rightarrow \infty$ P(find yourself at an optimal sol) gets larger relative to other solutions

Tabu Search

Basic Tabu Search:

1. Generate initial solution S, S* = S
2. Find *best* $T \in \mathcal{N}(S)$
3. If $z(T) \geq z(S)$

   Add T to tabu list

4 S = T

5 if $z(S) < z(S^*)$ then S* = S

6 if stopping condition not met, goto 2

Tabu Search

Tabu List:

- List of solutions cannot be revisited

Tabu Tenure

- The length of time a solution remains tabu
- = length of tabu list

- Tabu tenure t ensures no cycle of length t

Tabu Search

Difficult/expensive to store whole solution

- Instead, store the "move" (delta between S and T)
- Make inverse move tabu
  - e.g. 2-opt adds 2 new arcs to solution
  - Make deletion of both(?) tabu

But

- Cycle of length *t* now possible
- Some non-repeated states tabu

Tabu Search

Tabu List:
- List of moves that cannot be undone

Tabu Tenure
- The length of time a move remains tabu

Stopping criteria
- No improvement for <param> iterations
- Others…

Tabu Search

- Diversification
  - Make sure whole solution space is sampled
  - Don't get trapped in small area
- Intensification
  - Search attractive areas well
- Aspiration Criteria
  - Ignore Tabu restrictions if very attractive (and can't cycle)
  - e.g.: $z(T) < best$

Tabu Search

- Diversification
  - Penalise solutions near observed local minima
  - Penalise solution features that appear often
  - Penalties can “fill the hole” near a local min

- Intensification
  - Reward solutions near observed local minima
  - Reward solution features that appear often

- Use $z'(S) = z(S)$ + penalties

Tabu Search – TSP

- TSP Diversification
  - Penalise (pred,succ) pairs seen in local minima
- TSP Intensification
  - Reward (pred,succ) pairs seen in local minima

- $z'(S) = z(S) + \Sigma_{ij}\ w_{ij}\text{count}(i,j)$
  - count(i,j): how many times have we seen (i,j)
  - w_{ij}: weight depending on diversify/intensify cycle

Adaptive Tabu Search

- If *t* (tenure) to small, we will return to the same local min
- Adaptively modify *t*
  - If we see the same local min, increase *t*
  - When we see evidence that local min escaped (e.g. improved sol), lower *t*
- … my current favourite

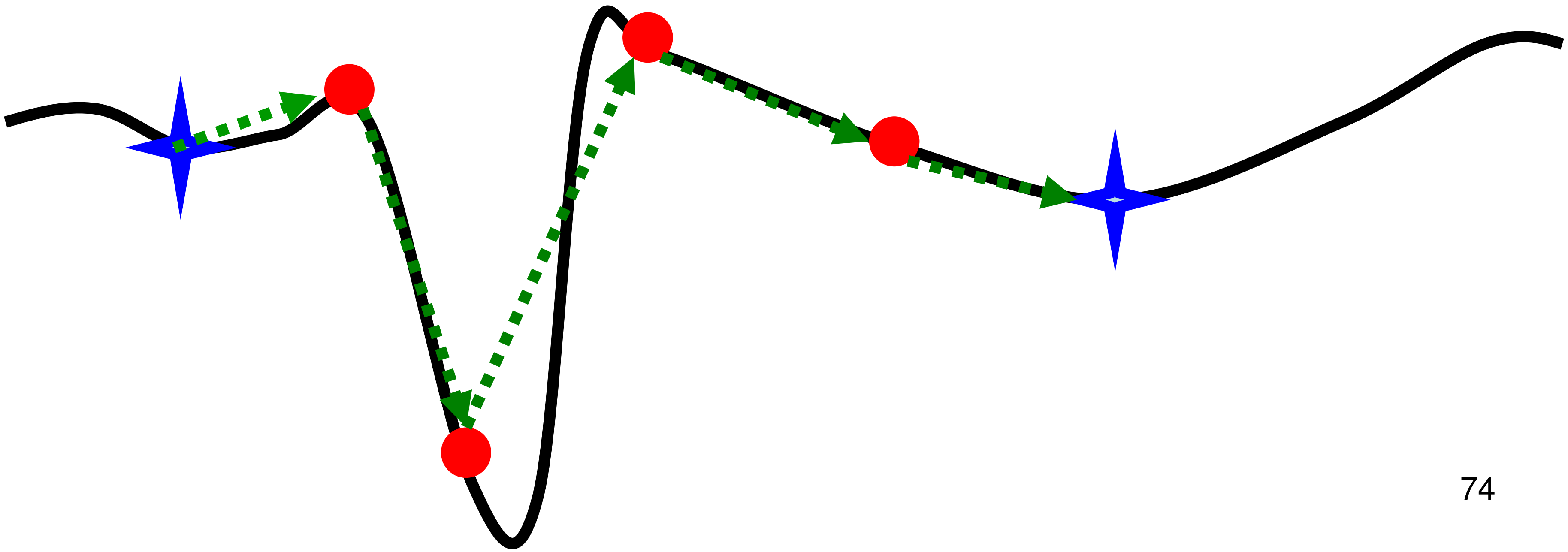
Tabu Search

1. Generate initial solution S ; S* = S
2. Generate $V^* \subseteq \mathcal{N}(S)$
   – Not tabu, or meets aspiration criterea
3. Find $T \in V^*$ which minimises z'
4. S = T
5. if $z(S) < z(S^*)$ then S* = S
6. Update tabu list, aspiration criterea, *t*
7. if stopping condition not met, goto 2

Path Relinking

Basic idea:

- Given 2 good solutions, perhaps a better solution lies somewhere in-between
- Try to combine “good features” from two solutions
- Gradually convert one solution to the other



Path Re-linking

TSP:

1	2	3	4	5	6
1	2	3	5	6	4
1	3	2	5	6	4
1	3	5	2	6	4
1	3	5	6	4	2

Genetic Algorithms

- Simulated Annealing and Tabu Search have a single “incumbent” solution (plus best-found)

- Genetic Algorithms are “population-based” heuristics – solution population maintained

Genetic Algorithms

- Problems are solved by an evolutionary process resulting in a best (fittest) solution (survivor).
- Evolutionary Computing
  - 1960s by I. Rechenberg
- Genetic Algorithms
  - Invented by John Holland 1975
  - Made popular by John Koza 1992

- Nature solves some pretty tough questions – let's use the same method

…begs the question… if homo sapien is the answer, what was the question??

Genetic Algorithms

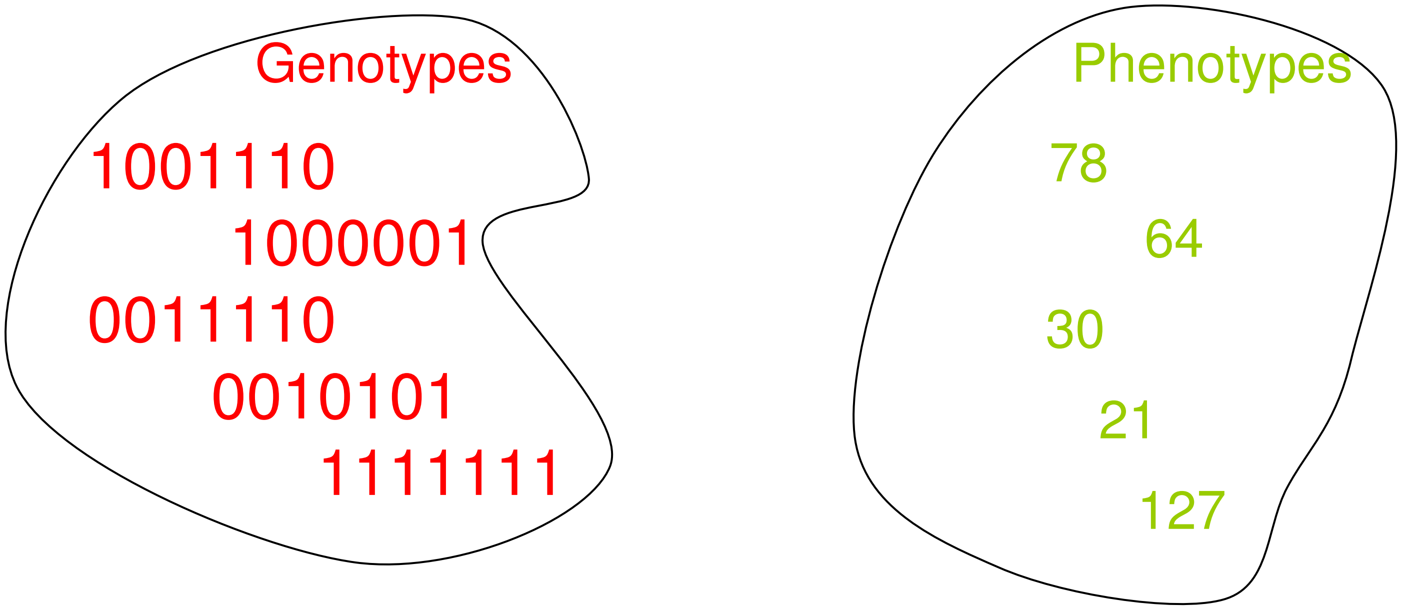
Vocabulary

- Gene – An encoding of a single part of the solution space (often binary)
- Genotype – Coding of a solution
- Phenotype – The corresponding solution
- Chromosome – A string of “Genes” that represents an individual – i.e. a solution.
- Population - The number of “Chromosomes” available to test

Vocabulary

Genotype: coded solutions
Phenotype: actual solutions

Measure fitness

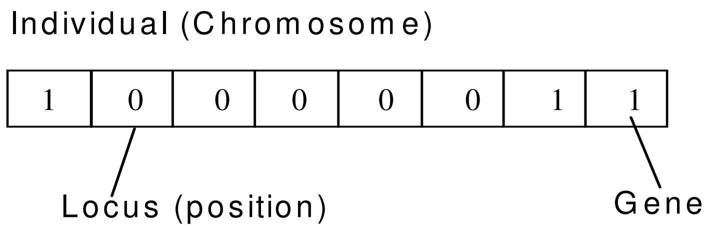


Search space

Solution space

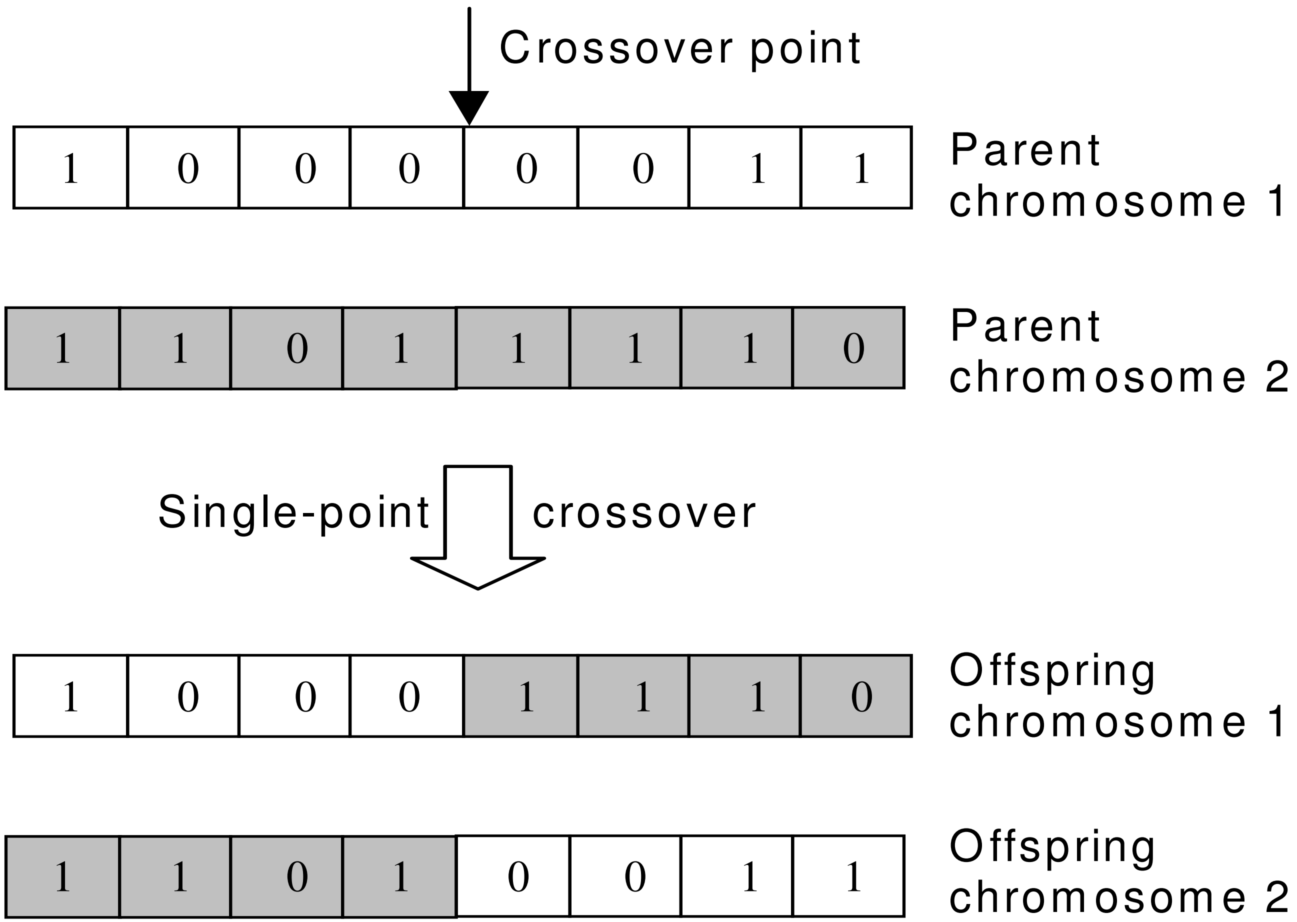
Note: in some evolutionary algorithms there is no clear distinction between genotype and phenotype

Vocabulary



Biology	*Computation*
Chromosome or individual	Bitstring that represents a candidate solution
Gene	A single bit (or a block of bits, in some cases)
Crossover	Random exchange of genetic material between chromosomes
Mutation	Random change of a certain bit in a chromosome
Genotype	Bit configuration of a chromosome
Phenotype	Solution decoded from a chromosome

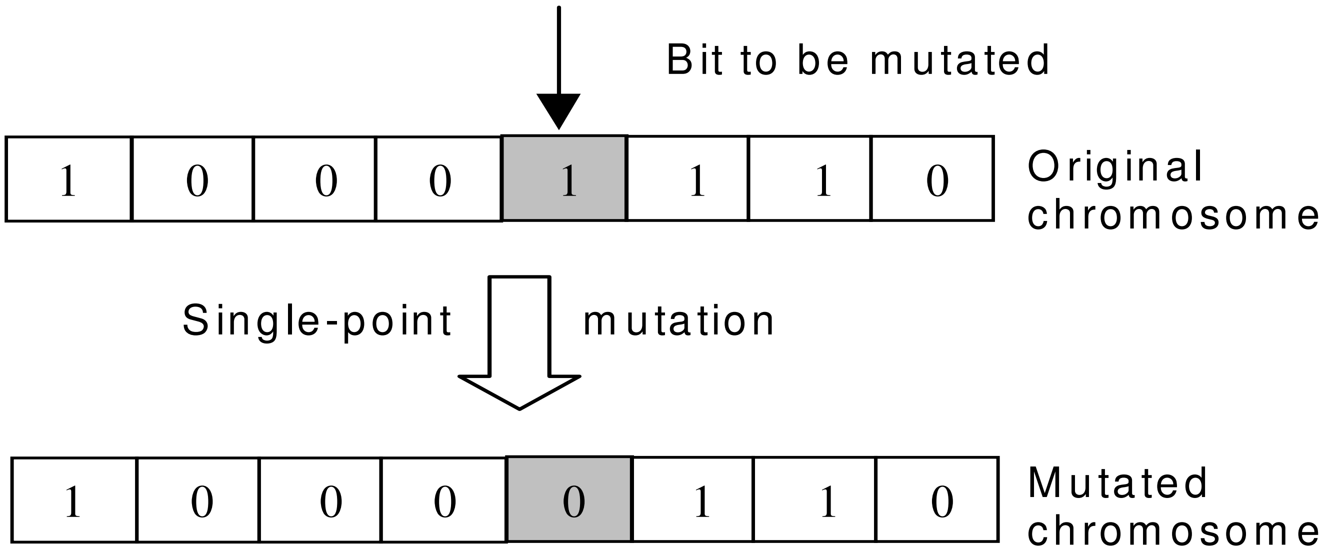
Crossover



Crossover point
1 0 0 0 0 0 1 1
Parent chromosome 1
1 1 0 1 1 1 1 0
Parent chromosome 2
Single-point crossover
1 0 0 0 1 1 1 0
Offspring chromosome 1
1 1 0 1 0 0 1 1
Offspring chromosome 2

Mutation

- Alter each gene independently with a prob p_m (mutation rate)
- $1/pop\_size < p_m < 1/ chromosome\_length$



Reproduction

- Chromosomes are selected to crossover and produce offspring
- Obey the law of Darwin:
  Best survive and create offspring.

- Roulette-wheel selection
- Tournament Selection
- Rank selection
- Steady state selection

Roulette Wheel Selection

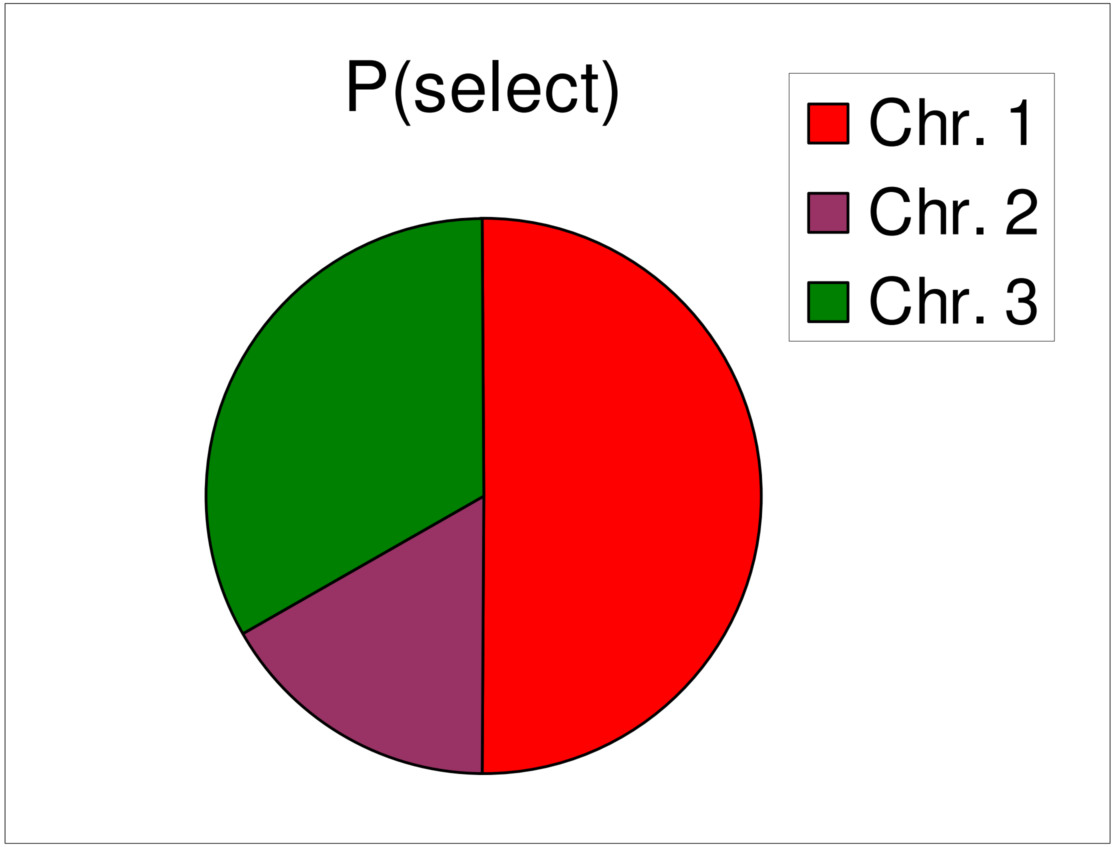
Main idea: better individuals get higher chance

– Chances proportional to fitness

– Assign to each individual a part of the roulette wheel

– Spin the wheel n times to select n individuals

	Fitness
Chr. 1	3
Chr. 2	1
Chr. 3	2



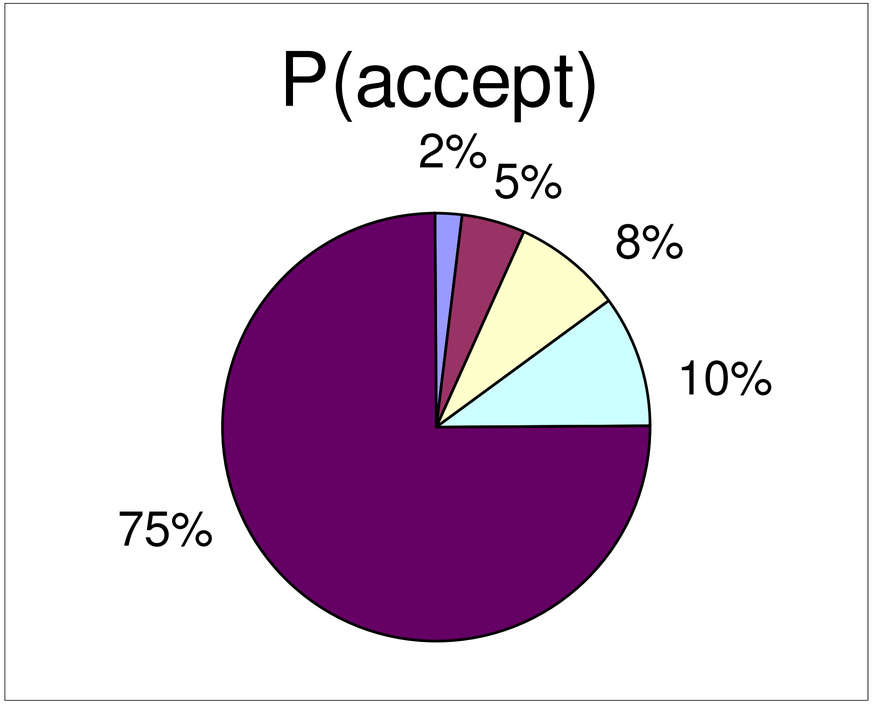
Tournament Selection

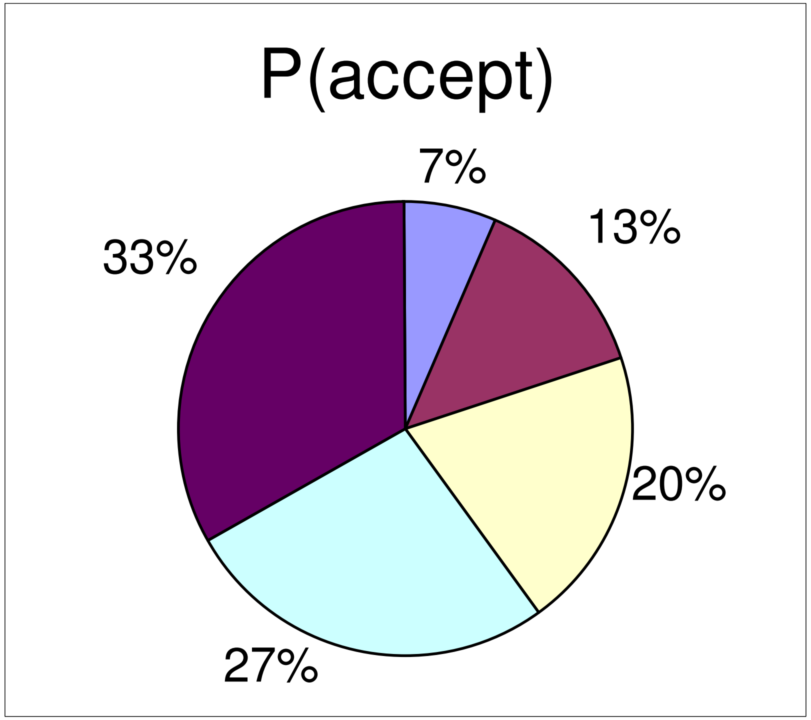
- Tournament competition among N individuals (N=2) are held at random.
- The highest fitness value is the winner.
- Tournament is repeated until the mating pool for generating new offspring is filled.

Rank Selection

- Roulette-wheel has problem when the fitness value differ greatly
- In rank selection the
  - worst value has fitness 1,
  - the next 2,......,
  - best has fitness N.

Rank Selection vs Roulette





Roulette Wheel

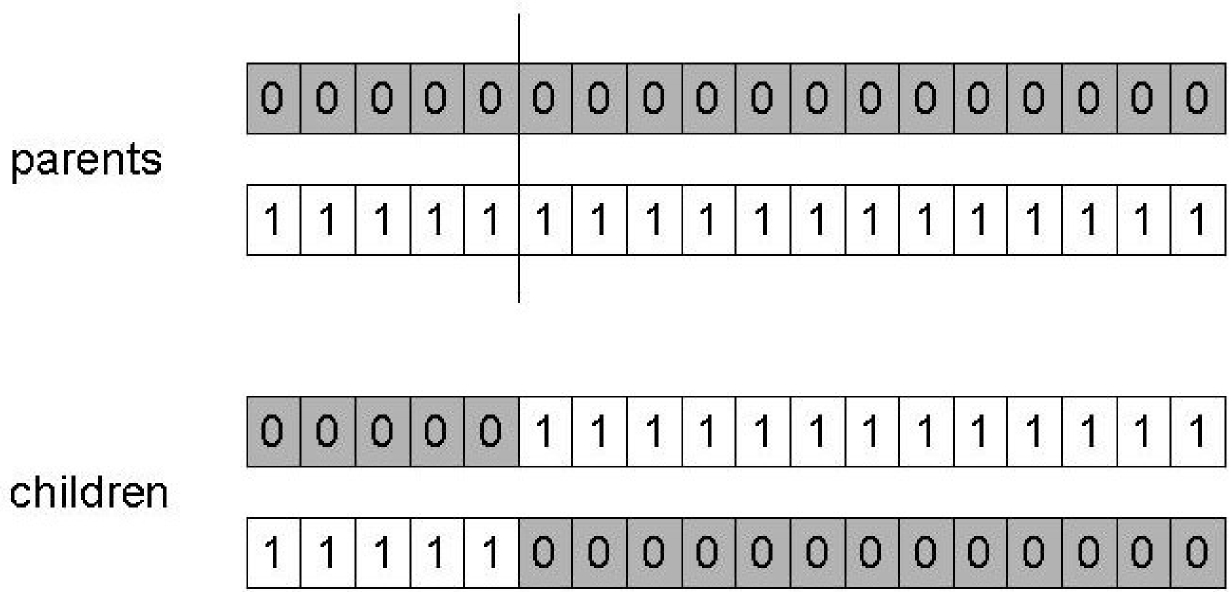
Rank

Crossover

- Single –site crossover
- Multi-point crossover
- Uniform crossover

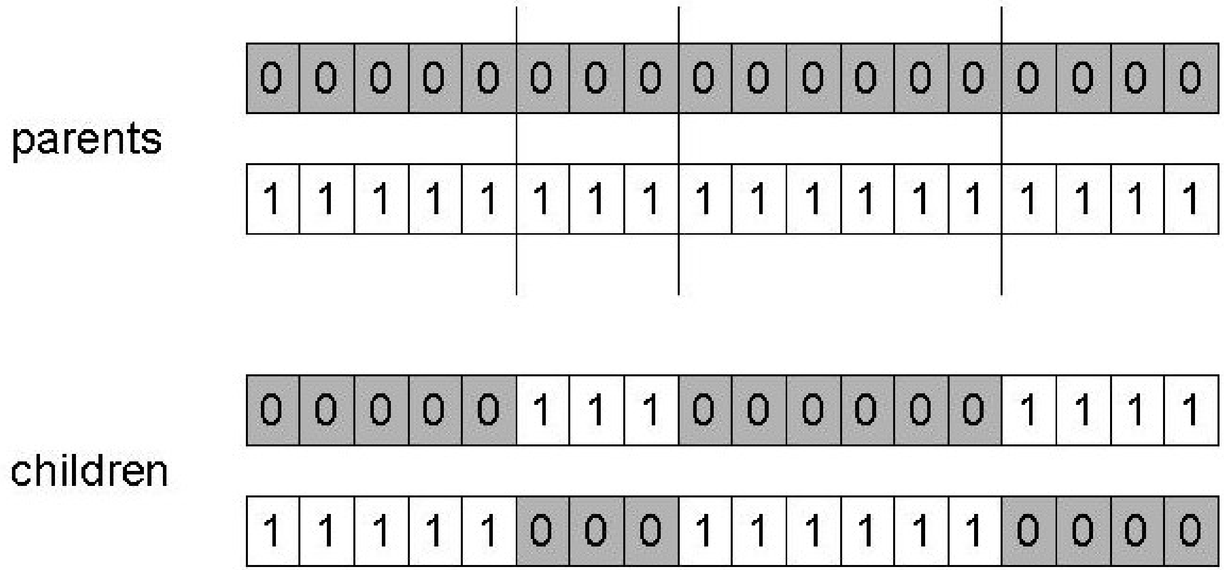
Single-site

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



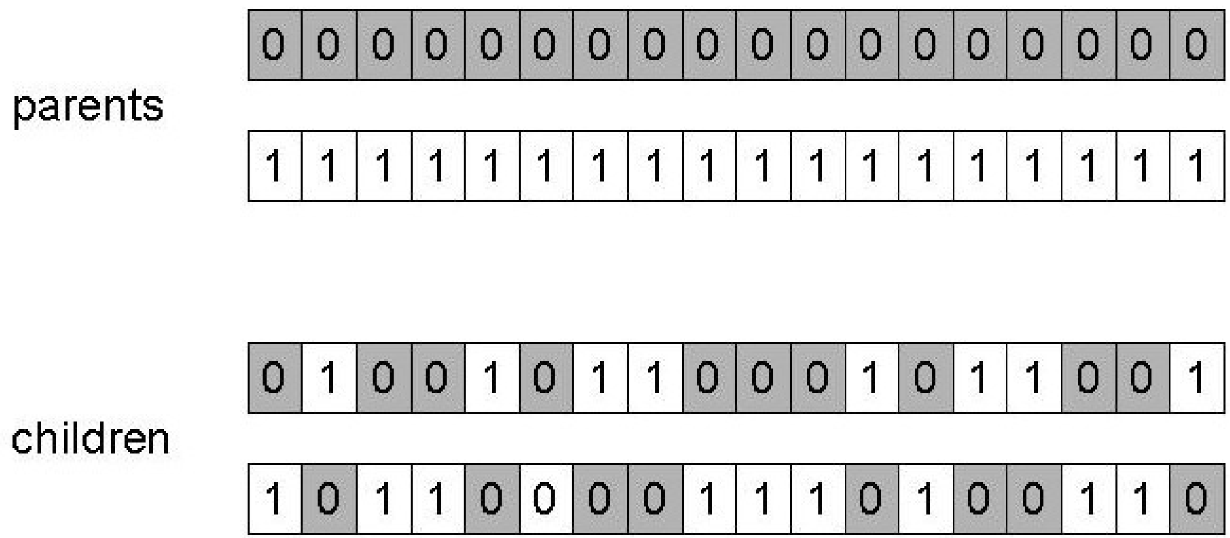
n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)

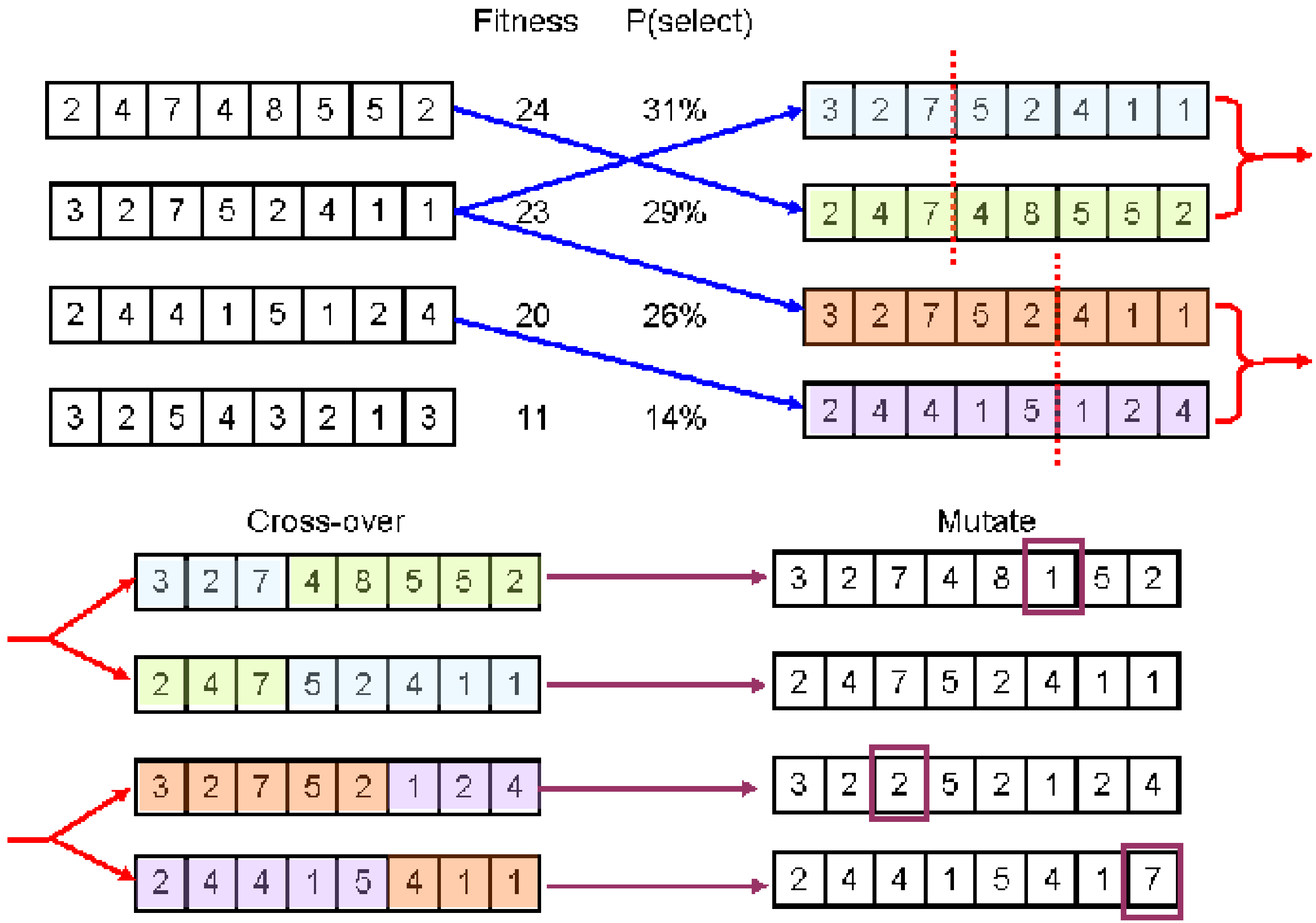


Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy for the second child
- Inheritance is independent of position



Genetic Algorithm



Memetic Algorithm

- Memetic Algorithm =
  Genetic Algorithm +
  Local Search

- E.g.:
  - LS after mutation
  - LS after crossover

Demo

- http://www.rennard.org/alife/english/gavintrgb.html

Ant Colony Optimization

- Another “Biological Analogue”

- Observation: Ants are very simple creatures, but can achieve complex behaviours

- Use pheromones to communicate

Ant Colony Optimization

- Ant leaves a pheromone trail
- Trails influence subsequent ants
- Trails evaporate over time

- E.g. in TSP
  - Shorter Tours leave more pheromone
  - Evaporation helps avoid premature intensification

ACO for TSP

- $p_k(i,j)$ is prob. moving from *i* to *j* at iter *k*

$$p_k(i,j) = \begin{cases} \dfrac{[\tau_{i,j}^k]^\alpha [c_{i,j}]^\beta}{\sum_{h \in N_i} [\tau_{i,h}^k]^\alpha [c_{i,h}]^\beta} & \text{if } (i,j) \in N_i \\ 0 & \text{otherwise} \end{cases}$$

- α, β parameters

ACO for TSP

- Pheromone trail evaporates at rate ρ

$$\tau_{ij}^{k} = \rho \tau_{ij}^{k-1}(t) + \Delta \tau_{ij}$$

- Phermone added proportional to tour quality

$$\Delta \tau_{i,j}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if } (i, j) \in \text{tour} \\ 0 & \text{otherwise} \end{cases}$$

References

- Emile Aarts and Jan Karel Lenstra (Eds), *Local Search in Combinatorial Optimisation* Princeton University Press, Princeton NJ, 2003

- Holger H. Hoos and Thomas Stützle, *Stochastic Local Search, Foundations and Applications,* Elsevier, 2005