Algorithmes Évolutionnaires (M2 MIAGE IA²)

Andrea G. B. Tettamanzi
Laboratoire I3S – Équipe SPARKS
andrea.tettamanzi@univ-cotedazur.fr

univ-cotedazur.fr

Séance 4
Représentations et opérateurs spécialisés

Plan

- Vers les applications du monde réal : programmes évolutionnaires
- Représentations pour
  - Problèmes d’allocation et d’optimisation de paramètres
  - Problèmes d’association
  - Problèmes de permutation
- Operateurs spécialisés pour
  - Problèmes d’allocation
  - Problèmes de permutation

Evolution Programs

Slogan:

Genetic Algorithms + Data Structures = Evolution Programs

Key ideas:

- use a data structure as close as possible to object problem
- write appropriate genetic operators
- ensure that all genotypes correspond to feasible solutions
- ensure that genetic operators preserve feasibility

Data Structure Close to Object Problem

- Exploit information about the problem
- Use natural representation suggested by the object problem
- Manipulate meaningful solution elements

Write Appropriate Genetic Operators

- Exploit information about solution structure
- Manipulate meaningful solution elements
- Preserve feasibility of candidate solutions
- Mutation, Recombination

Ensure Genotypes = Feasible Solutions

- Processing infeasible solution is a waste of time
- Feasible solutions = smaller search space
- Unfortunately, not always possible
- Problem with interacting constraints

Preserve Feasibility

- Genetic operators should respect constraints
- In-depth understanding of problem is required
- Ad hoc genetic operators
- Lower degree of s/w reuse, more development required

Gene “Orthogonality”

- Advisable to design encodings where genes are orthogonal
- Semantics of each gene should:
  - depend on its value (*allele*);
  - not depend on the value of other genes.
- Epistasis: interactions among genes

Designing Representations

- Representation is a critical success factor for Eas
- No cookbook available
- Coarse classification of problems:
  - allocation problems (“pie” problems)
  - parameter optimization problems
  - permutation problems
  - mapping problems

Allocation ("Pie") Problems

- Given:
  - a limited amount of resources
  - a set of opportunities (or tasks)
  - a cost/benefit function
- Determine:
  - optimal allocation of resources to opportunities
- Subject to:
  - all resources must be employed
  - resource limit cannot be exceeded
  - other problem-dependent constraints

Pie Problem Example

- Limited resources: €100,000
- Opportunity set:
  - W: European Equity
  - X: American Equity
  - Y: Euro Bonds
  - Z: US Bonds

- Candidate solution:

  Invest €15,000 in X, €25,000 in Y, €20,000 in Y, €40,000 in Z

Pie Problem Representation

- Vector of absolute amounts
  - >= 0
  - sum up to total resources
- Vector of percentages
  - >= 0
  - sum up to 100%
- Constraint elimination...

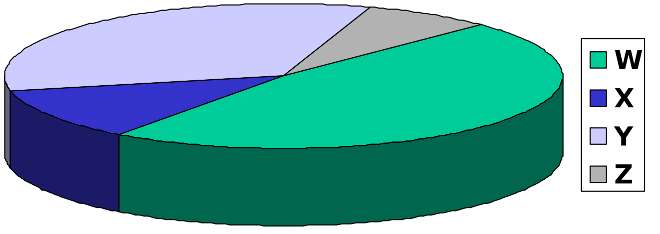
“Clever” Representation

W	X	Y	Z
128	32	90	20
0–255	0–255	0–255	0–255

“Clever” Representation

W	X	Y	Z
128	32	90	20
0–255	0–255	0–255	0–255

X = 32/270 = 11.85%



Parameter Optimization Problems

$f(x_1, x_2, \ldots, x_n)$

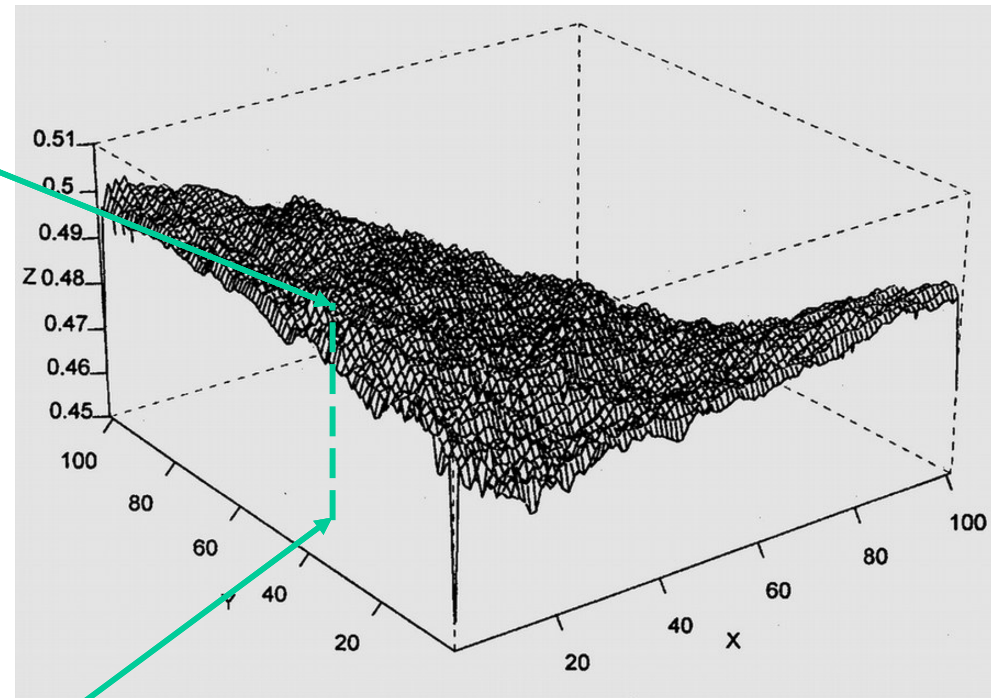


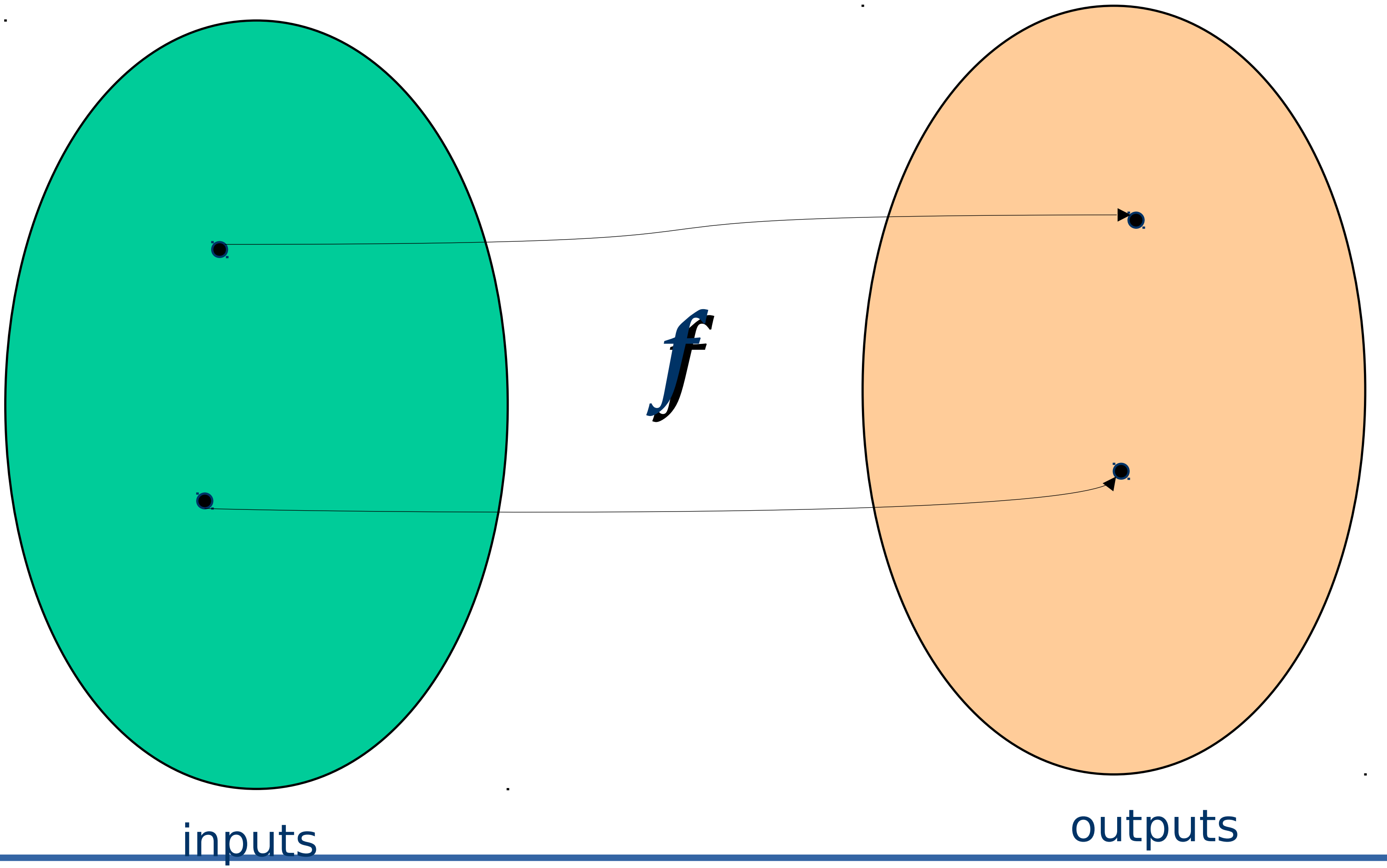
Figure 13.2 Downside risk surface

$(x_1, x_2, \ldots, x_n)$

Solution Representation

- A solution is an assignment of values to parameters
- Natural representation: a vector

Mapping Problems



Mapping Problem Examples

- Symbolic Regression
- Time Series Prediction
- System Modeling
- Data Mining
- Control

Solutions

- Mathematical Formulas
- Simple Programs
- Decision Trees
- Finite State Machines
- Neural Networks
- Fuzzy Rule Bases
- etc…

Solution Representation

- GP Trees a natural representation for
  - mathematical formulas
  - programs

- Advantages
  - well-established set of genetic operators and techniques
- Drawbacks
  - results are not easy to interpret/understand
  - sensitive on the choice of GP primitives

Alternative Approaches (1)

- Pre-determine a parametric model for the mapping
- Fit the model to data
- Problem reduces to parameter optimization problem

- Advantages
  - parameter optimization is in general simpler
- Drawbacks
  - a simplistic model could lead to nonsatisfactory solutions

Alternative Approaches (2)

- Non-parametric models like
  - neural networks
  - (fuzzy) rule bases
  - (fuzzy) decision trees
  - etc.

- Where structure is not pre-determined

Permutation Problems

- Given:
  - a discrete set of objects
- Determine:
  - a suitable permutation for those objects

Permutation Problem Examples

- Traveling Salesman Problem
- Timetable Problem
- Job Shop Scheduling
- Vehicle Routing Problem
- ...

Representing a Permutation

- Assign an integer to each permutation object
- A permutation is a list of integers, e.g.:
  1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7
- Direct (or “path”) representation:
  - list permutation elements
  - example: (1, 2, 4, 3, 8, 5, 9, 6, 7)

Adjacency Representation

- One integer per object
- *i*th integer denotes the next element after object *i*
- Example: (2, 4, 8, 3, 9, 7, 1, 5, 6)
  - 2 comes after 1 (1st position)
  - 4 comes after 2 (2nd position)
  - 8 comes after 3 (3rd position)
  - etc.

  - Result: 1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7

Ordinal Representation

- Vector of n — 1 integers
  ([1..n], [1..n-1], [1..n-2], …, [1..2])

- Decoding:
  - place all object in a list
  - for i = 1 to n — 1,
    - remove the x[i]-th object from the list
    - append it to the permutation

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

↓

Objects: (1, 2, 3, 4, 5, 6, 7, 8, 9)

Permutation:

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

↙

Objects: (2, 3, 4, 5, 6, 7, 8, 9)

Permutation: 1

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

↙

Objects: (3, 4, 5, 6, 7, 8, 9)

Permutation: 1 - 2

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

Objects: (3, 5, 6, 7, 8, 9)

Permutation: 1 - 2 - 4

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

Objects: (5, 6, 7, 8, 9)

Permutation: 1 - 2 - 4 - 3

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

Objects: (5, 6, 7, 9)

Permutation: 1 - 2 - 4 - 3 - 8

Etc...

Ordinal Representation Example

Genotype: (1, 1, 2, 1, 4, 1, 3, 1, 1)

Objects: ()

Permutation: 1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7

Matrix Representation

- Square {0, 1} matrix
- Entry (i, j) is 1 iff i th object before j th object

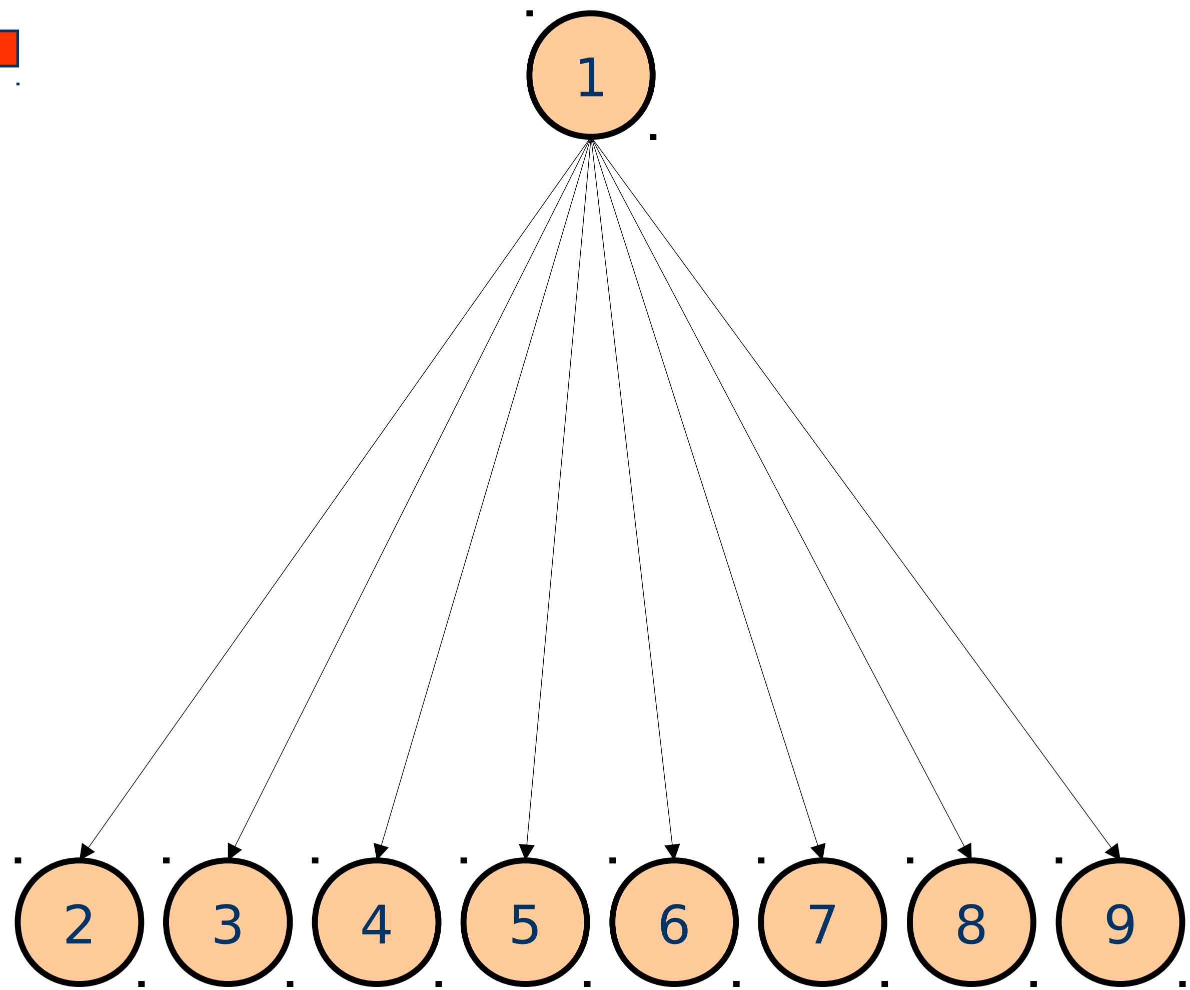
- Decoding:
  - build partial order directed graph
  - eliminate cycles

Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0

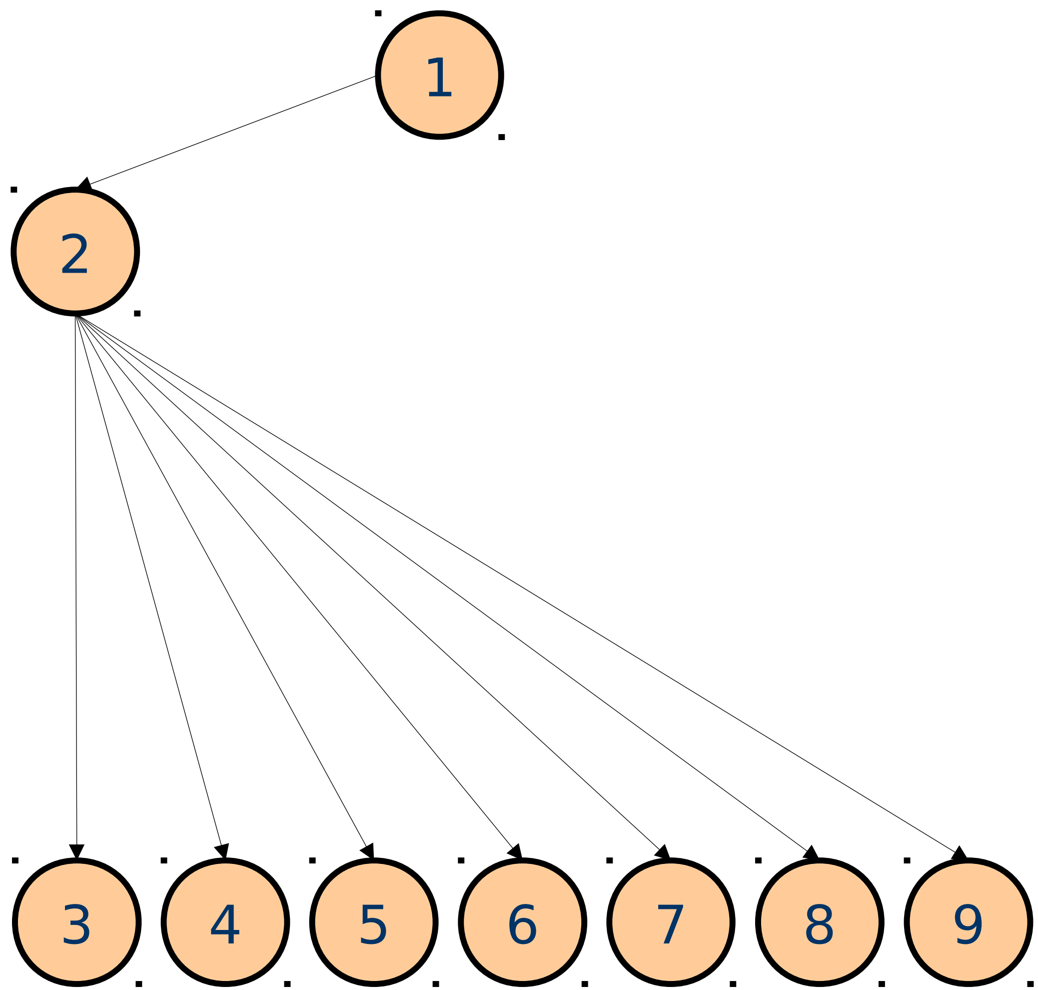
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



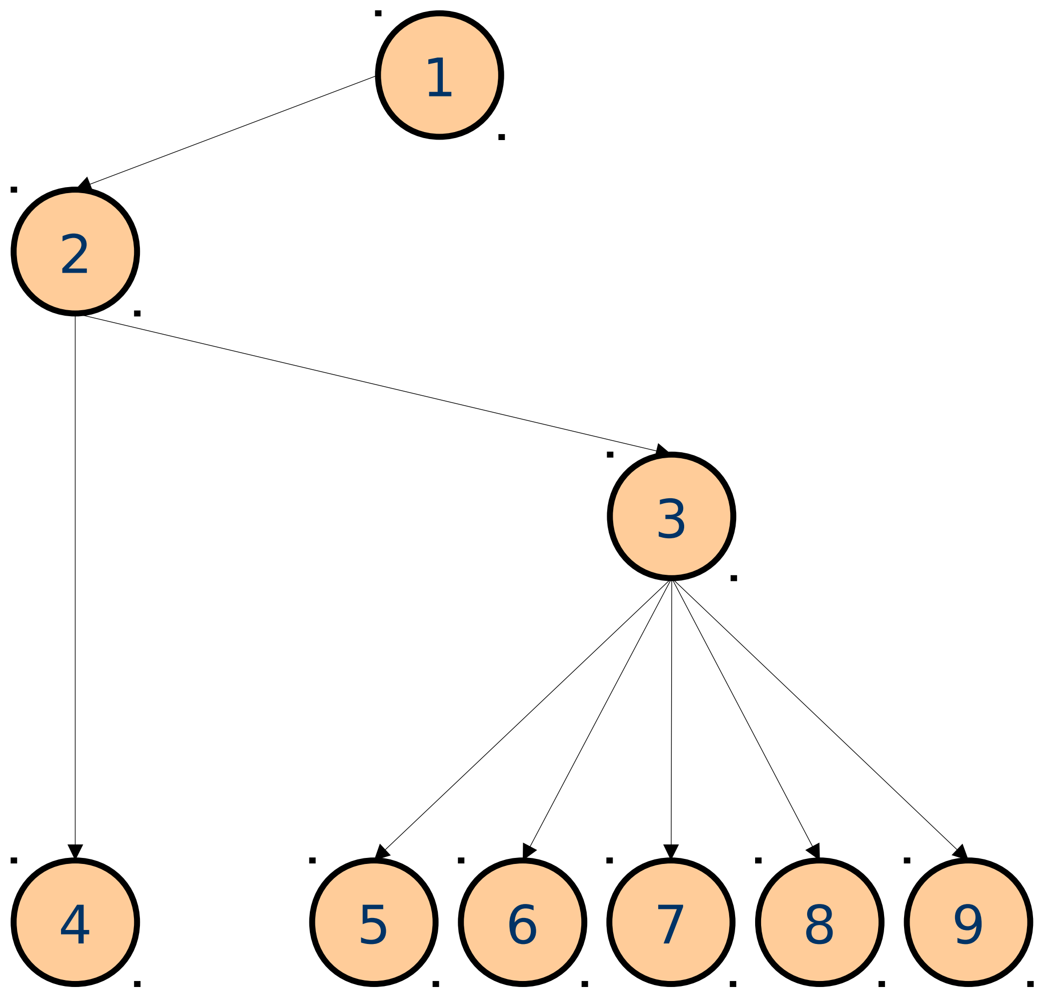
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



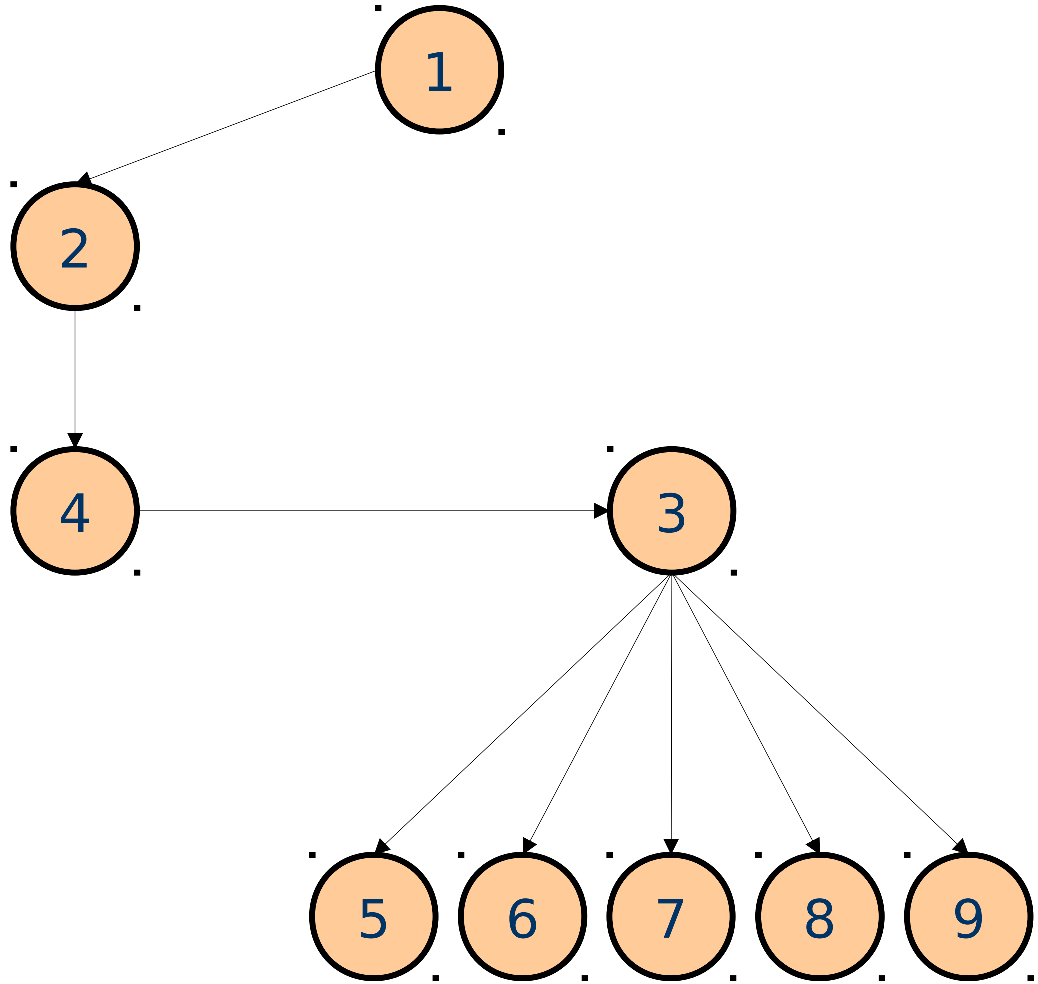
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



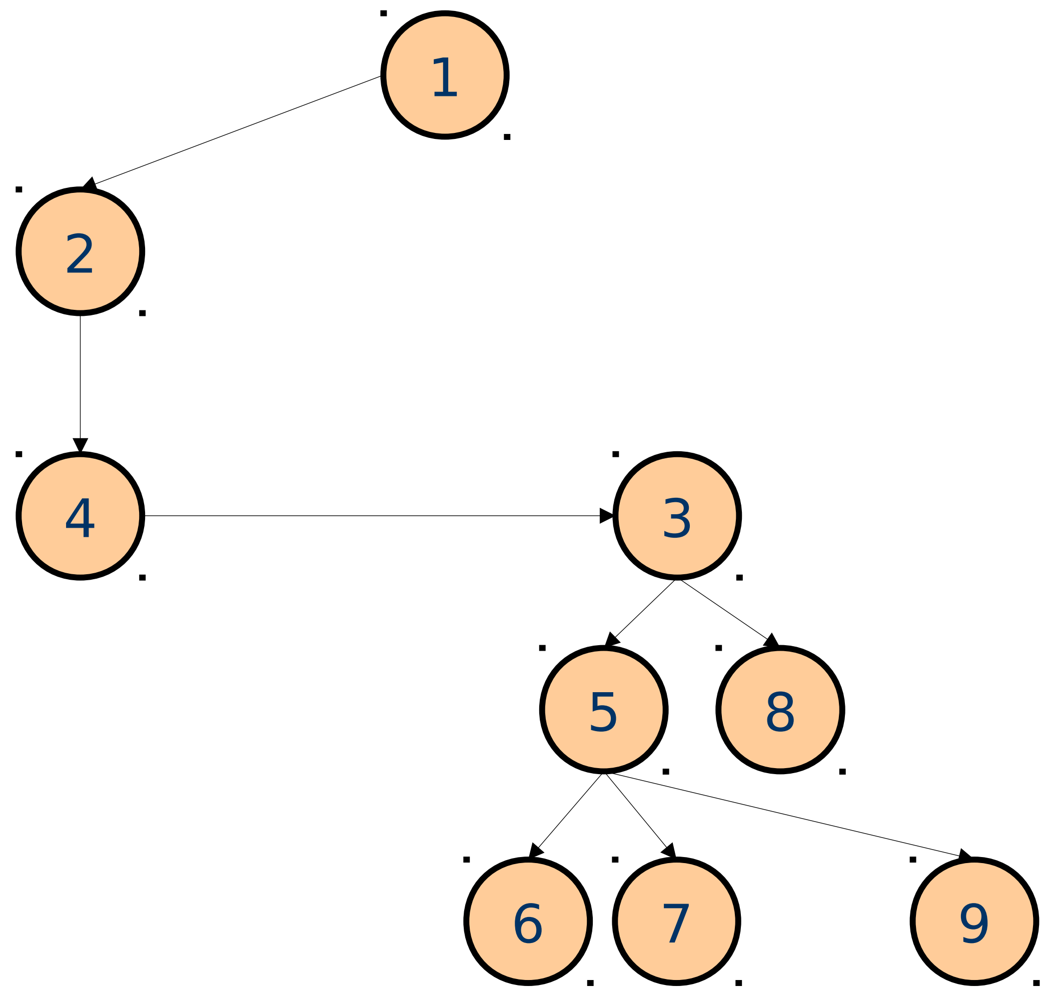
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



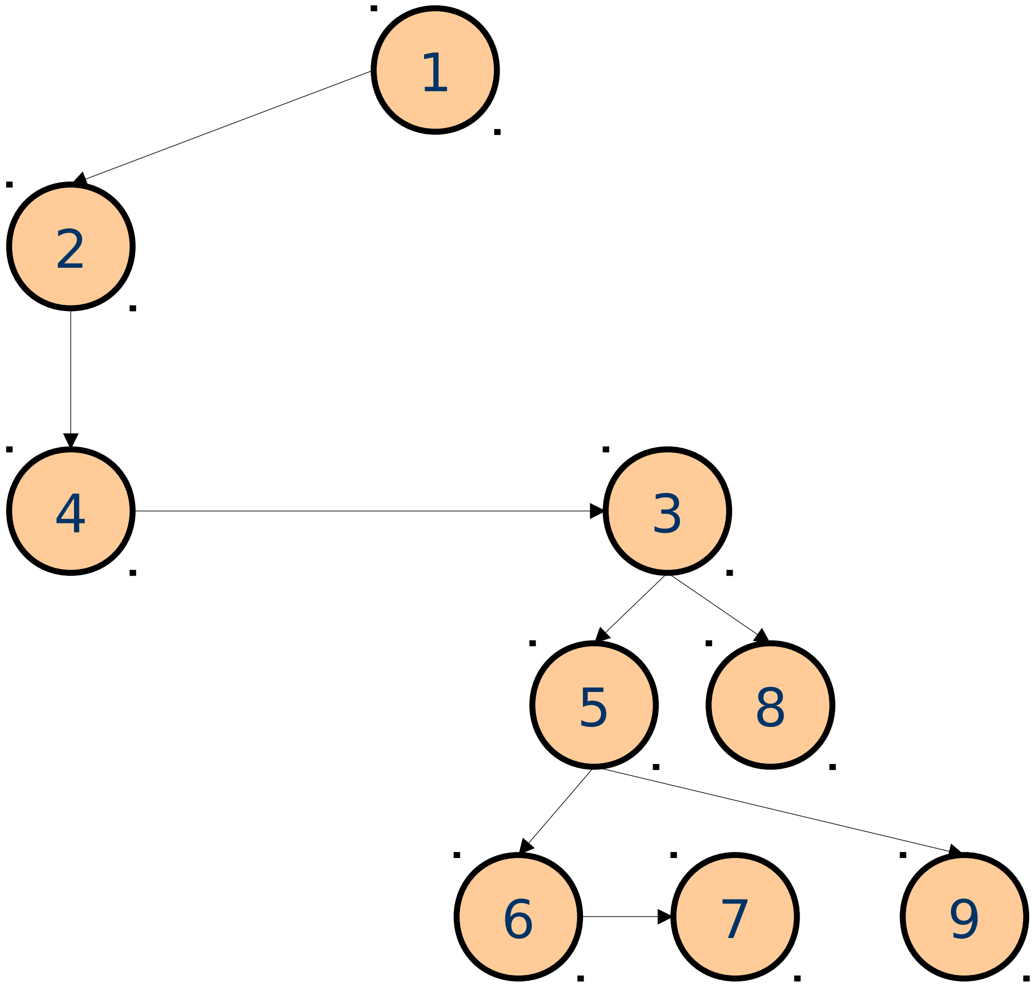
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



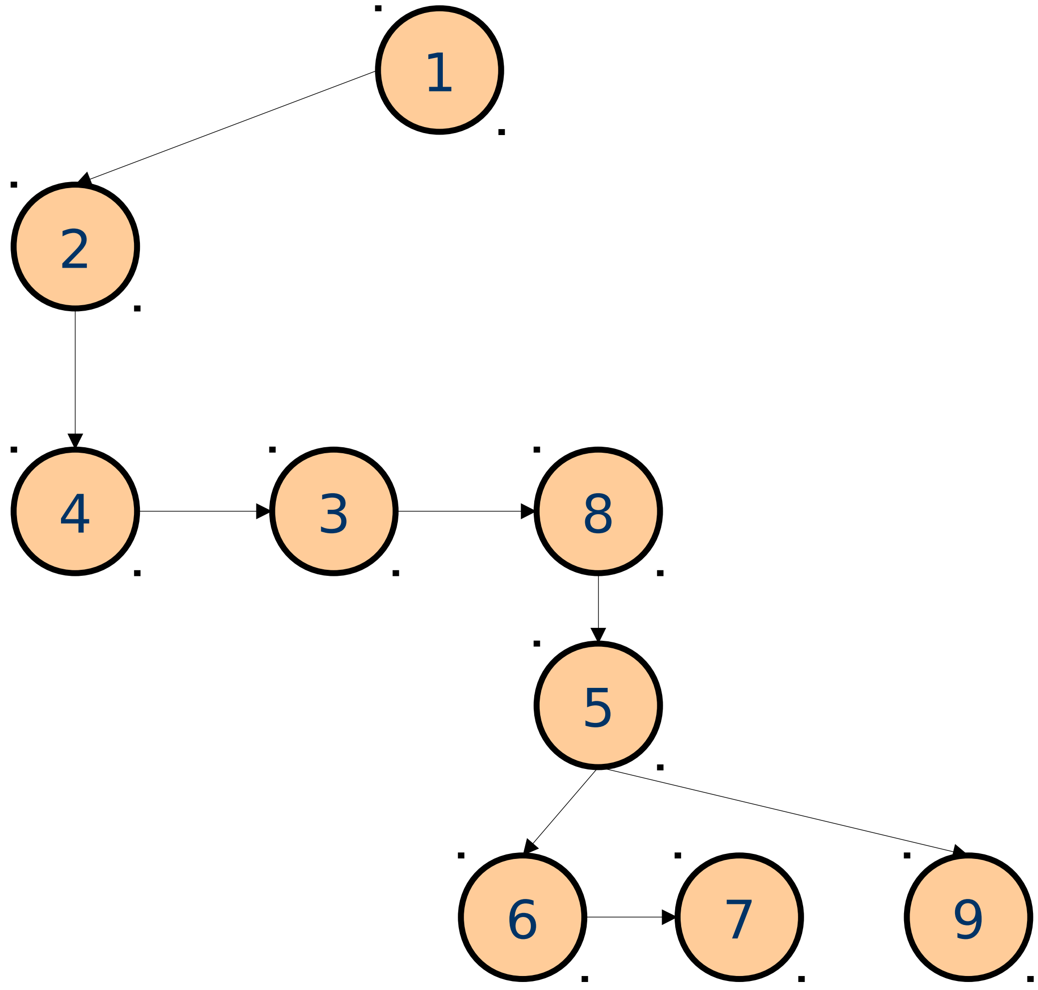
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



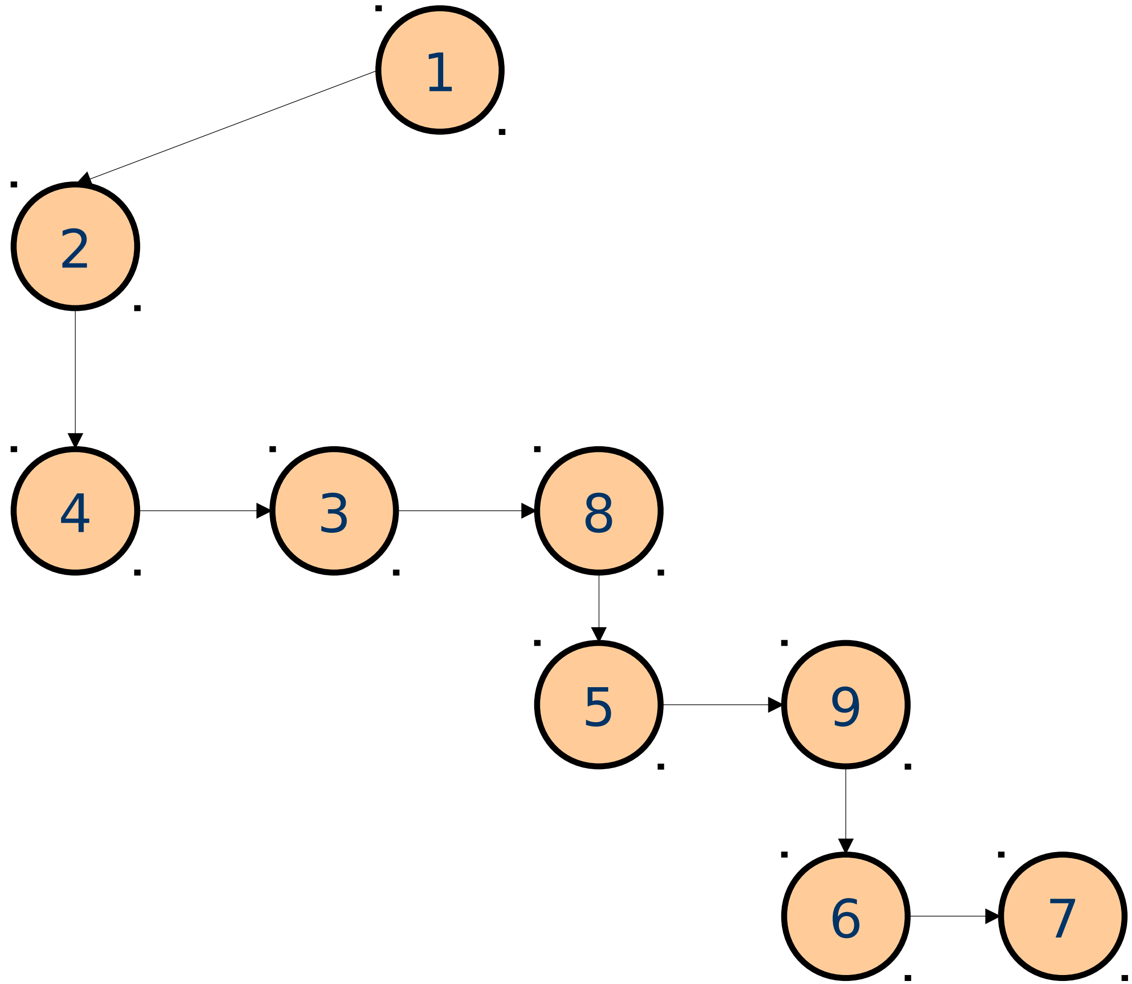
Matrix Representation Example

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



Matrix Representation Example

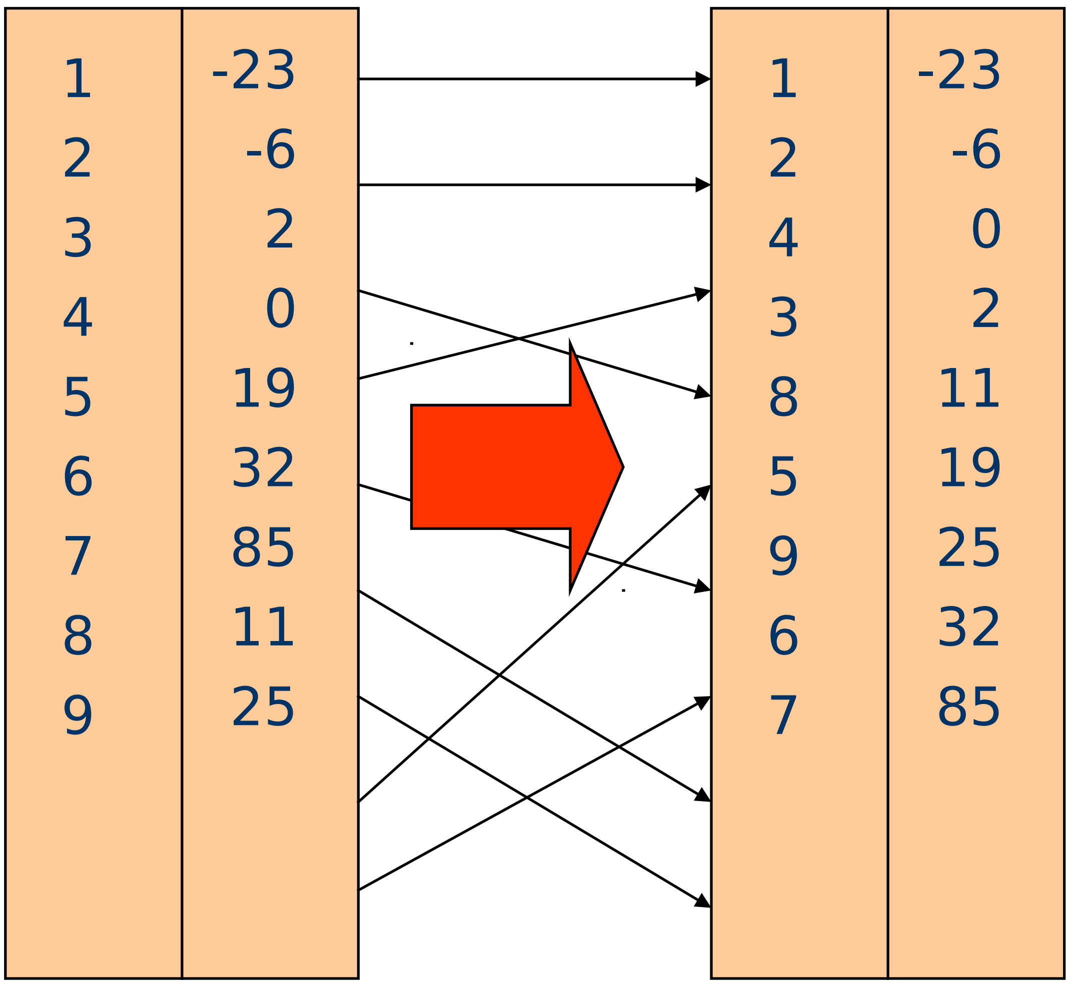
0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0



Sorting Representation

- Associate a real weight to each object
- Sort object according to their weight
- The order of objects is the permutation

Example



Permutation:

1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7

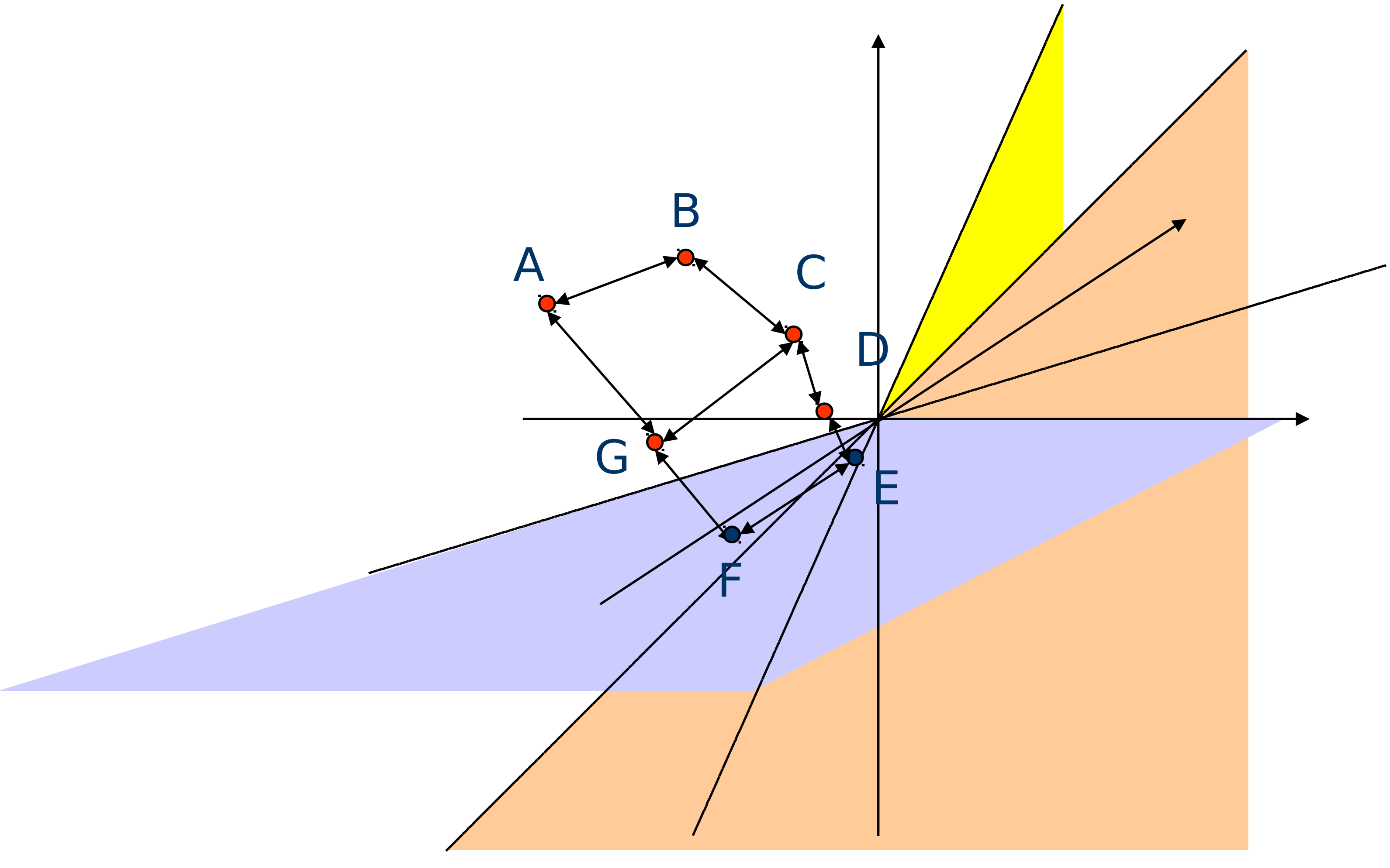
Degeneracy

- Many different genotypes correspond to the same permutation

- In particular, the n-dimensional Euclidean space gets partitioned into n! “slices”, each corresponding to one partition
- All n! “slices” touch at the origin

- Not necessarily bad for Eas
- Leads to the emergence of so-called “neutral networks”

Degeneracy and Neutral Networks



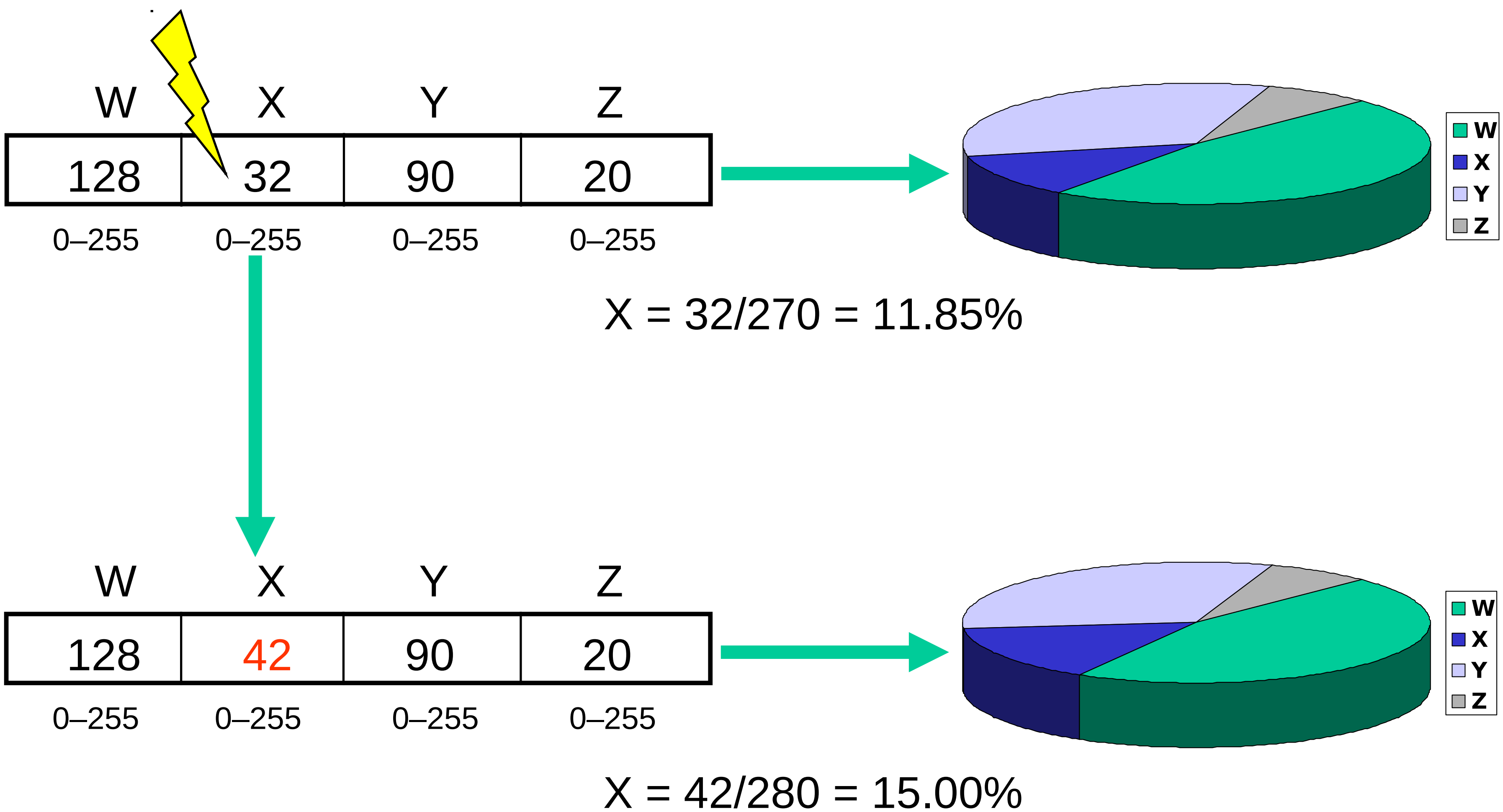
Discussion of Sorting Representation

- Advantages:
  - no need for specialized operators
  - presence of neutral networks
- Drawbacks:
  - search space is much larger than solution space
  - decoder has a complexity of O(n log n)

Specialized Operators

- Straightforward mutation and recombination operators may produce illegal chromosomes
- Devise specialized versions adapted to each particular representation

Mutation for Pie Problems



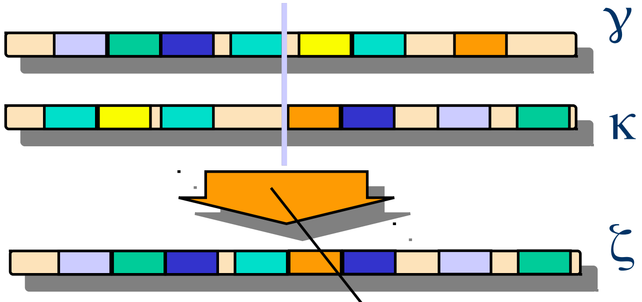
Recombination for Pie Problems

- Simply performing one-point crossover or uniform crossover is not satisfactory

- Gene semantics depends on their context
- Example:
  - In (9, 1, 0), 9 "means" 90%
  - In (9, 46, 45), 9 "means" 9%

- We need to take this meaning into account

"Balanced" Crossover

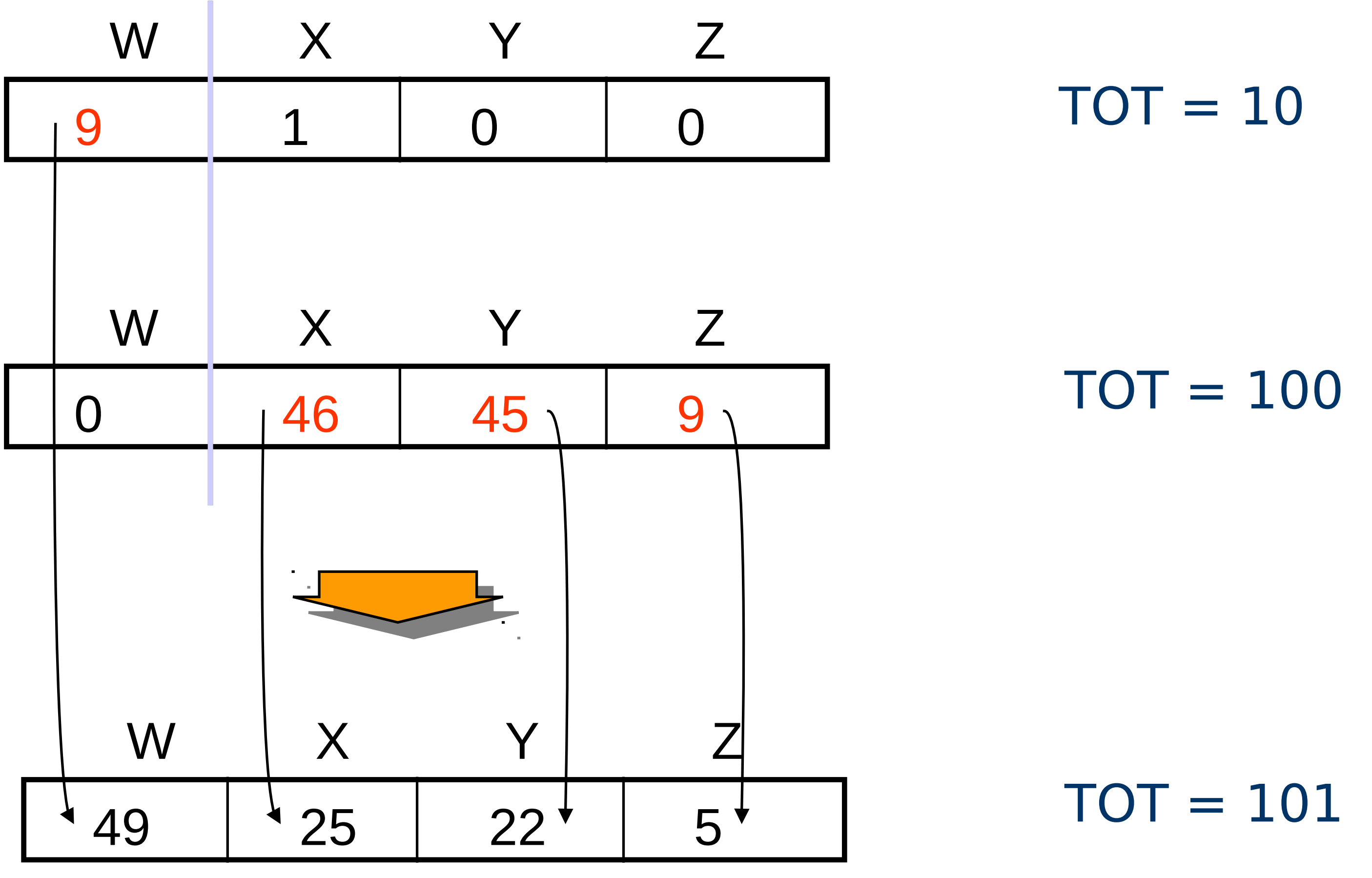


$$\mathrm{TOT}_\gamma = \sum_i \gamma_i,$$

$$\mathrm{TOT}_\kappa = \sum_i \kappa_i;$$

$$\zeta_i = \min\left(g_{\max}, \gamma_i \frac{\mathrm{TOT}_\gamma + \mathrm{TOT}_\kappa}{2\,\mathrm{TOT}_\gamma} \right).$$

“Balanced” Crossover Example



Discussion of "Balanced" Crossover

- What do we learn from this simple example?
- An operator should not only preserve feasibility
- It should operate at the semantic level

Mutations for Permutation Problems

(Path representation)

- Insertion Mutation
- Displacement Mutation
- Swap Mutation
- Heuristic Mutation
- ...

Insertion Mutation

- Randomly pick a position, then insert its content into a random position
- Example:

(2, 4, 8, 3, 9, 7, 1, 5, 6)

(2, 7, 4, 8, 3, 9, 1, 5, 6)

Displacement Mutation

- A generalization of Insertion Mutation
- Move various elements at once
- Example:

(2, 4, 8, 3, 9, 7, 1, 5, 6)

(2, 7, 4, 3, 8, 5, 9, 1, 6)

Swap Mutation

- Randomly pick two position, then swap their contents
- Example:

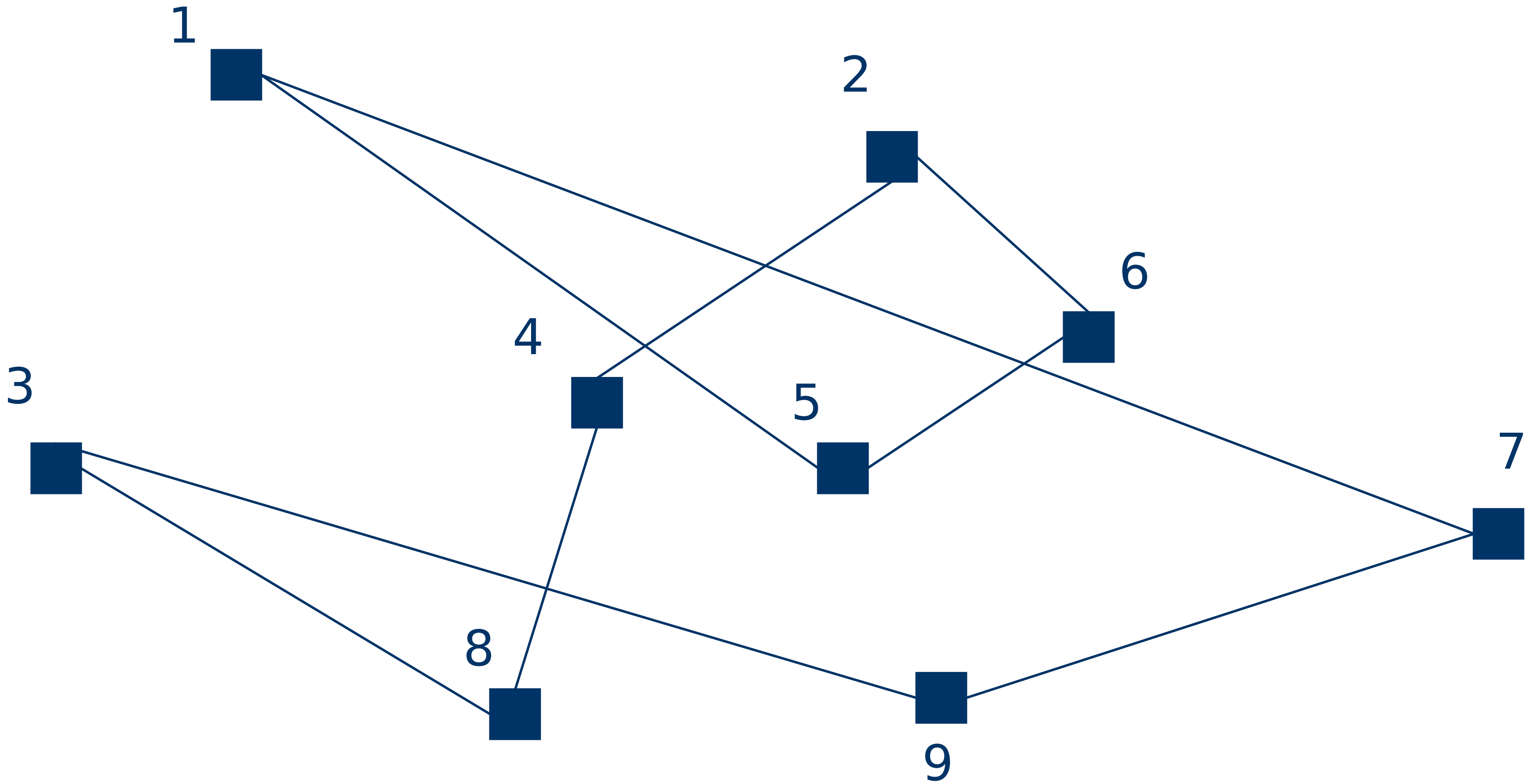
(2, 4, 8, 3, 9, 7, 1, 5, 6)

(2, 7, 8, 3, 9, 4, 1, 5, 6)

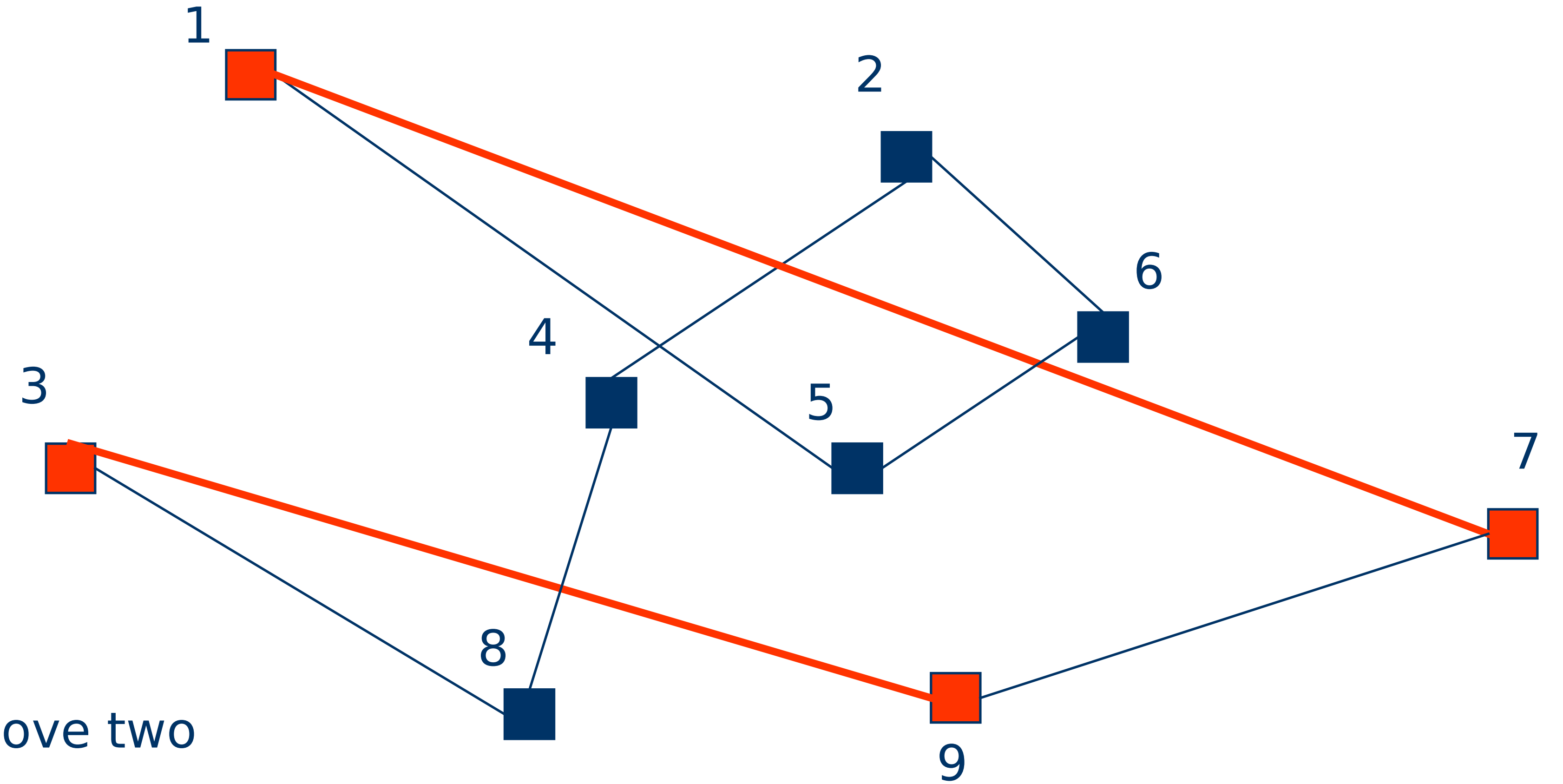
Heuristic Mutations

- Good perturbation heuristics are known for most combinatorial optimization problems from local optimization techniques
- Idea: use those moves as mutation operators
- Example:
  - 2-opt heuristics in TSP: remove two edges and reconnect the two resulting paths in a different way

2-opt Mutation

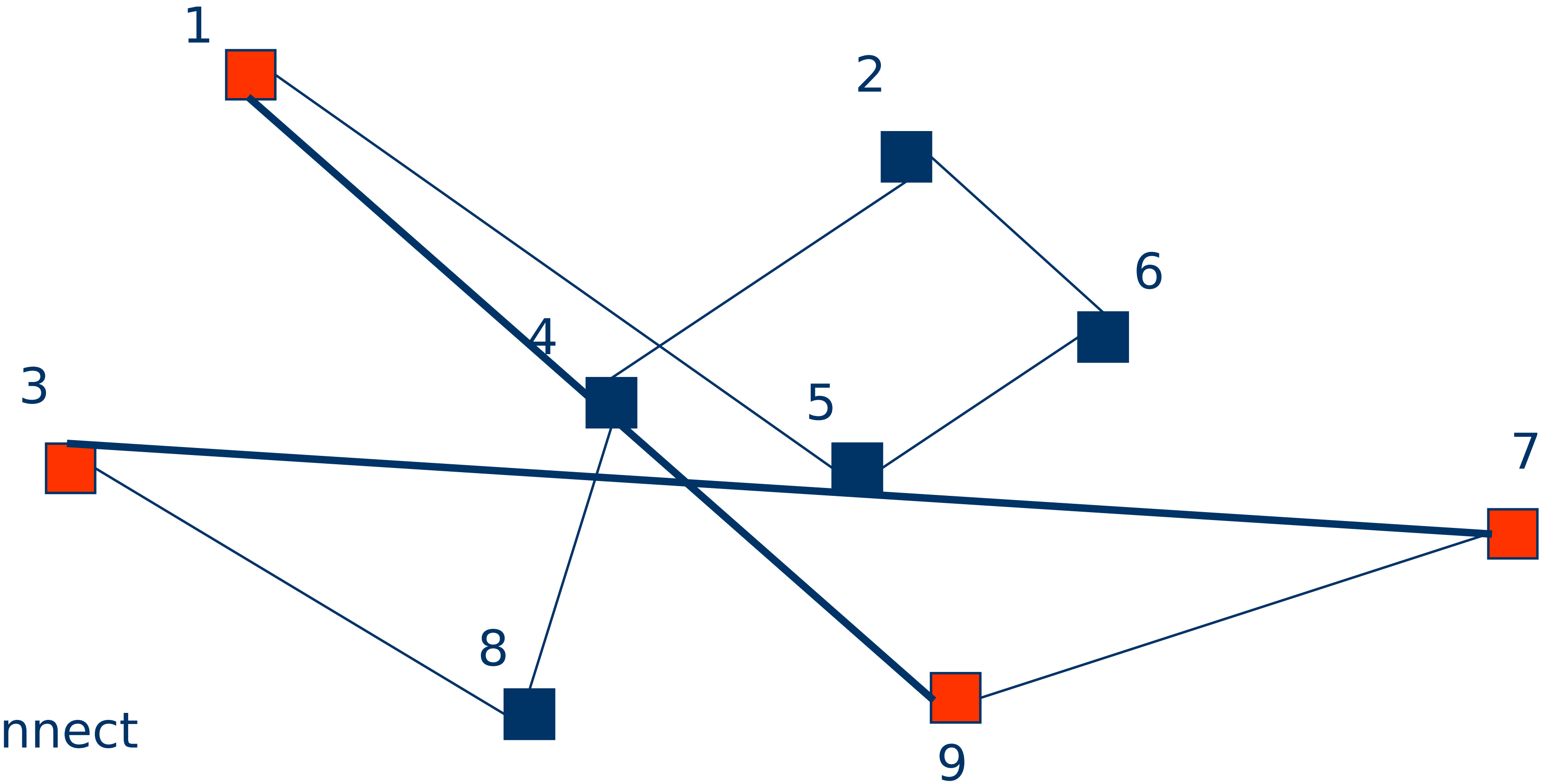


2-opt Mutation



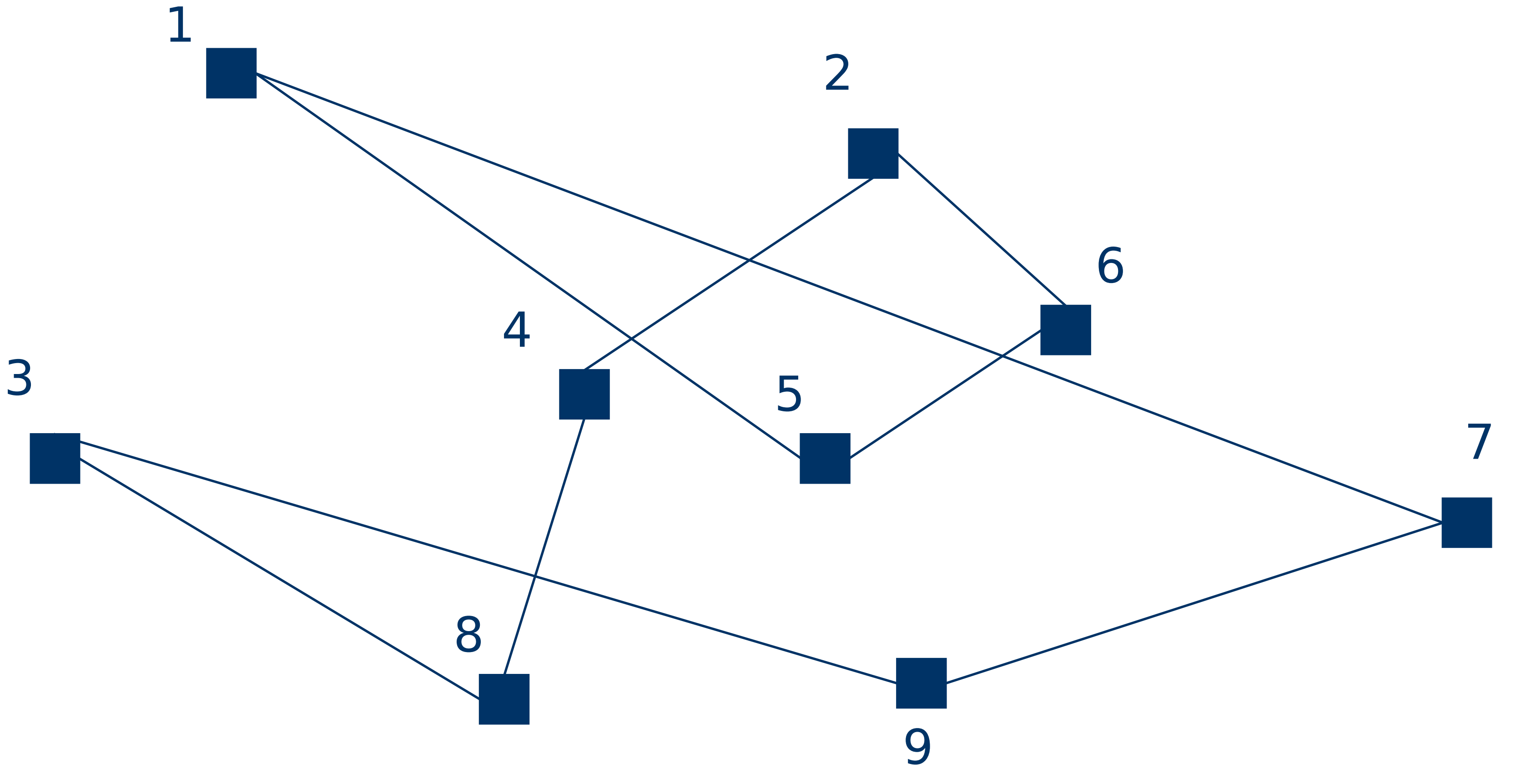
Remove two
edges at random...

2-opt Mutation

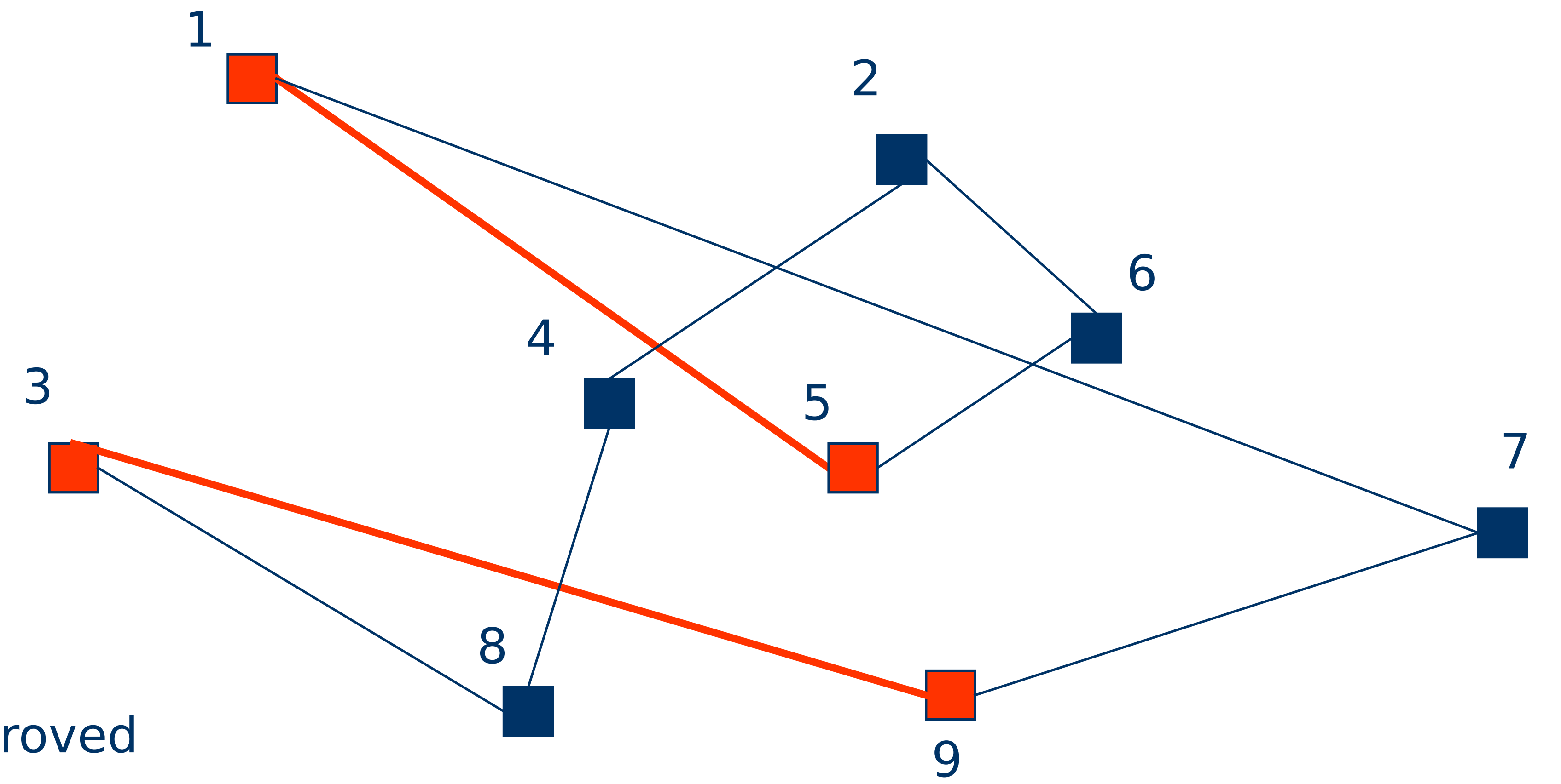


Reconnect
the path...

2-opt Mutation (2)

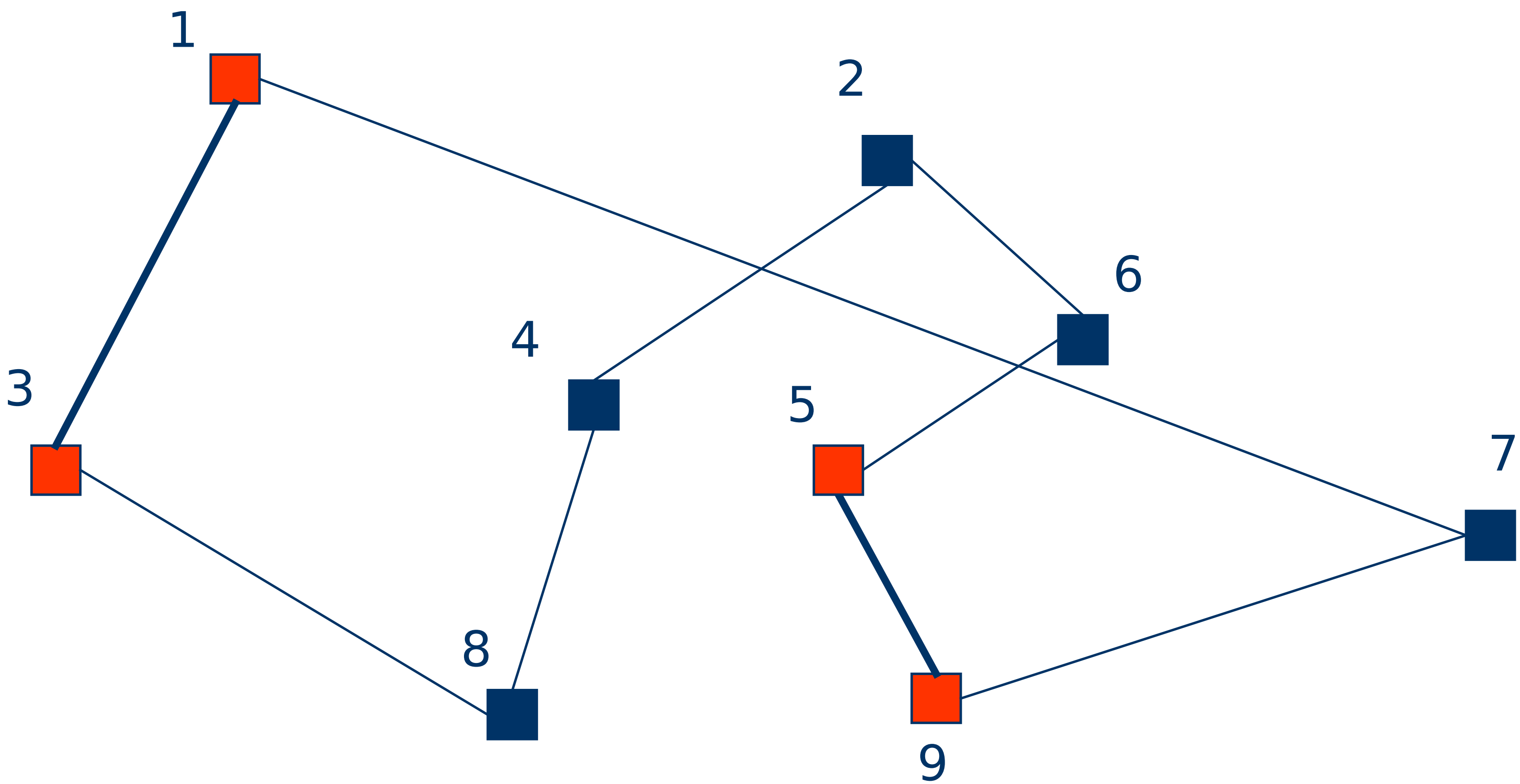


2-opt Mutation



Improved
solution

2-opt Mutation



Recombinations for Permutation Problems

(Path representation)

- Order Crossover (Davis, 1995)
- Partially Mapped Crossover (Goldberg and Lingle, 1985).
- Position-Based Crossover
- Order-Based Crossover
- Cycle Crossover
- ...

Order Crossover

- Give two parents P1 and P2
- Select a random substring S of P1
- Copy substring S to the first offspring O1
- Delete from P2 the elements in S
- Insert the remaining elements of P2 into empty position of O1
- Copy the remaining elements of P2 into O2
- Fill the empty positiond of O2 with the elements in S

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6) S = (8, 3, 9, 7)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6) S = (8, 3, 9, 7)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)

- O1 = (_, _, 8, 3, 9, 7, _, _, _)
- O2 = (_, _, _, _, _, _, _, _, _)

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6) S = (8, 3, 9, 7)
- P2 = (_, _, _, 6, 5, 4, _, 2, 1)

- O1 = (_, _, 8, 3, 9, 7, _, _, _)
- O2 = (_, _, _, _, _, _, _, _, _)

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6) S = (8, 3, 9, 7)
- P2 = (_, _, _, 6, 5, 4, _, 2, 1)

- O1 = (6, 5, 8, 3, 9, 7, 4, 2, 1)
- O2 = (_, _, _, _, _, _, _, _, _)

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6) S = (8, 3, 9, 7)
- P2 = (_, _, _, 6, 5, 4, _, 2, 1)

- O1 = (6, 5, 8, 3, 9, 7, 4, 2, 1)
- O2 = (_, _, _, 6, 5, 4, _, 2, 1)

Order Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6) S = (8, 3, 9, 7)
- P2 = (_, _, _, 6, 5, 4, _, 2, 1)

- O1 = (6, 5, 8, 3, 9, 7, 4, 2, 1)
- O2 = (8, 3, 9, 6, 5, 4, 7, 2, 1)

- Done!

Partially Mapped Crossover (PMX)

- Randomly pick two crossover points
- Exchange the two substrings within the crossover points
- Fill the remaining positions in the offspring by mapping the elements of the parents:
  - if an element does not occur in the substring within the crossover points, leave it unchanged
  - otherwise, replace it with the element in the substring of the other parent

PMX Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)

PMX Example

- P1 = (2, 4, | 8, 3, 9, 7, | 1, 5, 6)
- P2 = (9, 8, | 7, 6, 5, 4, | 3, 2, 1)

PMX Example

- P1 = (2, 4, | 8, 3, 9, 7, | 1, 5, 6)
- P2 = (9, 8, | 7, 6, 5, 4, | 3, 2, 1)

- O1 = (2, 4, | 7, 6, 5, 4, | 1, 5, 6)
- O2 = (9, 8, | 8, 3, 9, 7, | 3, 2, 1)

PMX Example

- P1 = (2, 4, | 8, 3, 9, 7, | 1, 5, 6)
- P2 = (9, 8, | 7, 6, 5, 4, | 3, 2, 1)

- O1 = (2, 8, | 7, 6, 5, 4, | 1, 3, 9)
- O2 = (6, 5, | 8, 3, 9, 7, | 4, 2, 1)

- Done!

Position-Based Crossover

- Select *k* random positions in P1
- Copy them into the corresponding positions of O1
- Fill the empty positions with the remaining elements *in the same order* as they occur in P2
- Build O2 by means of the dual operation

- O1 inherits
  - absolute positions from P1 for *k* elements
  - relative positions from P2 for the other elements

Order-Based Crossover

- Select *k* random positions
- Impose the order in which their elements appear in P1 to P2 to produce O1
- Impose the order in which their element appear in P2 to P1 to produce O2

Cycle Crossover

- Select a random position in P1
- Look up the content of the same position in P2 and look for the same element in P1
- Continue like that until going back to the initial position: i.e., until a cycle has formed
- Copy into O1 the positions of P1 containing elements of the cycle
- Fill the other positions of O1 with the elements found in P2
- Construct O2 in a complementary fashion

Cycle Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)
- Cycle = (8, 7, 4)

Cycle Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)
- Cycle = (8, 7, 4)

- O1 = (_, 4, 8, _, _, 7, _, _, _)
- O2 = (_, 8, 7, _, _, 4, _, _, _)

Cycle Crossover Example

- P1 = (2, 4, 8, 3, 9, 7, 1, 5, 6)
- P2 = (9, 8, 7, 6, 5, 4, 3, 2, 1)
- Cycle = (8, 7, 4)

- O1 = (9, 4, 8, 6, 5, 7, 3, 2, 1)
- O2 = (2, 8, 7, 3, 9, 4, 1, 5, 6)

- Done!

Conclusions

- Examples of specialized mutation and recombination operators adapted to particular representations
- Many degrees of freedom
- Not always obvious which alternative is best
- Empirical evaluation of alternatives