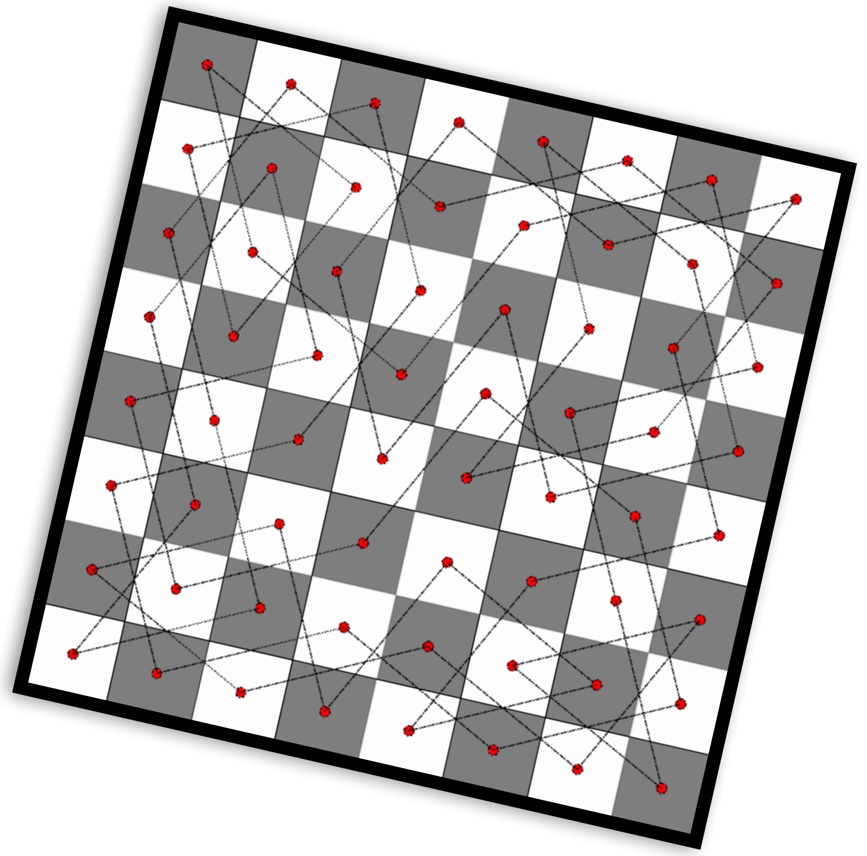
Advanced Algorithms for Optimization

Local Search

Part 1

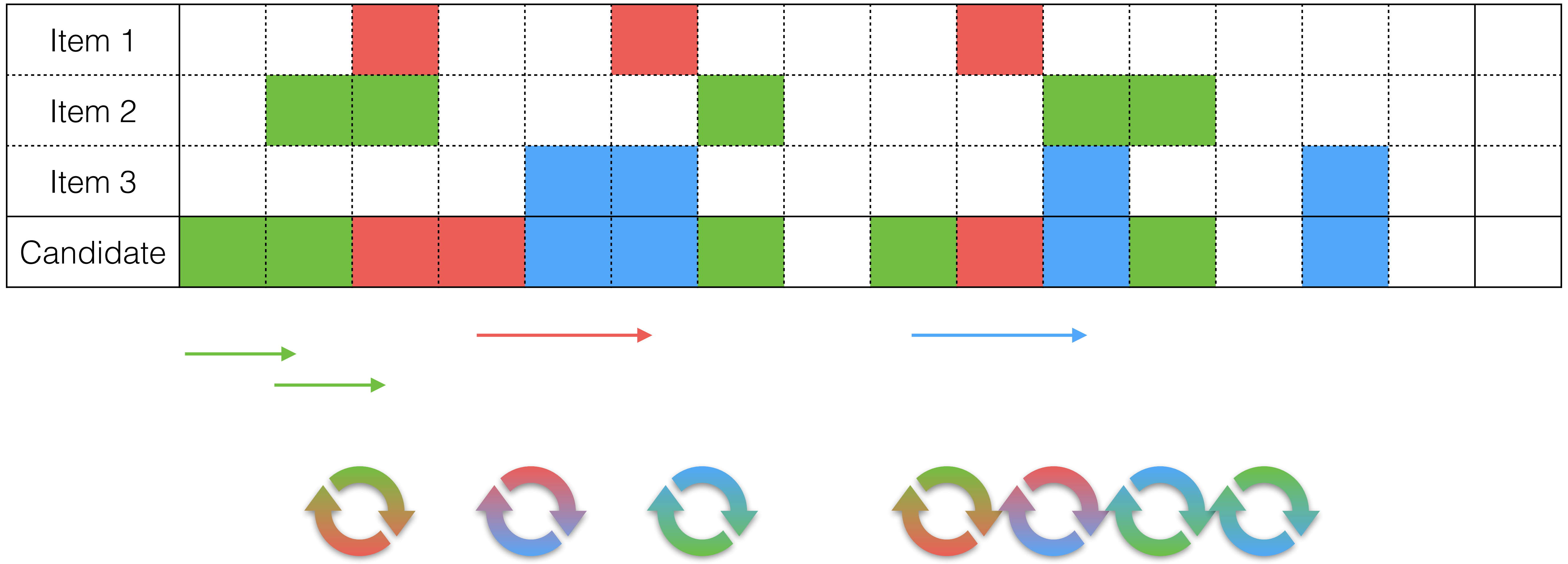
Pierre Schaus



https://github.com/pschaus/linfo2266

Next Project: A Discrete Lot Sizing Problem

- A set of orders for each item (at most one per time-slot). Strong constraint (deadlines)
- You must produce at most one per time slot (machine) to meet the deadlines
- Stocking cost (when you produce too early) + transition cost (adaptation of the machine to minimize



Item 1			🟥			🟥				🟥						
Item 2		🟩	🟩				🟩				🟩	🟩				
Item 3					🟦	🟦					🟦			🟦		
Candidate	🟩	🟩	🟥	🟥	🟦	🟦	🟩		🟩	🟥	🟦	🟩		🟦		

Transition Costs

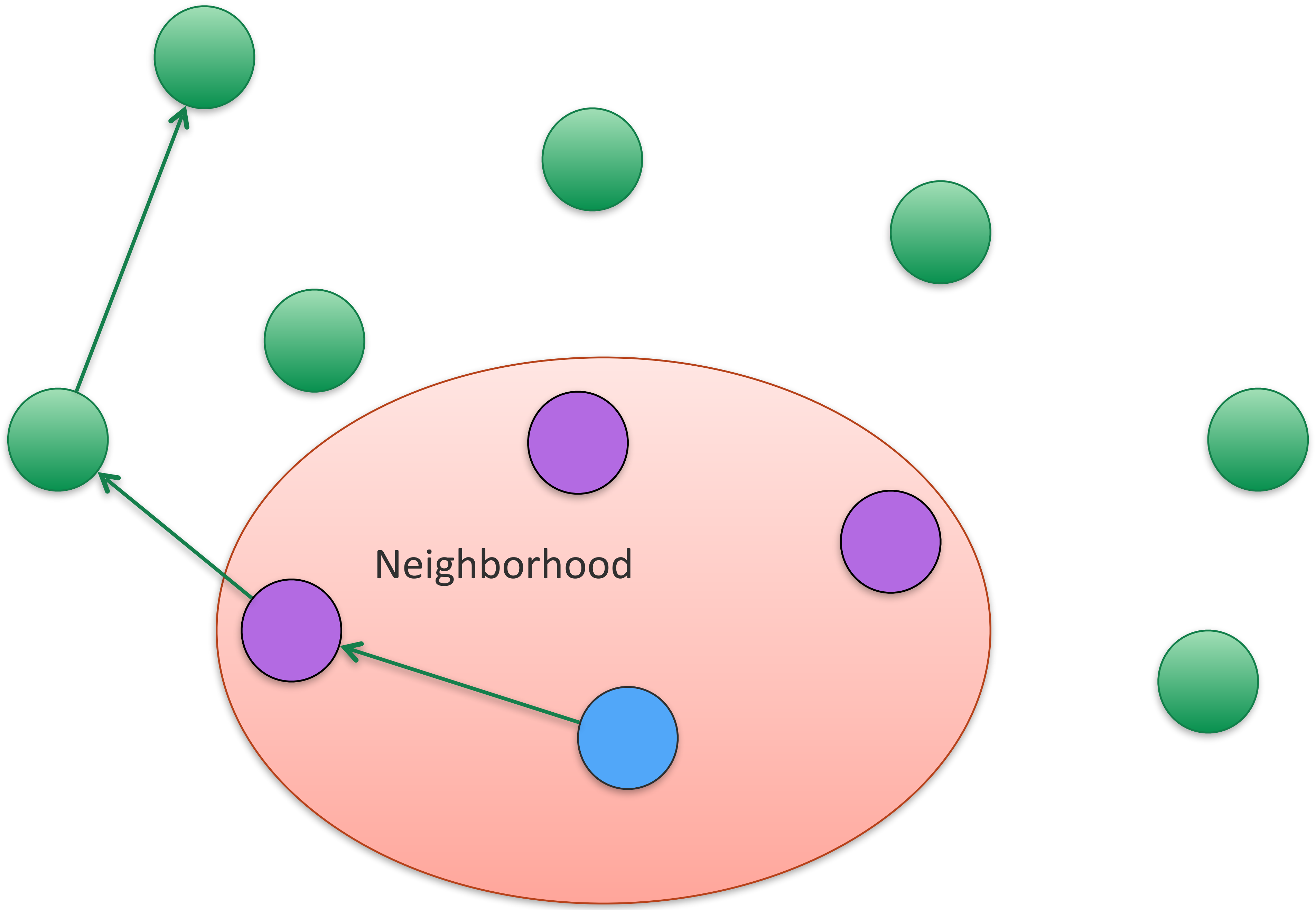
	🟩	🟥	🟦
🟩	0	4	6
🟥	2	0	4
🟦	1	3	0

Stocking Costs (per day)

🟩	0
🟥	5
🟦	10

Local Search: The idea

keep a single current state and move to neighbouring states to improve it



Generic Local Search

- Given a initial solution *s*,
- *N(s)* is the neighborhood of *s*.
- At a specific computation step, a neighbor may be legal or forbidden. *L(N(s),s)* is the set of legal moves of solution s.
- The operator S is in charge of selecting the move

```
Procedure LocalSearch(f,N,L,S,s)
    $s^* := s$;
    for $k := 1$ to $MaxTrials$ do
        if $sastifiable(s) \wedge f(s) < f(s^*)$ then
            $s^* := s$ ;
        $s := S(L(N(s), s), s)$ ;
    return $s^*$;
```

Improvement Heuristic

```
Procedure L-Improvement(N,s)
  return $\{n \in N | f(n) < f(s)\}$;
```

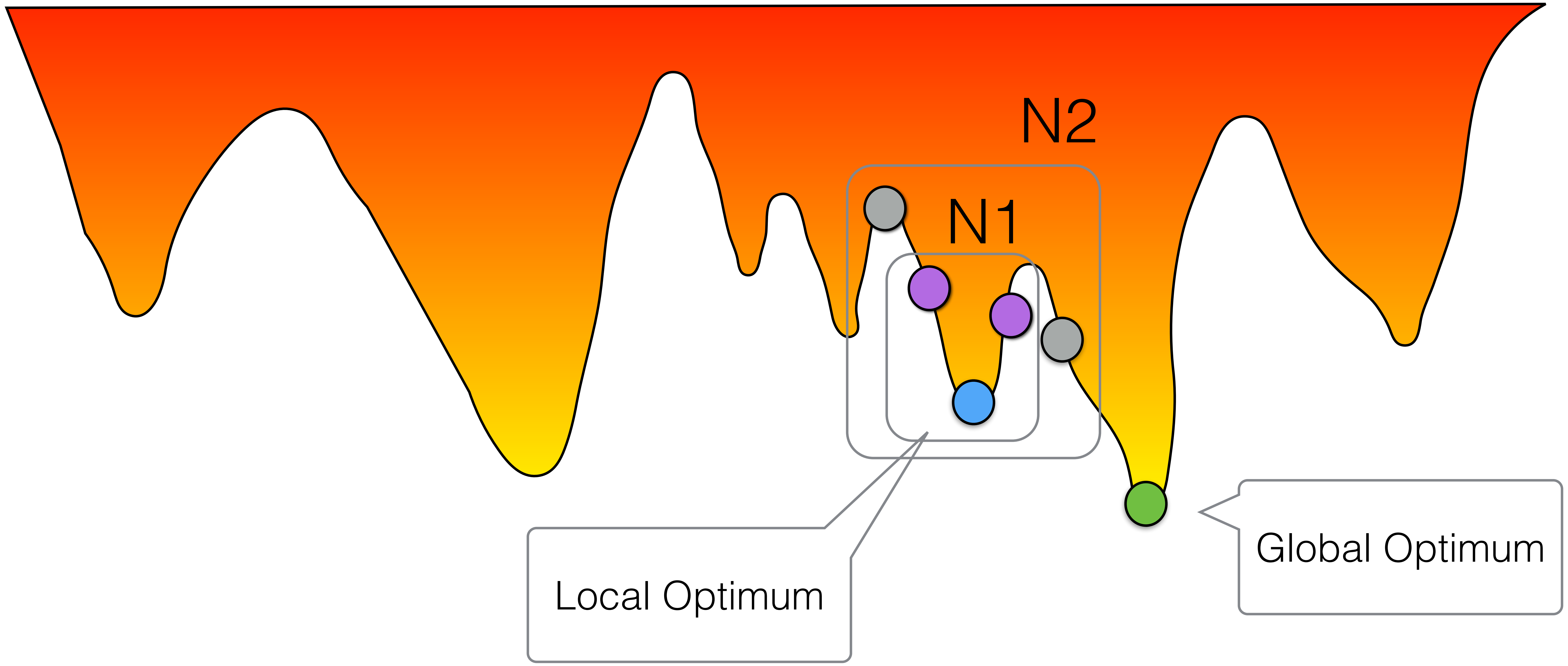
only accept an improving move

```
Procedure S-Best(N,s)
  $N^* := \{n \in N | f(n) = \min_{s \in N} f(s)\}$ ;
  return $n \in N^*$ with probability $1/|N^*|$;
```

Select one of the bests (ties broken randomly)

```
Procedure LocalSearch(f,N,L,S,s)
  $s^* := s$;
  for $k := 1$ to $MaxTrials$ do
      if $sastifiable(s) \land f(s) < f(s^*)$ then
          $s^* := s$ ;
      $s := S(L(N(s), s), s)$ ;
  return $s^*$;
```

The problem: Local Minima

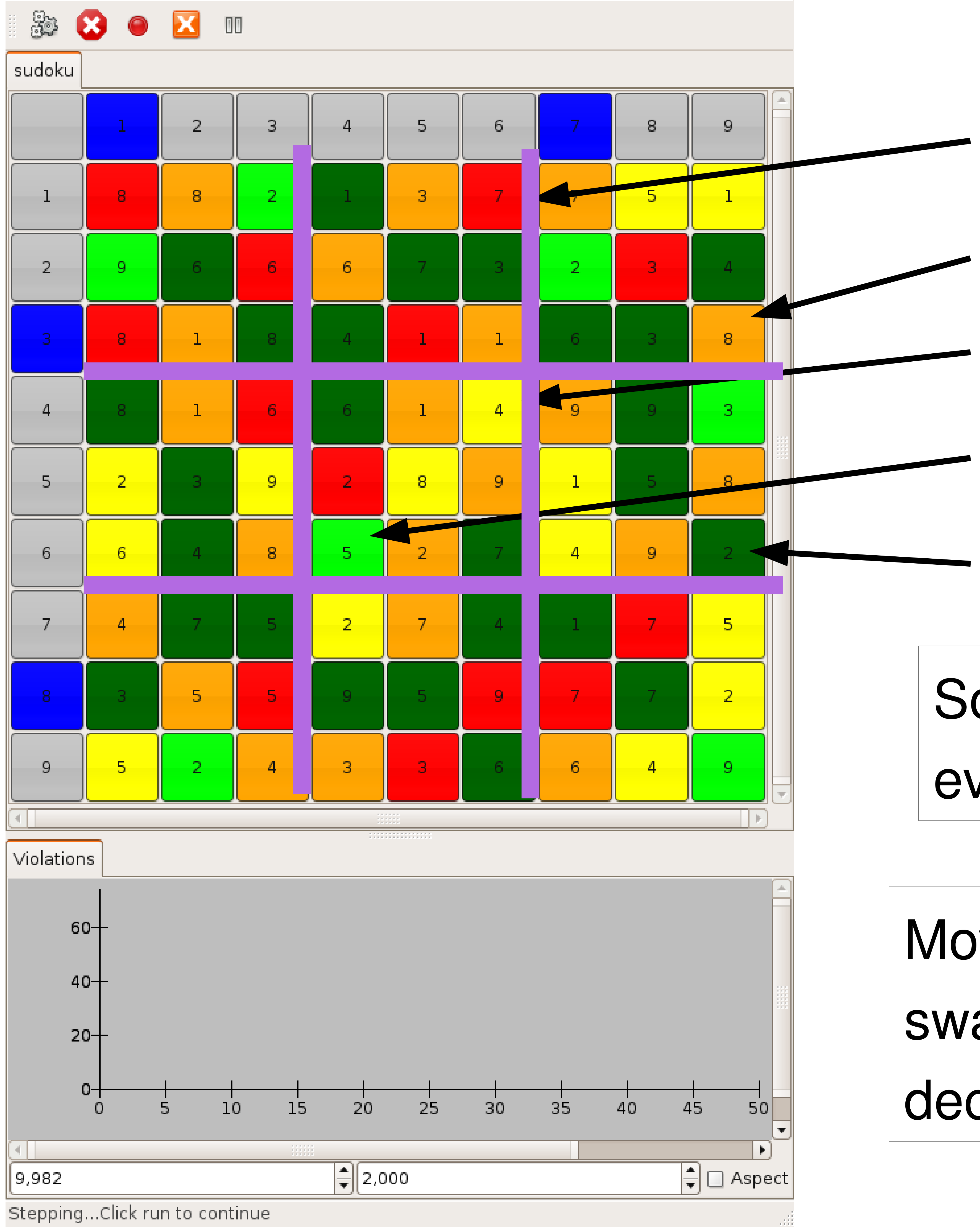


Two solutions

1. Accept to degrade the solution (meta-heuristics)
2. Enlarge the neighborhood

Sudoku

violation(cell) = number of cells in the same line/column/block with the same value



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

9,982
2,000
☐ Aspect
Stepping...Click run to continue

Cell with a violation >2

Cell with a violation =2

Cell with a violation =1

Cell with no violation

Given Cell

Solution Found when everything is Green

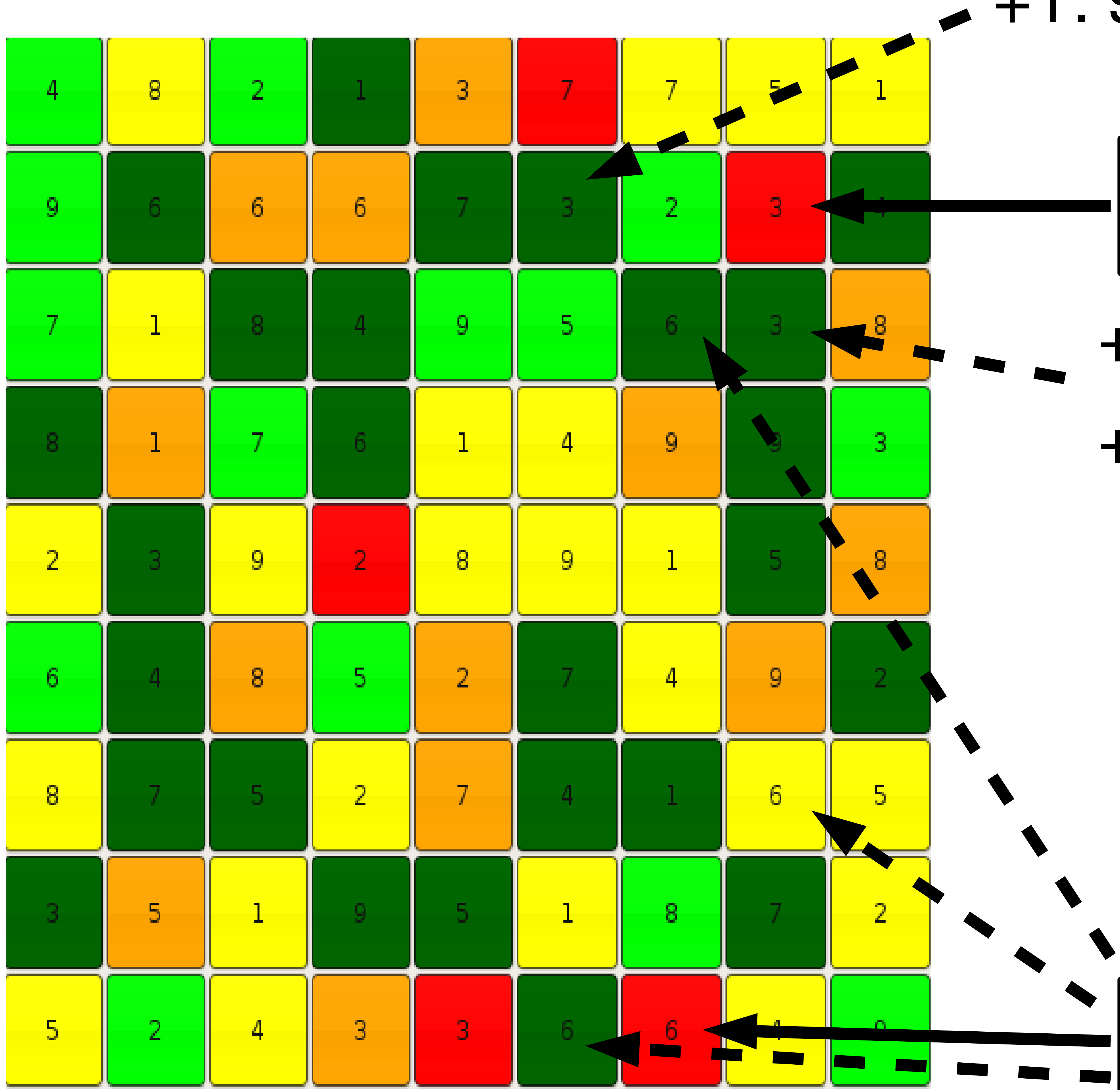
Moves:
swap of two cells to decrease global violation

Sudoku Hard and Soft Constraint

- Hard Constraints (cannot be violated)
  - Each number in {1..9} occurs 9x

- Soft Constraints (can be violated)
  - Each number in {1..9} occurs 1x in a row
  - Each number in {1..9} occurs 1x in a column
  - Each number in {1..9} occurs 1x in a 3x3 block

Sudoku: Computation of violation



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

+1: same line

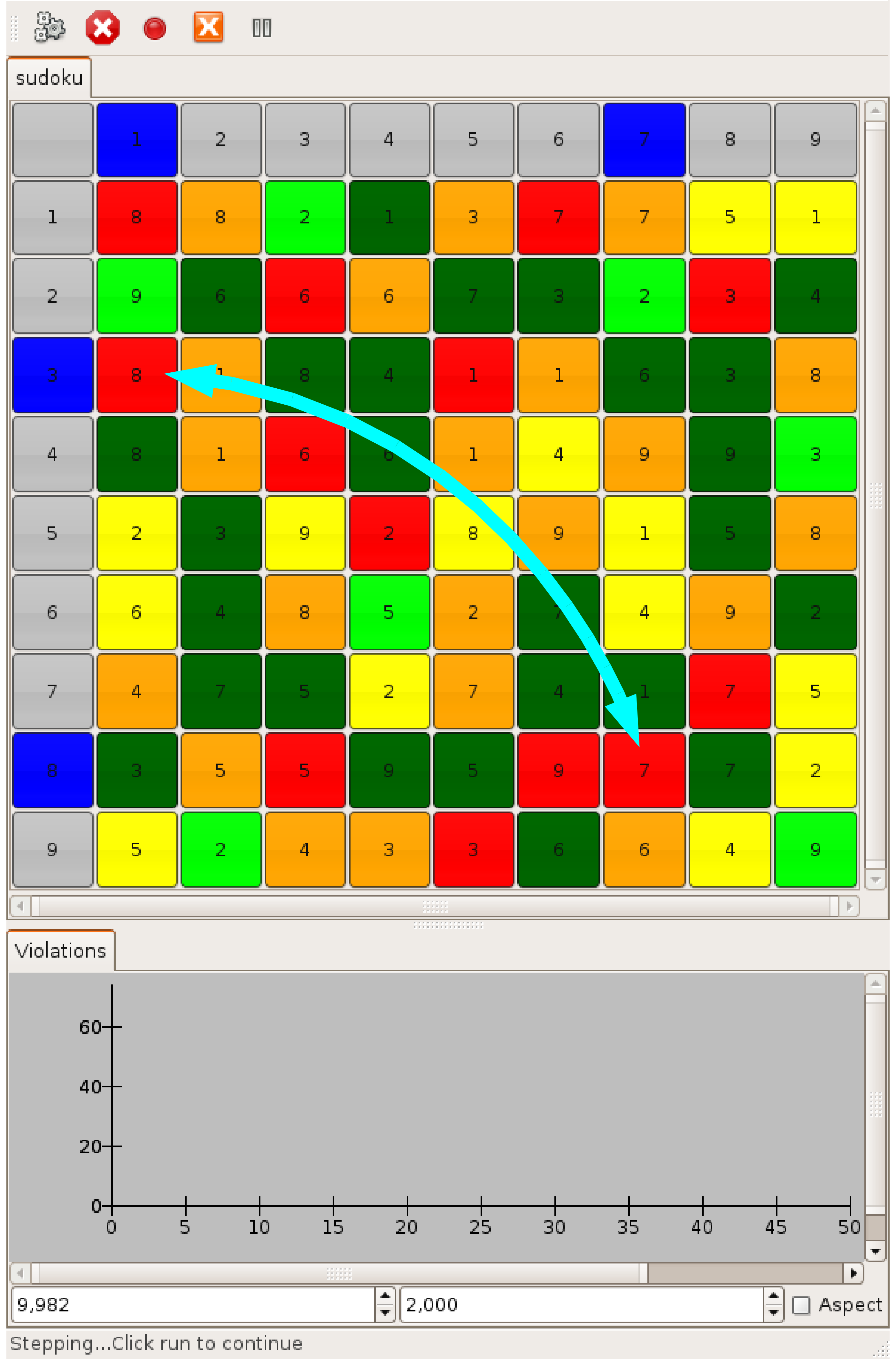
Violation 3

+1 same column

+1 same block

Violation 3

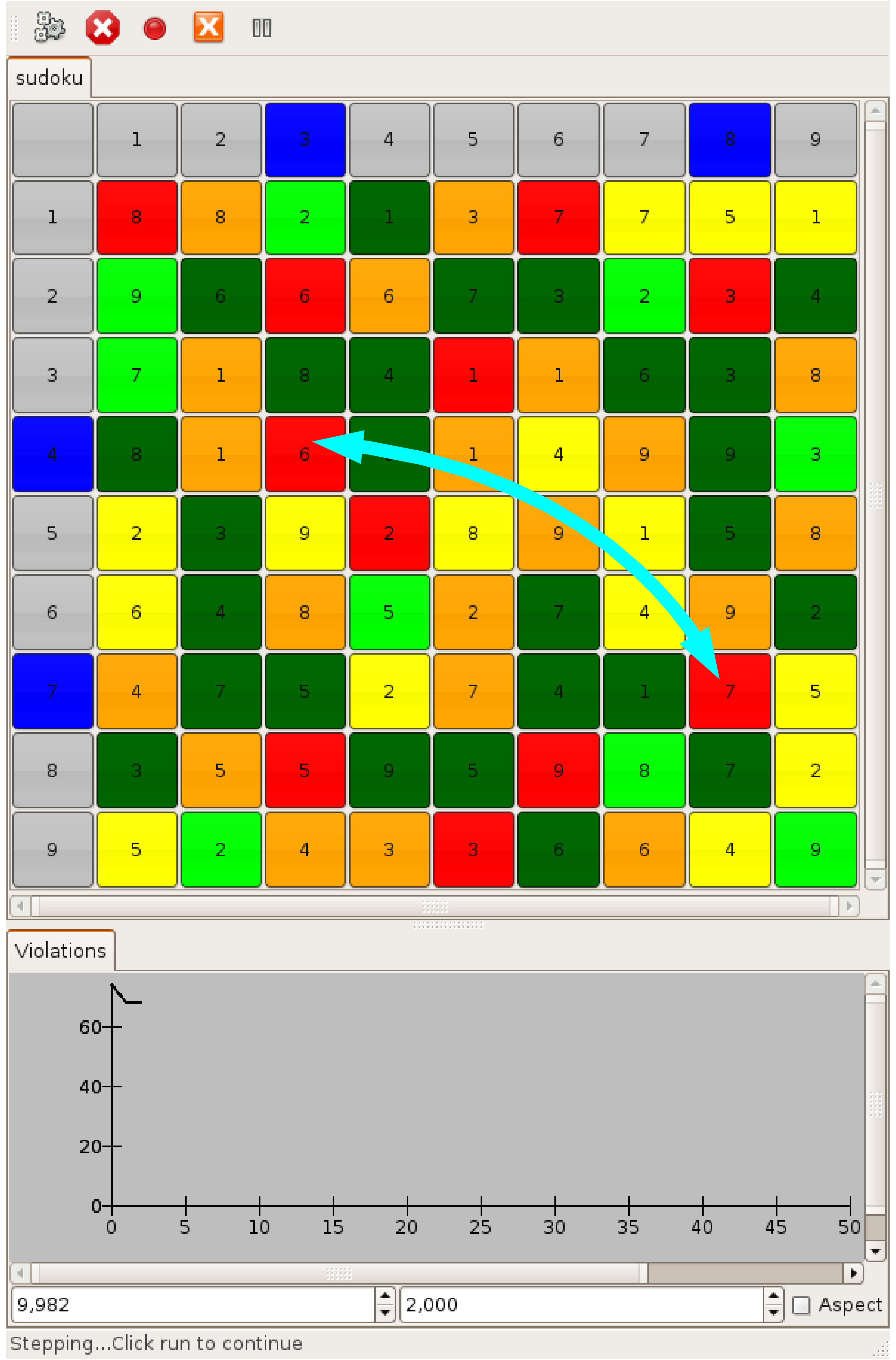
Sudoku: Swap Moves 1



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

9,982 | 2,000 | ☐ Aspect

Stepping...Click run to continue

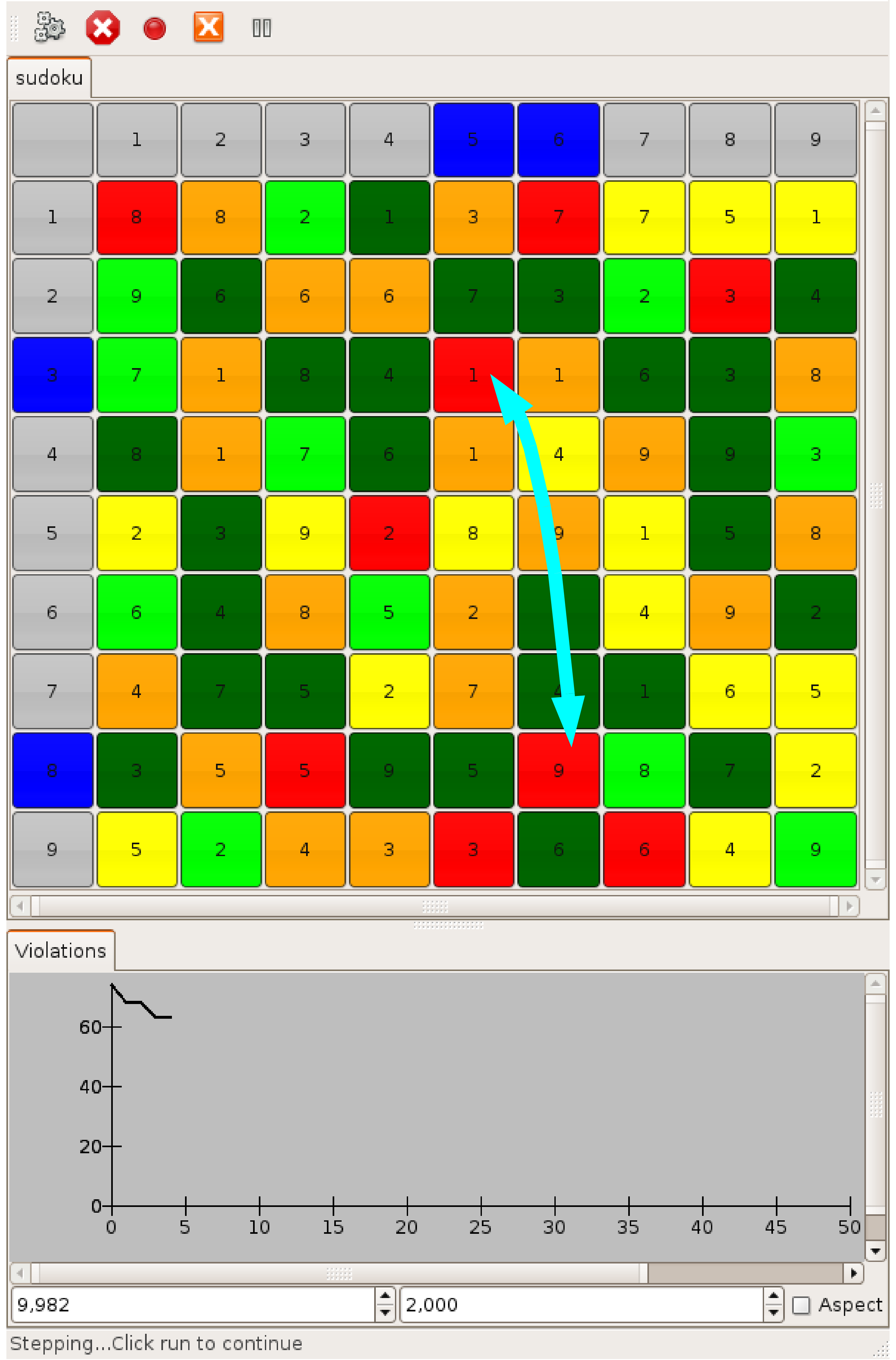


	1	2	3	4	5	6	7	8	9
1	8	8	2	1	3	7	7	5	1
2	9	6	6	6	7	3	2	3	4
3	7	1	8	4	1	1	6	3	8
4	8	1	6	6	1	4	9	9	3
5	2	3	9	2	8	9	1	5	8
6	6	4	8	5	2	7	4	9	2
7	4	7	5	2	7	4	1	7	5
8	3	5	5	9	5	9	8	7	2
9	5	2	4	3	3	6	6	4	9

9,982 | 2,000 | ☐ Aspect

Stepping...Click run to continue

Sudoku: Swap Moves 2



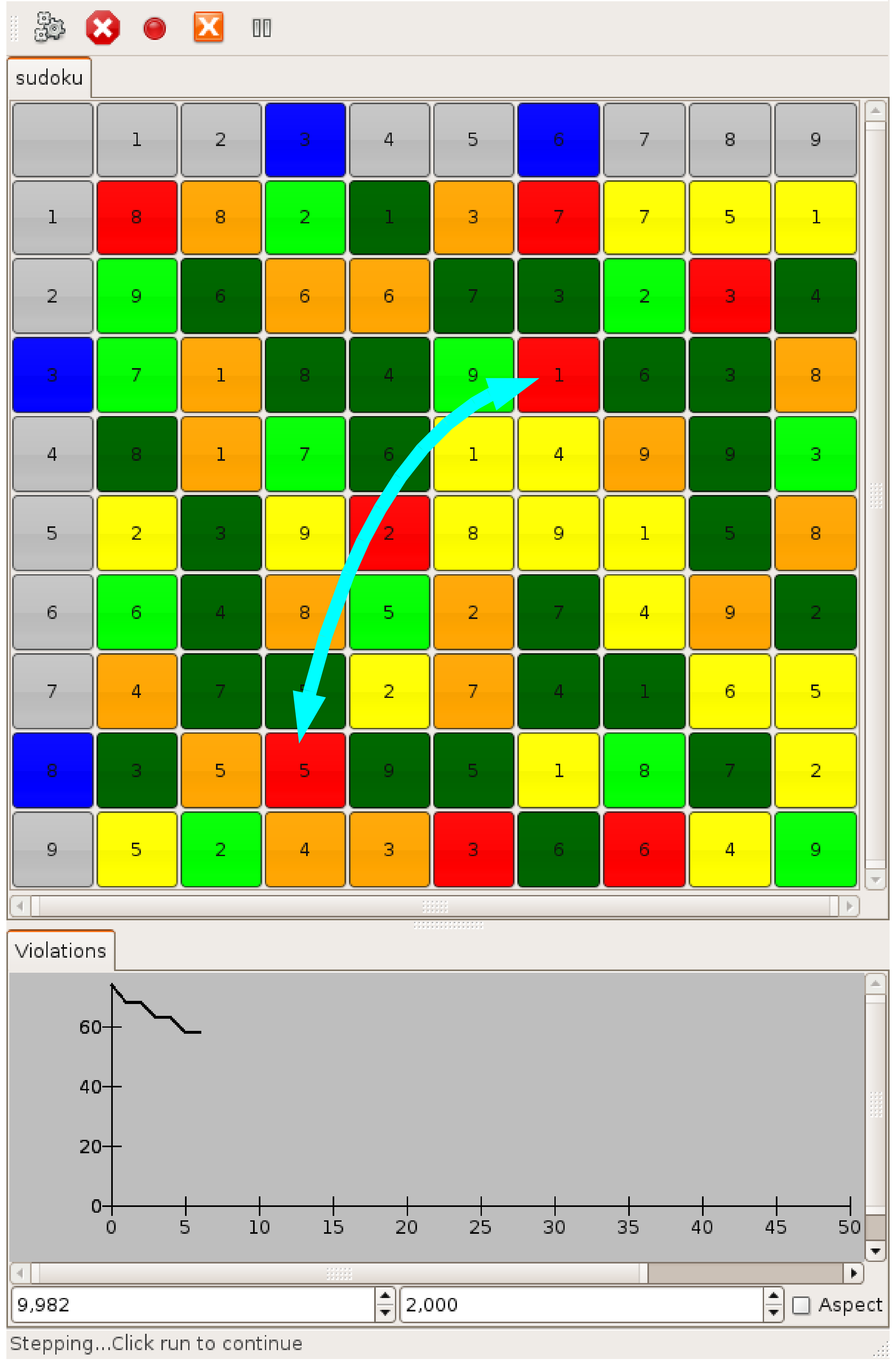
	1	2	3	4	5	6	7	8	9
1	8	8	2	1	3	7	7	5	1
2	9	6	6	6	7	3	2	3	4
3	7	1	8	4	1	1	6	3	8
4	8	1	7	6	1	4	9	9	3
5	2	3	9	2	8	9	1	5	8
6	6	4	8	5	2	[illegible]	4	9	2
7	4	7	5	2	7	4	1	6	5
8	3	5	5	9	5	9	8	7	2
9	5	2	4	3	3	6	6	4	9

9,982

2,000

☐ Aspect

Stepping...Click run to continue



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

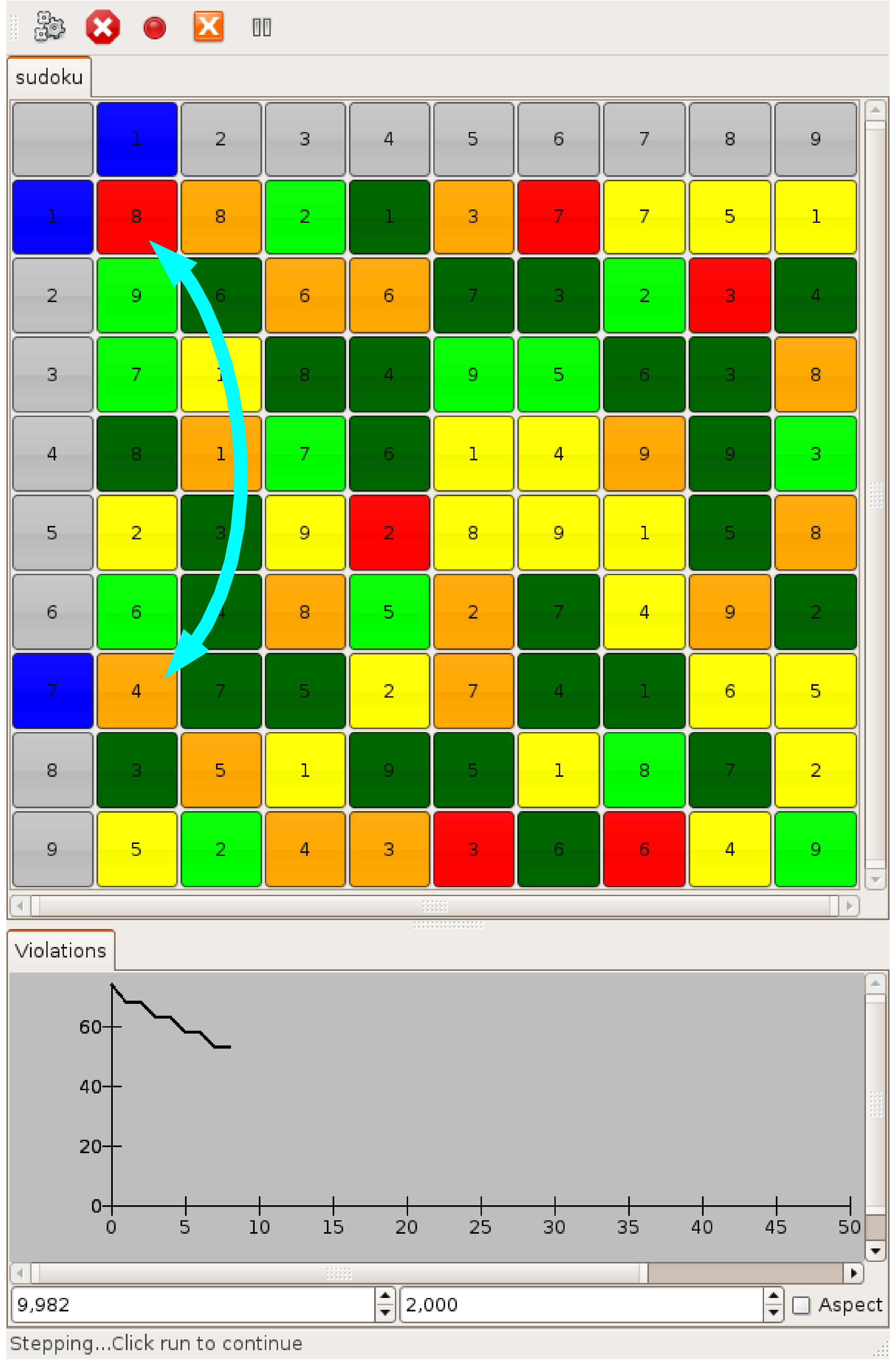
9,982

2,000

☐ Aspect

Stepping...Click run to continue

Sudoku: Swap Moves 3



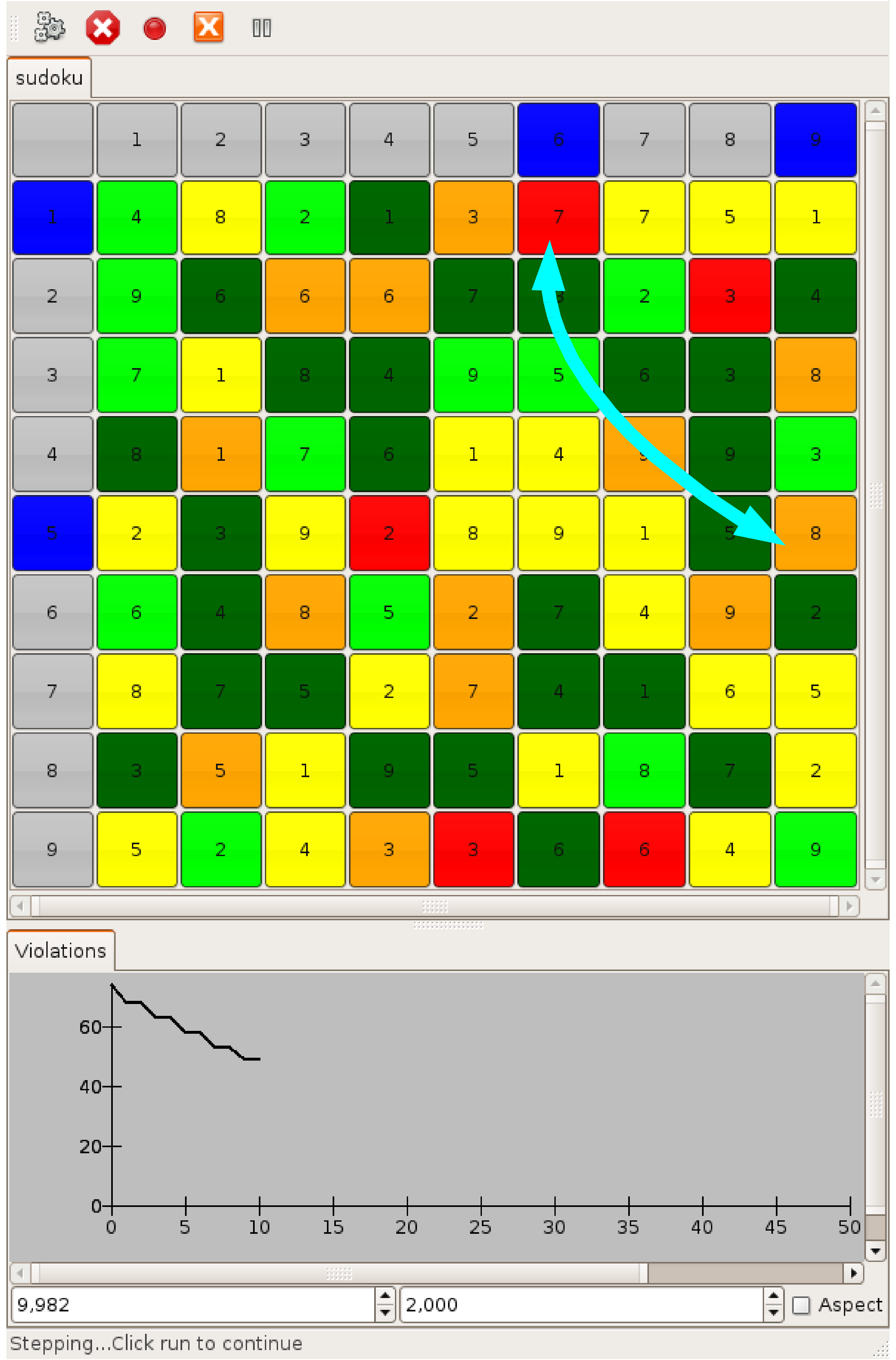
sudoku

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

Violations

9,982 2,000 ☐ Aspect

Stepping...Click run to continue



sudoku

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

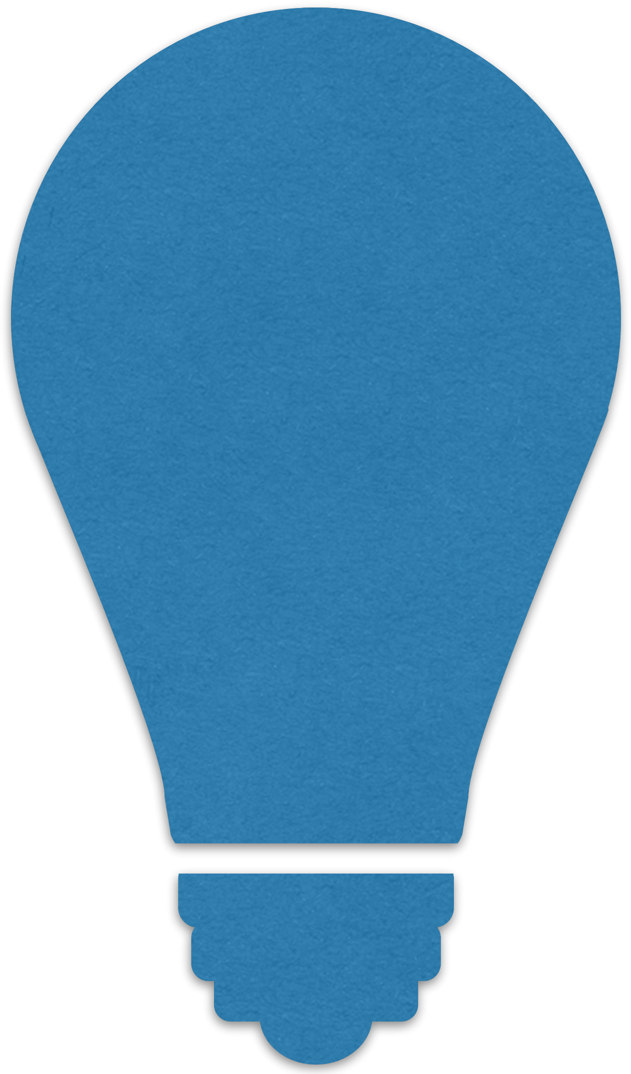
Violations

9,982 2,000 ☐ Aspect

Stepping...Click run to continue

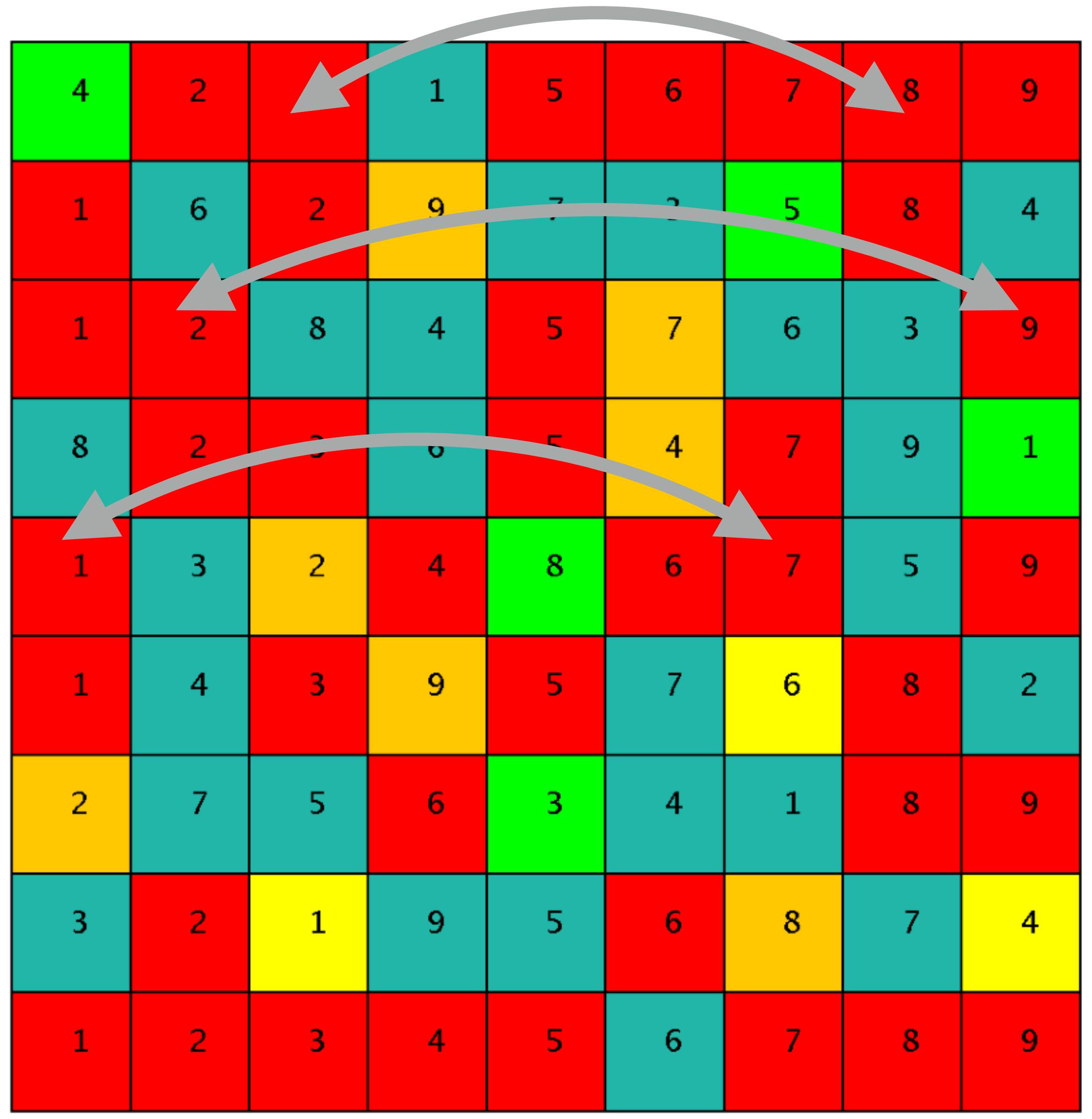
Idea 1

- Remain “more feasible” than having the right number of occurrences of each numbers:
  - Some soft constraints will be come hard



Sudoku Hard and Soft Constraint

- Hard Constraints (cannot be violated)
  - Each number in {1..9} occurs 9x
  - Each number in {1..9} occurs 1x in a row
- Soft Constraints (can be violated)
  - Each number in {1..9} occurs 1x in a block
  - Each number in {1..9} occurs 1x in a column



4	2	[illegible]	1	5	6	7	8	9
1	6	2	9	7	3	5	8	4
1	2	8	4	5	7	6	3	9
8	2	3	6	5	4	7	9	1
1	3	2	4	8	6	7	5	9
1	4	3	9	5	7	6	8	2
2	7	5	6	3	4	1	8	9
3	2	1	9	5	6	8	7	4
1	2	3	4	5	6	7	8	9

Hard constraints are enforced by:
1. initialization
2. moves = swaps of two cells on a same row

Implementation

- Using a Constrained Based Local Search (Library)

- Facilitates the development of Local Search algorithms
  - you can add your constraints and objective functions
  - it will compute for you the violations
  - it will compute for you the delta’s (what if I exchange the values of two variables)
  - …

- You can focus on the interesting part: moves and meta-heuristics. We will come back to that later, let’s first understand the magic

CBLS Sudoku Model

```
int[] instance1 = new int[]{
        0, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 6, 0, 0, 7, 3, 0, 0, 4,
        0, 0, 8, 4, 0, 0, 6, 3, 0,
        8, 0, 0, 6, 0, 0, 0, 9, 0,
        0, 3, 0, 0, 0, 0, 0, 5, 0,
        0, 4, 0, 0, 0, 7, 0, 0, 2,
        0, 7, 5, 0, 0, 4, 1, 0, 0,
        3, 0, 0, 9, 5, 0, 0, 7, 0,
        0, 0, 0, 0, 0, 6, 0, 0, 0};

IntVarLS [] grid;
IntVarLS violation;
ConstraintSystem constraintSystem;
ArrayList<Pair> possibleSwaps;

SolverLS ls = makeSolver();


grid = makeIntVarArray(9*9, i -> makeIntVar(ls,init[i]));


ArrayList<Constraint> constraints = new ArrayList<>();
for (int k = 0; k < 9; k++) {
    final int i = k;
    Constraint allDiffCol = new AllDifferent(makeIntVarArray(n, j -> grid[j * 9 + i]));
    Constraint allDiffBlock = new AllDifferent(makeIntVarArray(n, j -> grid[blocks.get(i).get(j)]));
    constraints.add(allDiffBlock);
    constraints.add(allDiffCol);
}

constraintSystem = new ConstraintSystem(constraints.toArray(new
violation = constraintSystem.violation();
```

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

decision variables: value in each cell

Constraint for rows and blocks

An object responsible to compute the total violation

CBLS Sudoku Model

```
// swap two cells on the same line
possibleSwaps = new ArrayList<>();
for (int l = 0; l < 9; l++) {
    for (int i = 0; i < 9; i++) {
        for (int j = i+1; j < 9; j++) {
            int v1 = l*9+i;
            int v2 = l*9+j;
            if (problem[v1] == 0 && problem[v2] == 0) {
                possibleSwaps.add(new Pair(v1,v2));
            }
        }
    }
}
public int swapDelta(int a, int b) {
    int before = violation.value();
    swap(a,b);
    int after = violation.value();
    swap(a,b);
    return after-before;
}

public void swap(int a, int b) {
    int va = grid[a].value();
    int vb = grid[b].value();
    grid[a].setValue(vb);
    grid[b].setValue(va);
}
```

Pre-compute all the possible swap position not involving hint position

Compute the delta if exchanging values in position a and b

Exchange values in position a and b

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

CBLS Sudoku Greedy Search

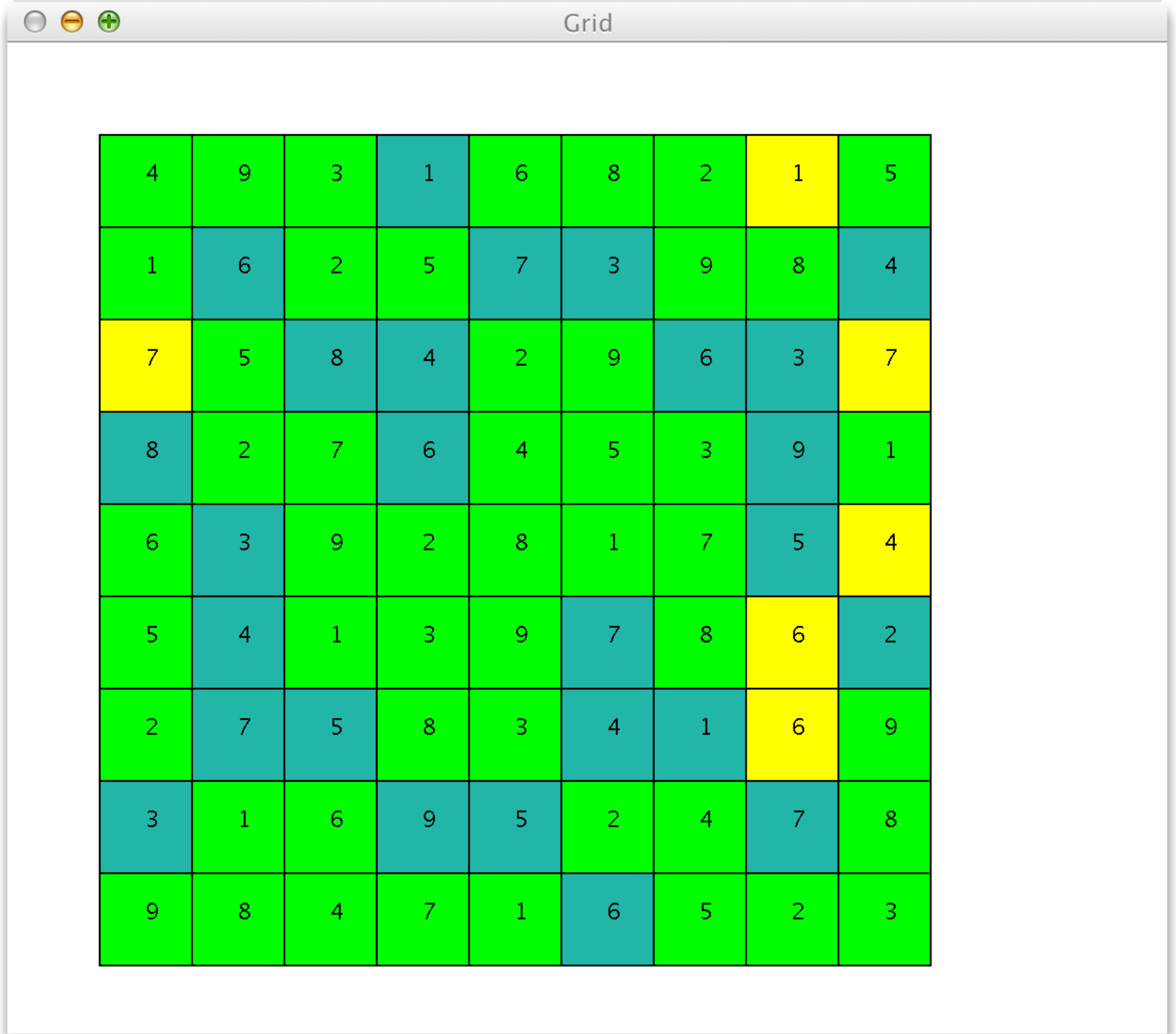
```
public void solve() {
    int iter = 1;
    while (constraintSystem.violation().value() > 0){
        iterationGreedy(iter++);
    }
}
public void iterationGreedy(int iter) {
    Pair bestSwap = bestSwap();
    grid[bestSwap.a].swap(grid[bestSwap.b]);
    notifyAllObservers(iter,bestSwap);
}

public Pair bestSwap() {
    Pair bestSwap = null;
    int bestDelta = Integer.MAX_VALUE;
    for (Pair p : possibleSwaps) {
        int delta = violation.getSwapDelta(grid[p.a],grid[p.b]);
        if (delta < bestDelta) {
            bestDelta = delta;
            bestSwap = p;
        }
    }
    return bestSwap;
}
```

Solve until feasible solution found (zero violation)

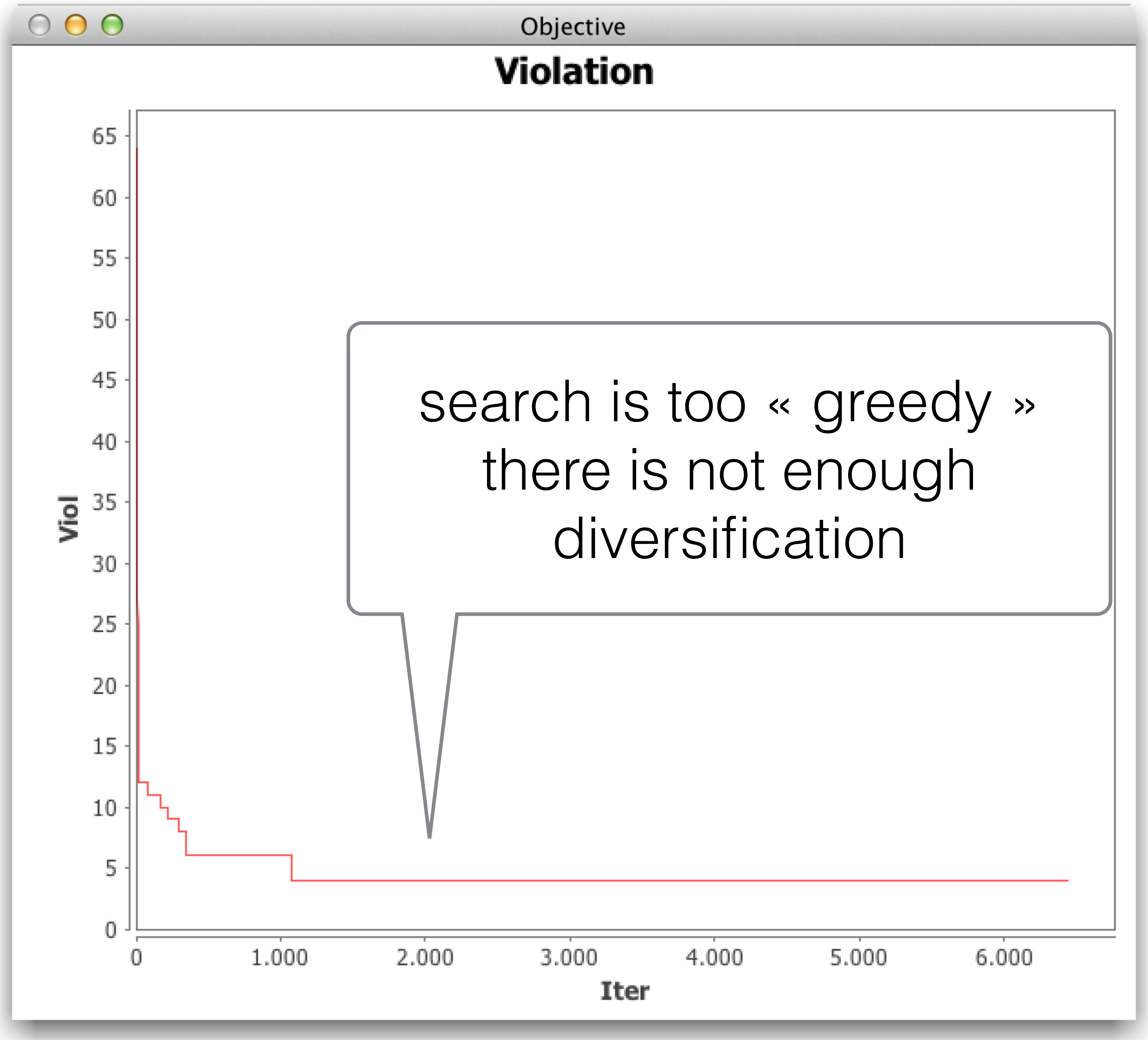
One iteration

Find the best exchange

Problem if too greedy



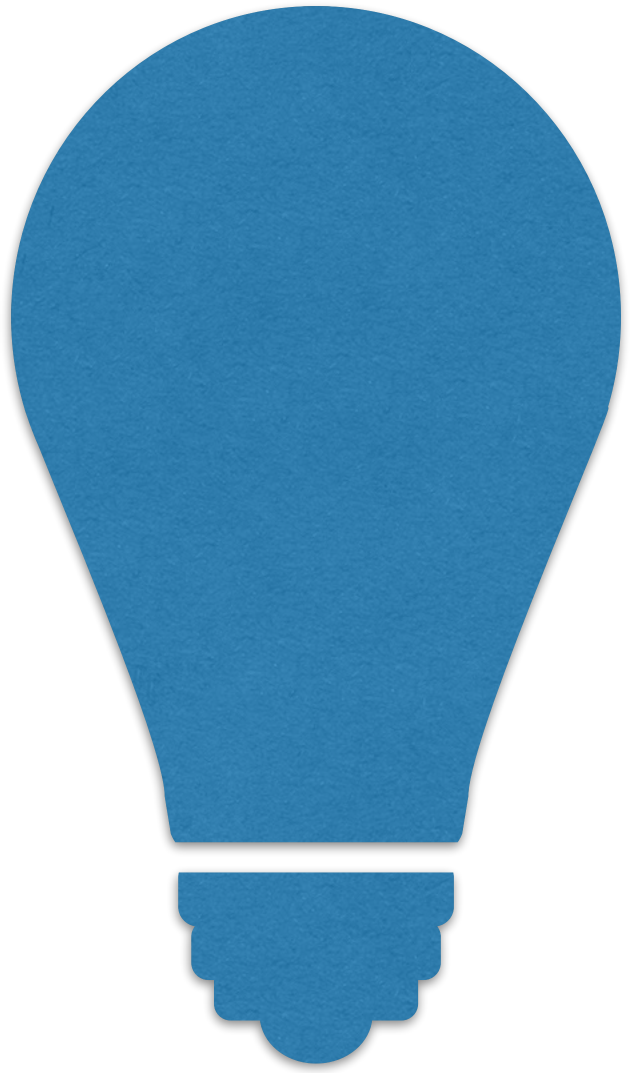
4	9	3	1	6	8	2	1	5
1	6	2	5	7	3	9	8	4
7	5	8	4	2	9	6	3	7
8	2	7	6	4	5	3	9	1
6	3	9	2	8	1	7	5	4
5	4	1	3	9	7	8	6	2
2	7	5	8	3	4	1	6	9
3	1	6	9	5	2	4	7	8
9	8	4	7	1	6	5	2	3



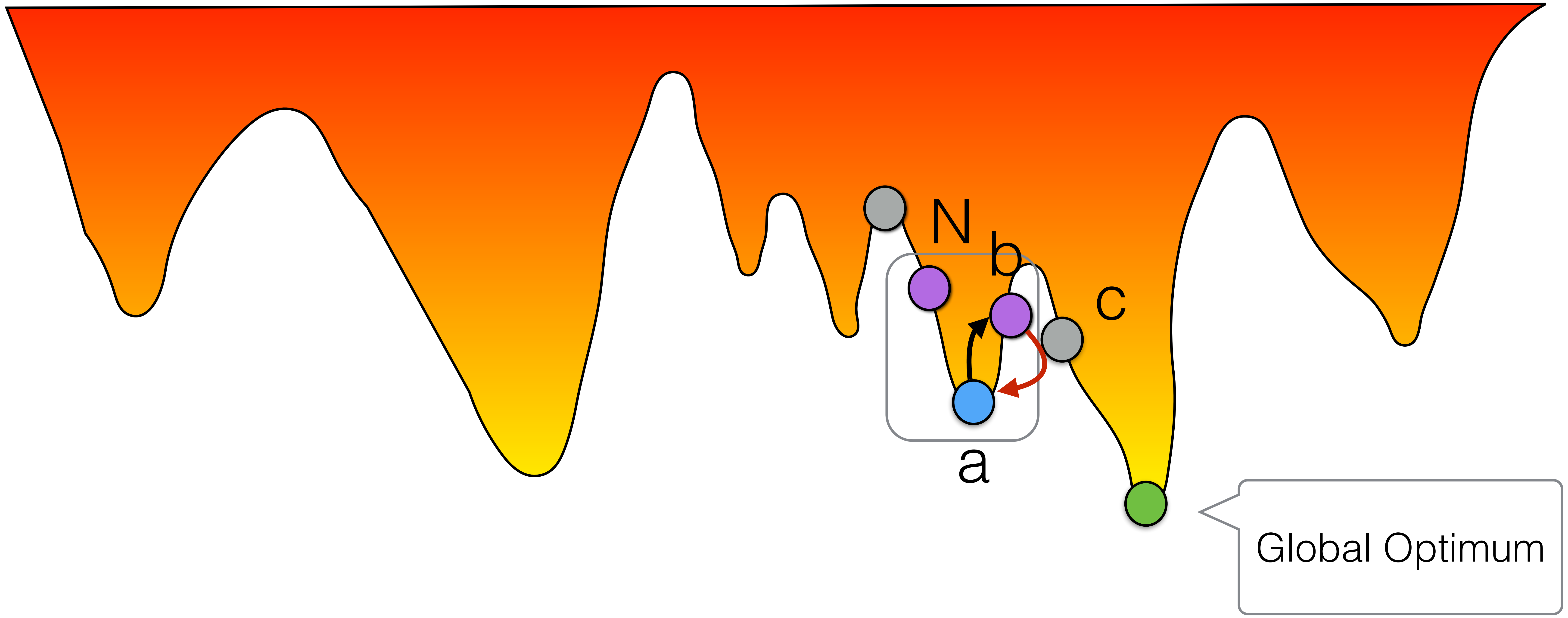
search is too « greedy »
there is not enough
diversification

Idea 2

- Avoid direct and short cycles in order to diversify and escape from local minima



Tabu Meta Heuristic



- From **a** you move to **b** (no better choice in neighboring N)
- But **the reverse move becomes tabu**, you don't want to come to **a** for a while (duration = tabu tenure).

CBLS Sudoku Search with Tabu

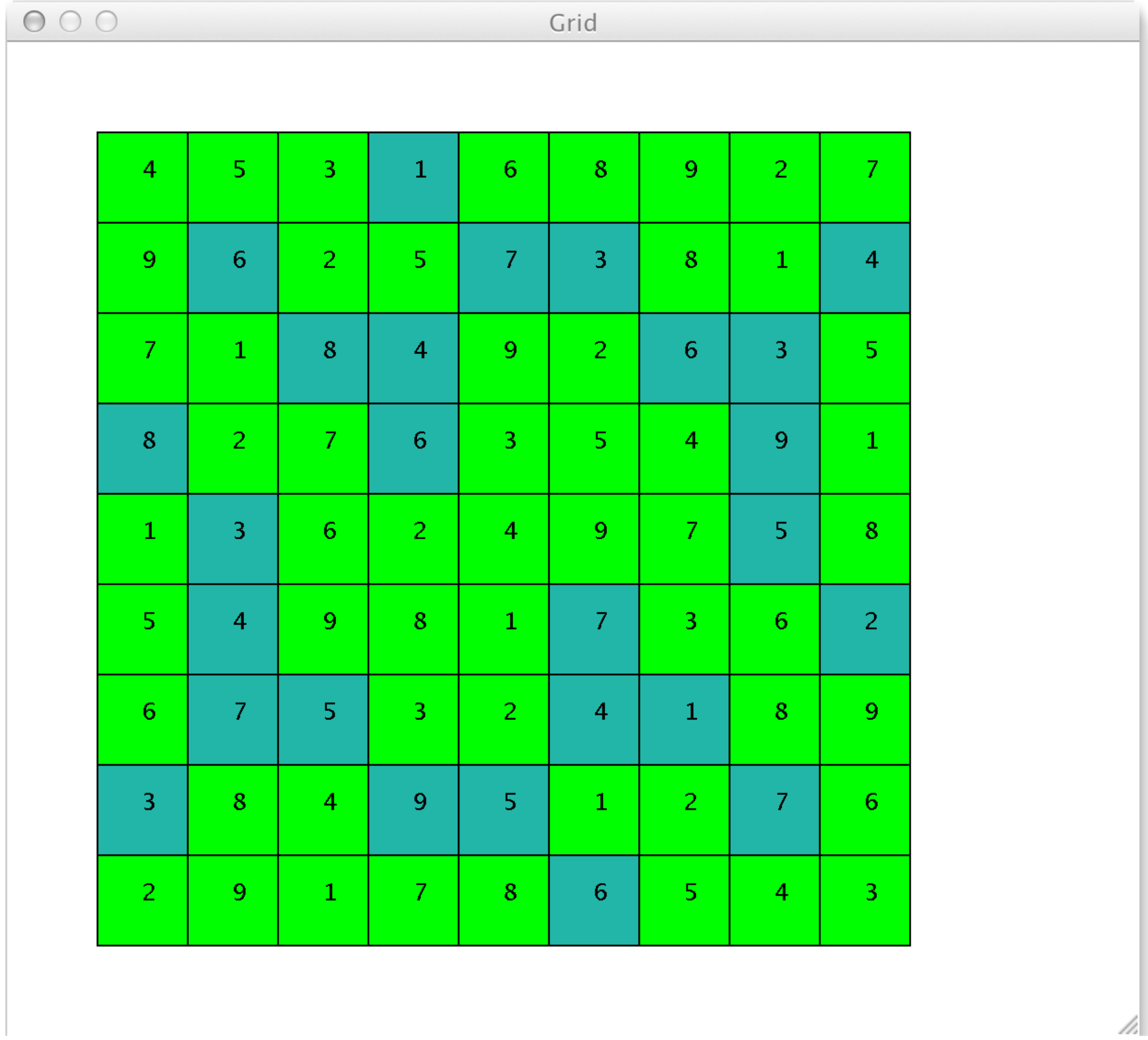
```
public void solve() {
    int iter = 1;
    int tabu = 20;
    while (constraintSystem.violation().value() > 0){
        iterationTabu(iter++,tabu);
        return;
    }
}

public void iterationTabu(int iter, int tabu) {
    Pair bestSwap = bestSwapNonTabu(iter);
    grid[bestSwap.a].swap(grid[bestSwap.b]);
    bestSwap.iter = iter + rand.nextInt(tabu);
    notifyAllObservers(iter,bestSwap);
}

public Pair bestSwapNonTabu(int iter) {
    Pair bestSwap = null;
    int bestDelta = Integer.MAX_VALUE;
    for (Pair p : possibleSwaps) {
        if (p.iter < iter) {
            int delta = violation.getSwapDelta(grid[p.a],grid[p.b]);
            if (delta < bestDelta) {
                bestDelta = delta;
                bestSwap = p;
            }
        }
    }
    return  bestSwap;
}
```

Solve until feasible solution found (zero violation)

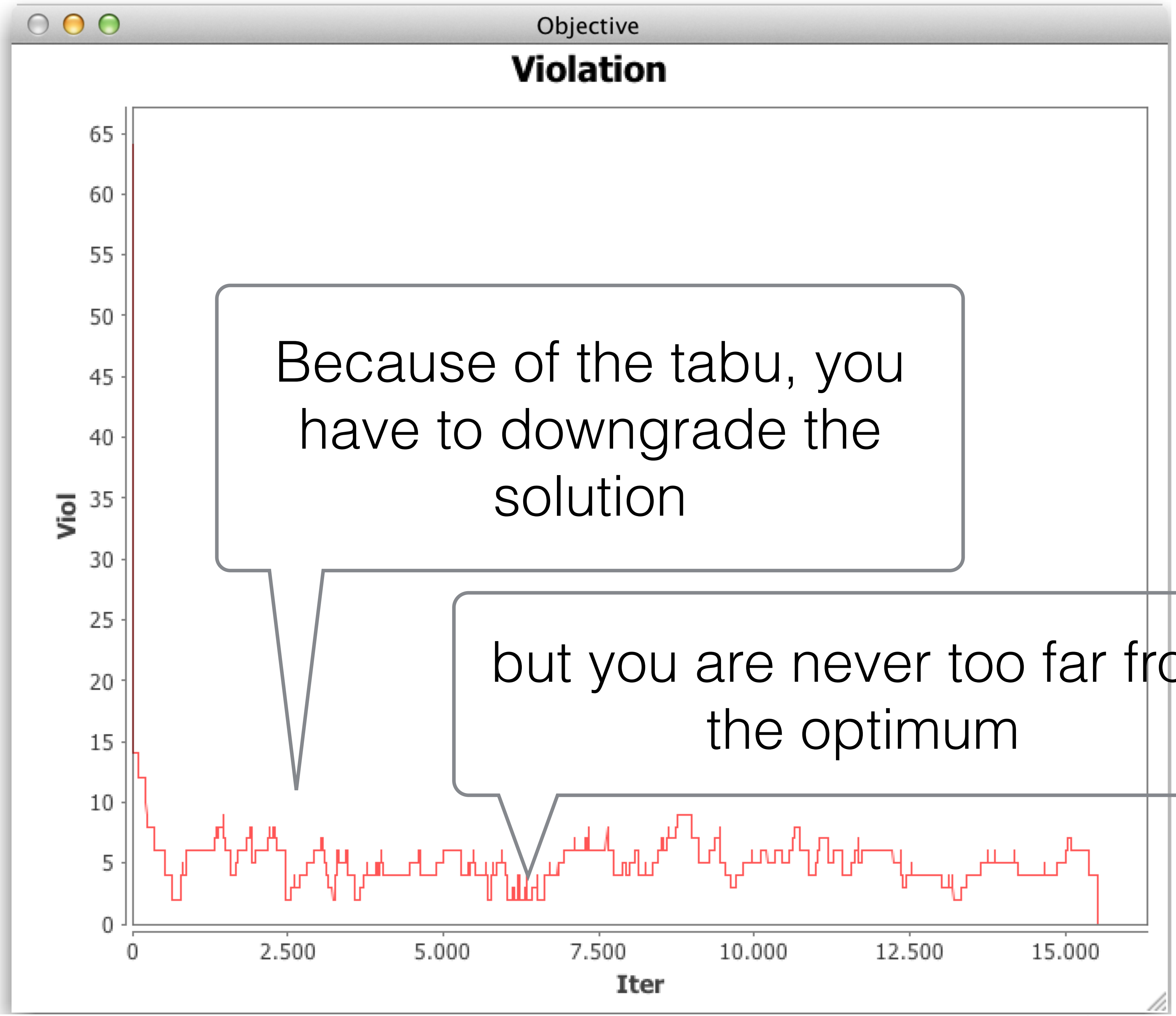
One iteration: bestSwap and set this swap tabu for a random number of iterations

Find the best exchange that is not tabu

Sudoku with Tabu Search



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



Because of the tabu, you have to downgrade the solution

but you are never too far from the optimum

Able to reach 0 violation.
Tabu offers a good Diversification/
Intensification tradeoff

Other possible improvements

- Aspiration
  - Maintain the current best violation
  - A move is not considered tabu if it can lead to the best so far objective violation
- Restart
  - Every X iterations, introduce random perturbations (for instance random swap moves)

Connected-Neighborhood

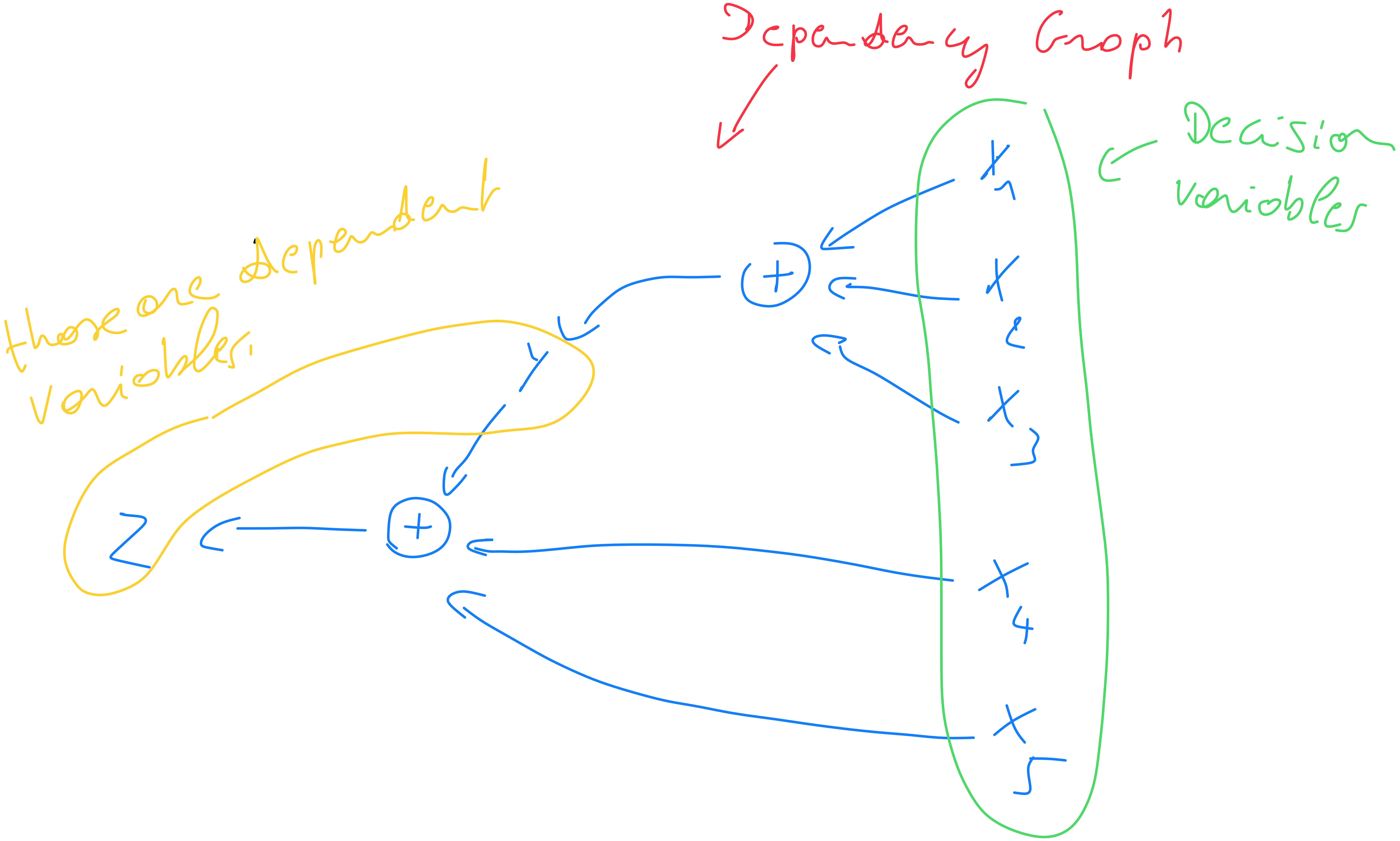
- A neighborhood is connected if and only if for each solution s, there exists a path to an optimal solution s*.

- Two advantages:
  - You don’t necessarily need a restarting strategy
  - Randomized heuristics where there is a non zero probability of accepting a neighbor k ∈ N(s) for each solution s, may be guaranteed to reach a global optimum (example: simulated annealing).

- To prove a neighborhood is connected, you must provide an algorithm to transform any solution s1 into a solution s2 by selecting the moves allowed by the neighborhood.

- Q: Is the swap move for sudoku a connected neighborhood? Why ?

CBLS Solver

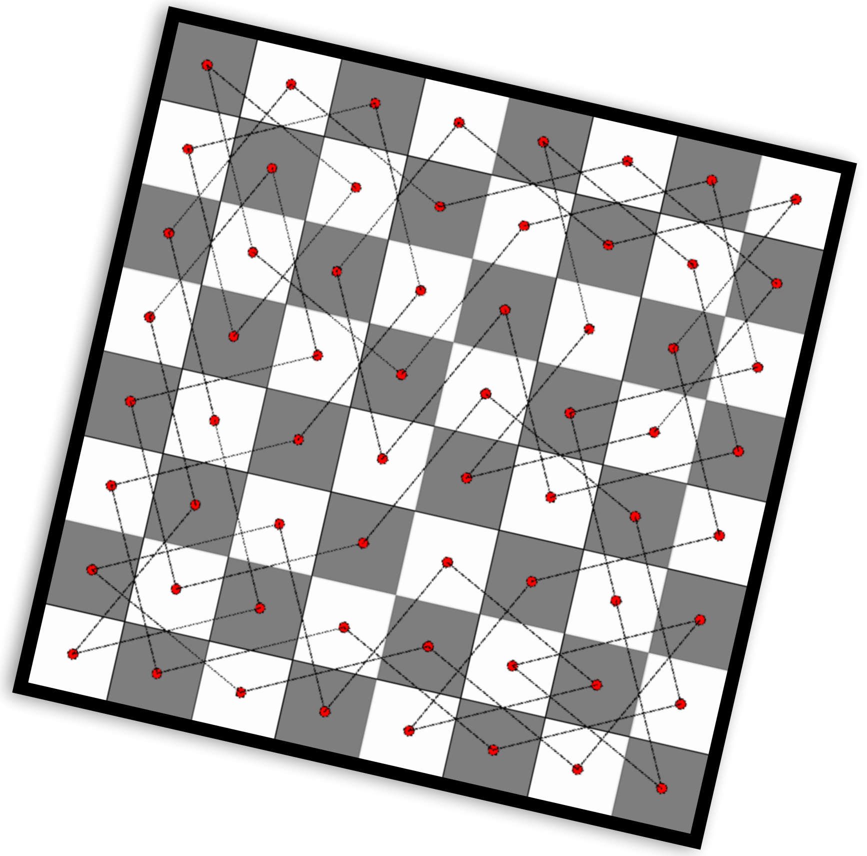

Dependency Graph
Decision variables
these are dependent variables.
x_1
x_2
x_3
x_4
x_5
+
+
y
z

Advanced Algorithms for Optimization

Local Search

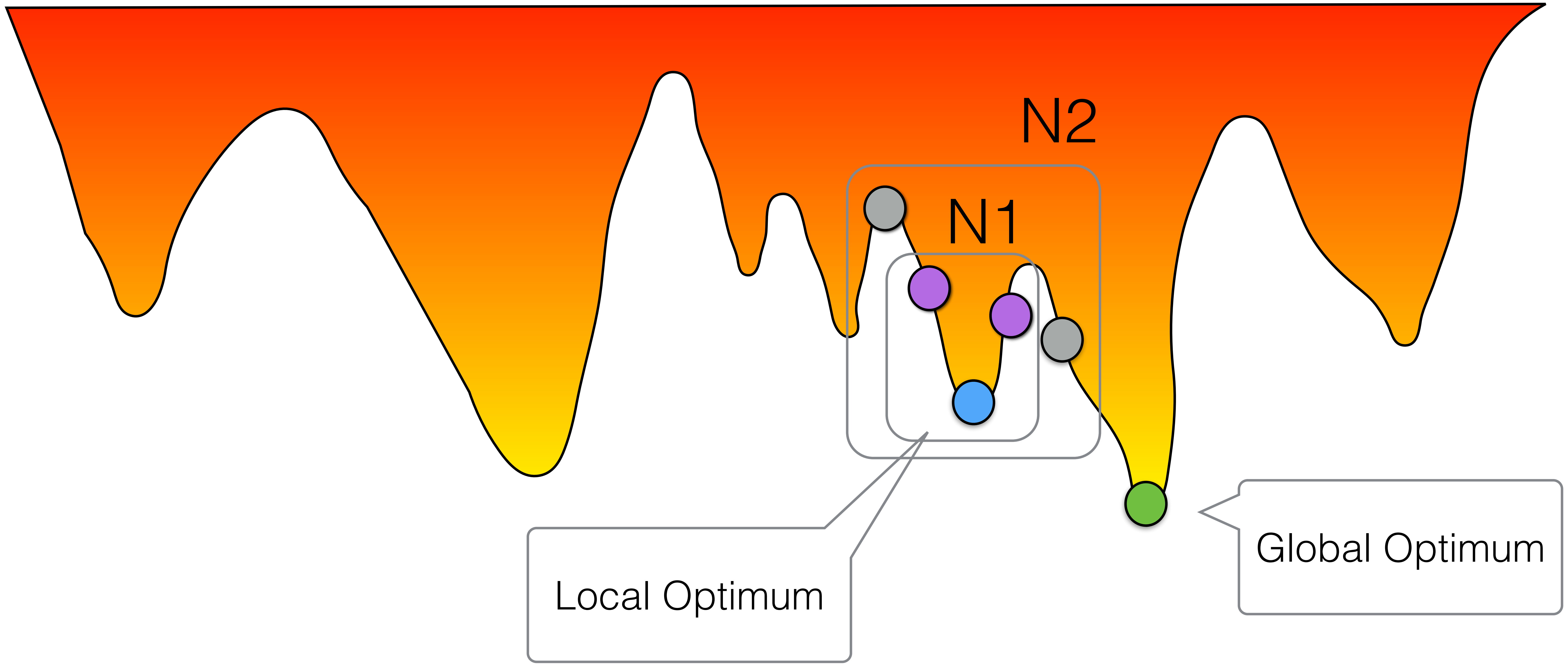
Part2

Pierre Schaus



https://github.com/pschaus/linfo2266

The problem: Local Minima

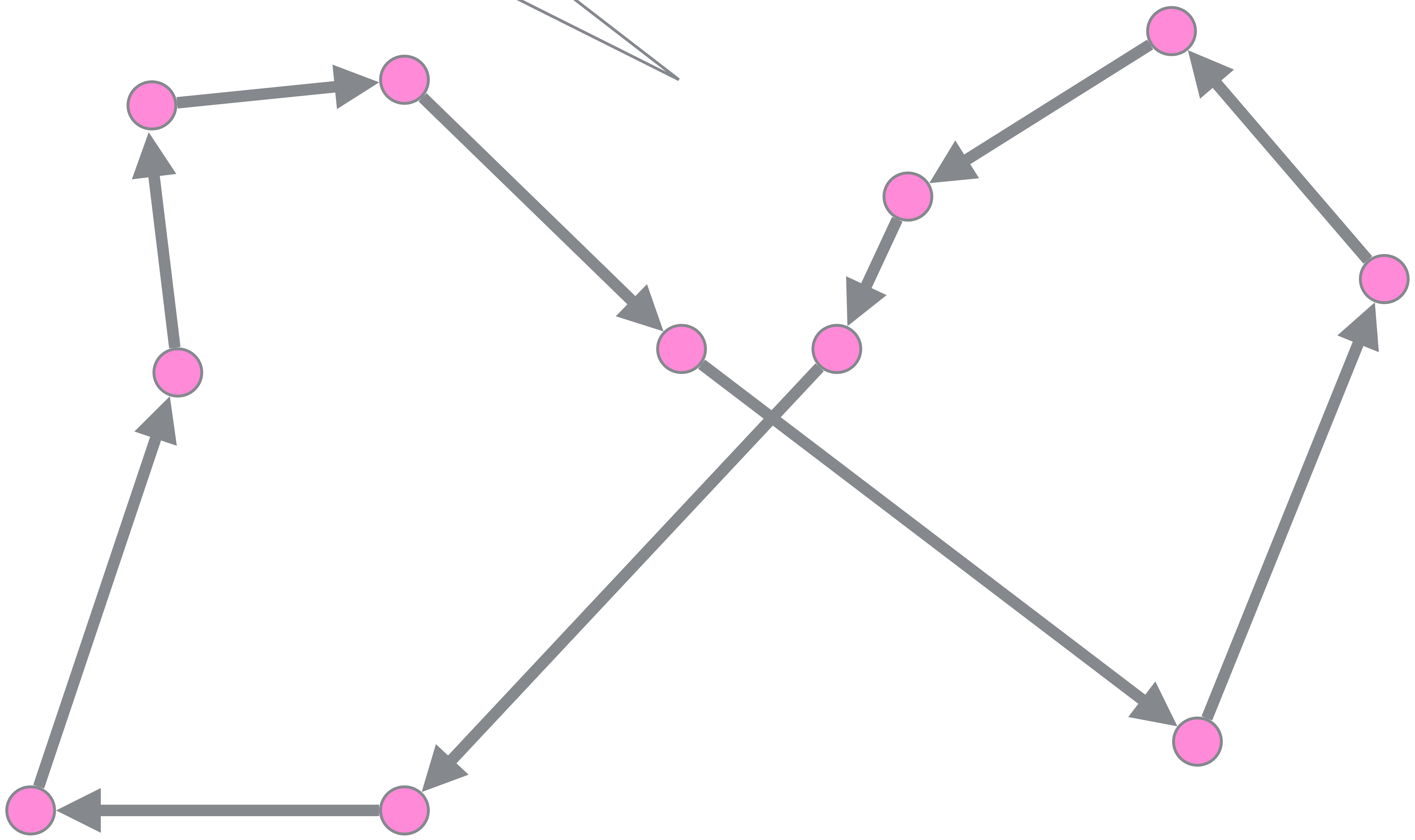


Two solutions

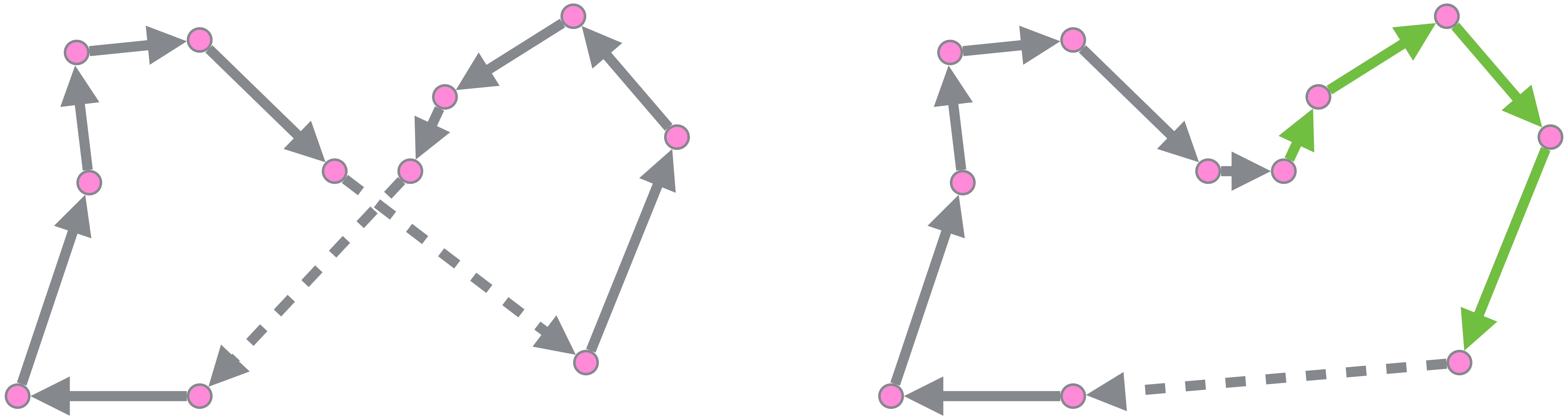
1. Accept to degrade the solution (meta-heuristics)
2. Enlarge the neighborhood

TSP Move

Can you improve this tour ?



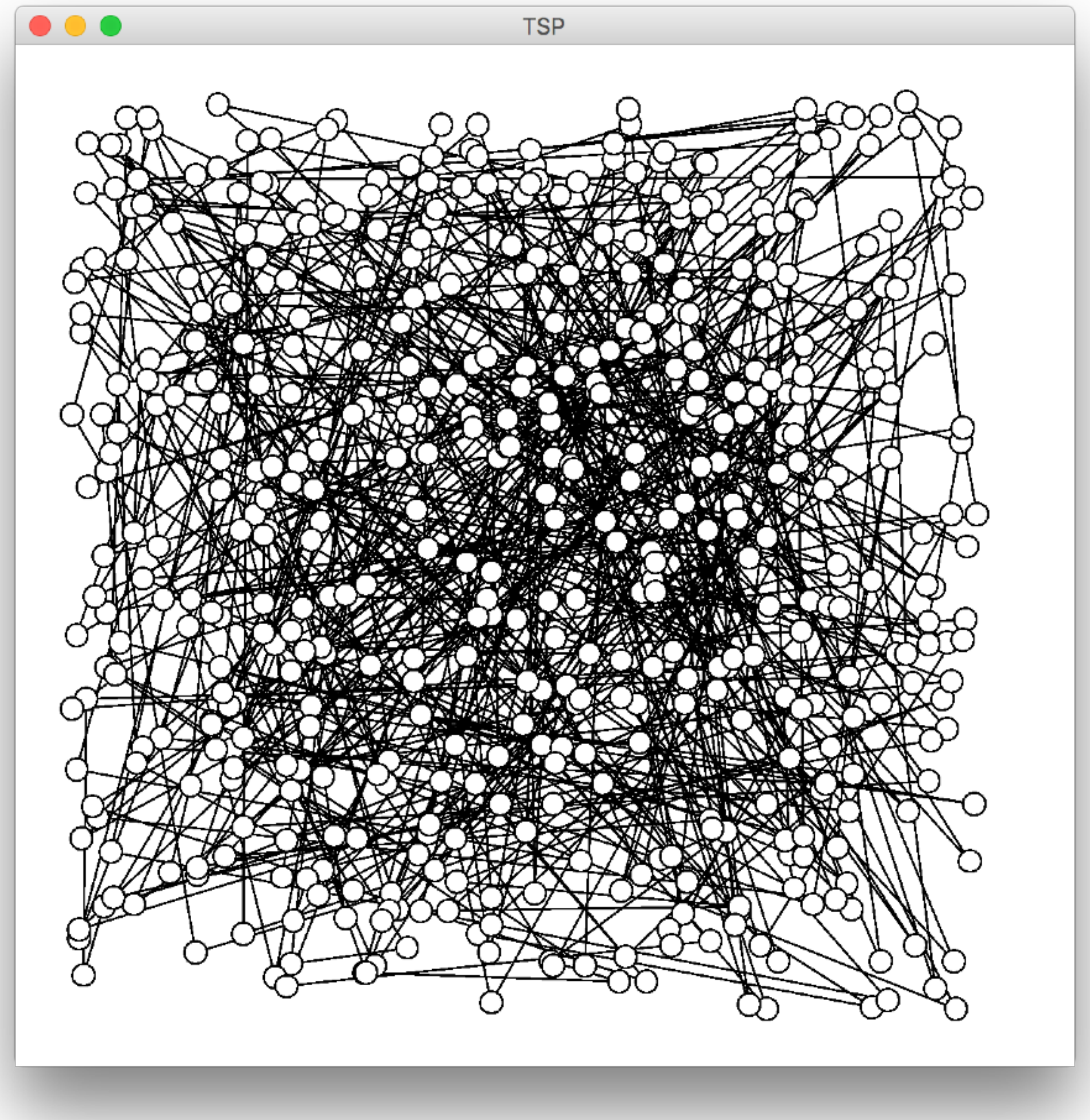
2 Opt

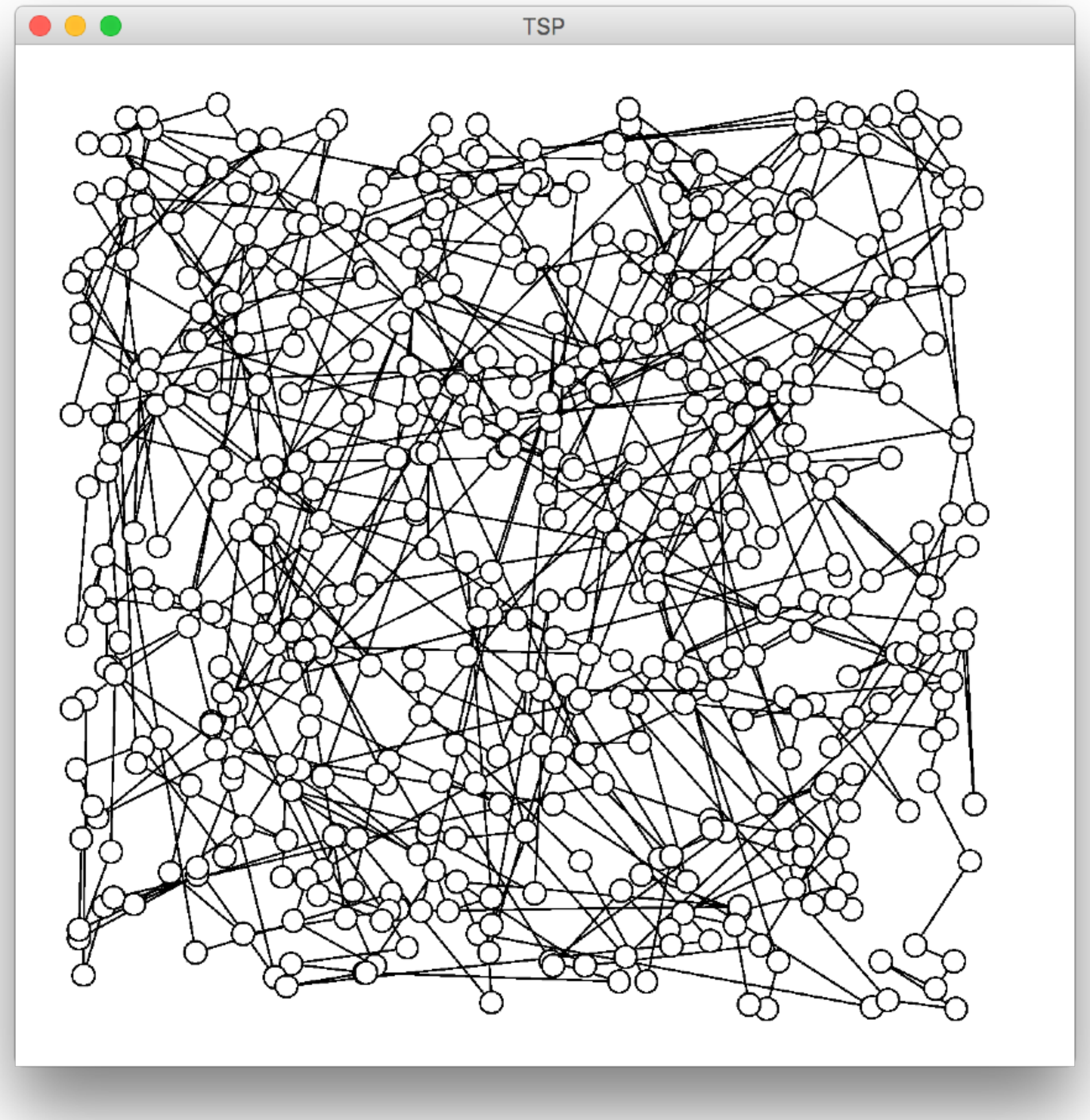


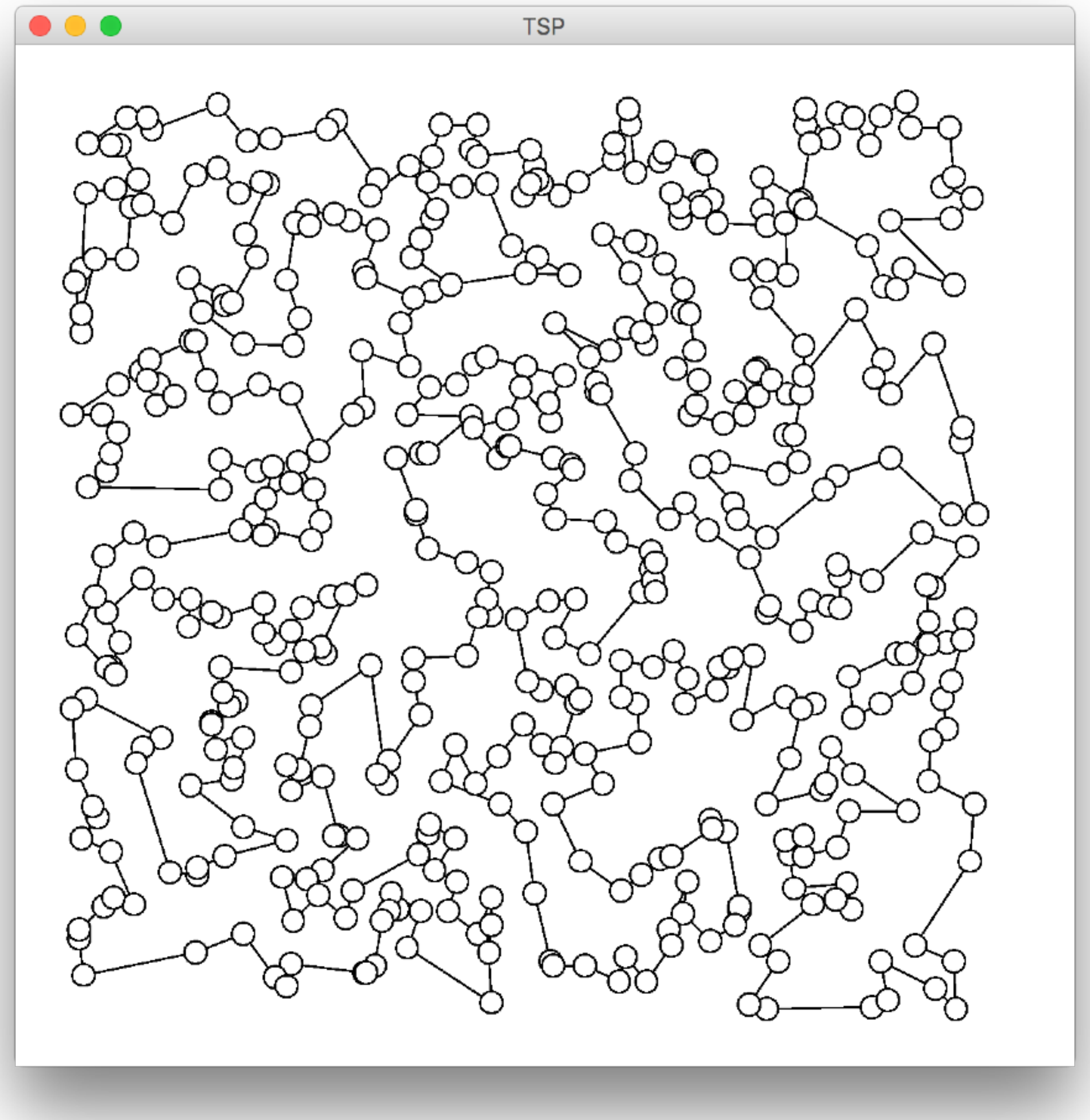
Disconnect by removing two edges, and re-construct the tour.

Euclidian TSP: optimal solution cannot have crossing edges. This move will avoid to have crossing edges.

2 Opt Evolution

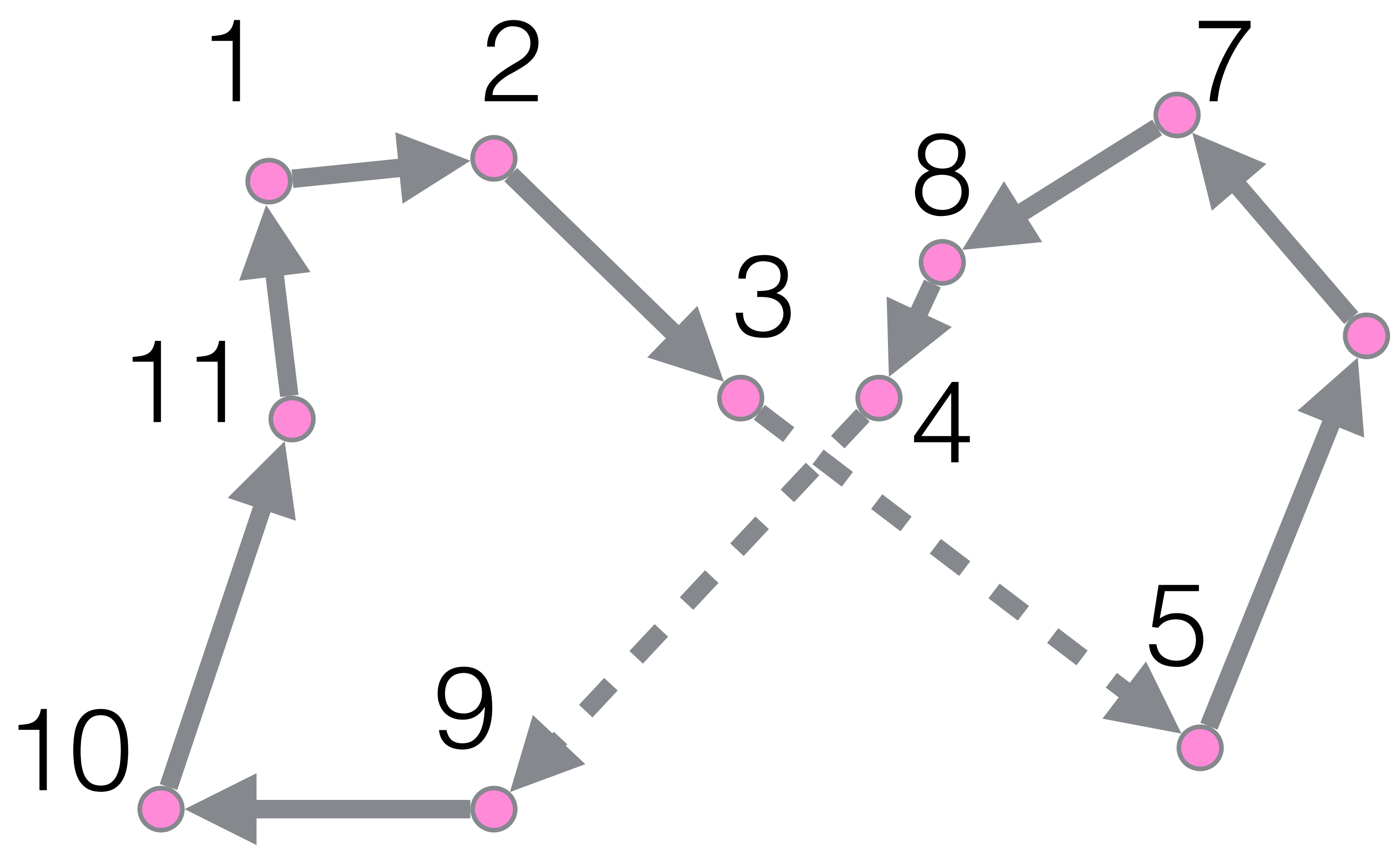






Is 2-Opt a connected?

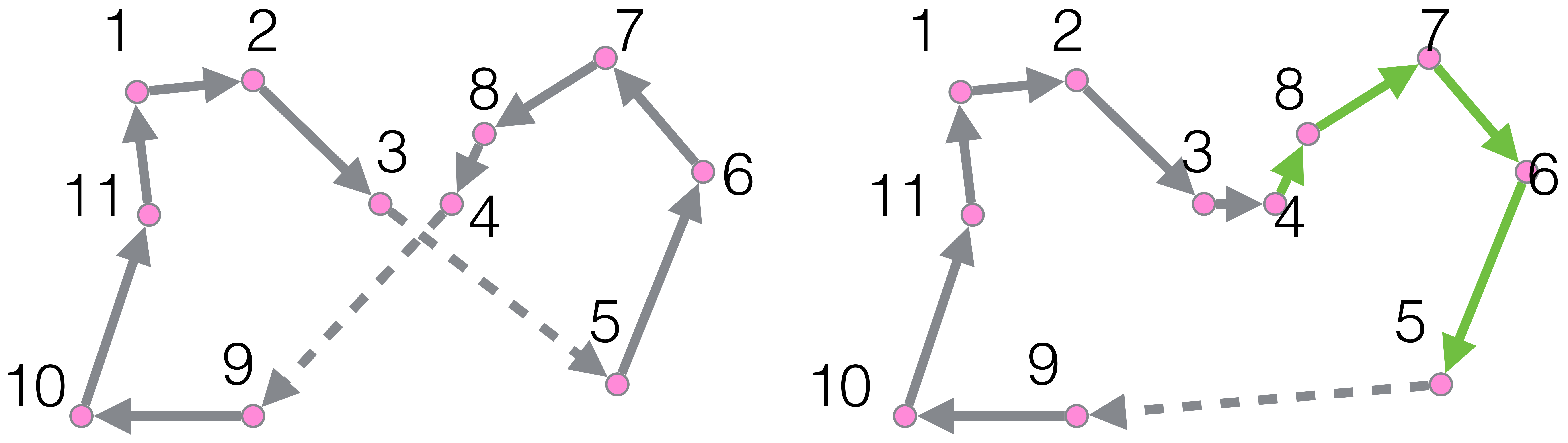
- Is the 2Opt move a connected neighbourhood for the TSP?



1,2,3,5,6,7,8,4,9,10,11

Solution represented as a permutation array

2-Opt and connectivity



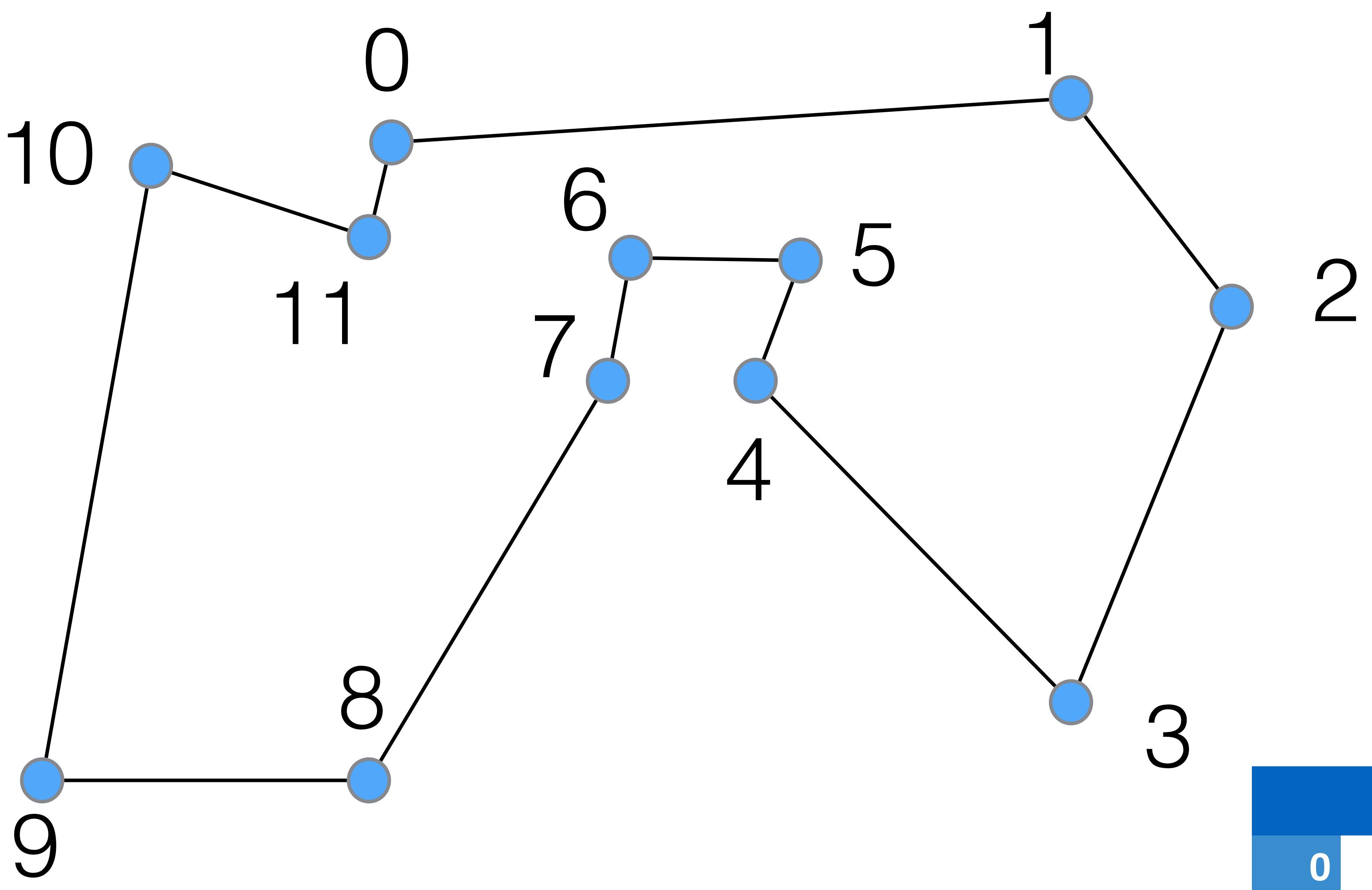
1,2,3,5,6,7,8,4,9,10,11 → 1,2,3,4,8,7,6,5,9,10,11

A 2-Opt move amounts at « reversing » a subsequence of the tour

3,1,2,4,7,6,5,8,9,11,10

assume this is the optimal solution s*, given a current solution s, can you find a sequence of 2-opt move to transform s into s* ?

Working example



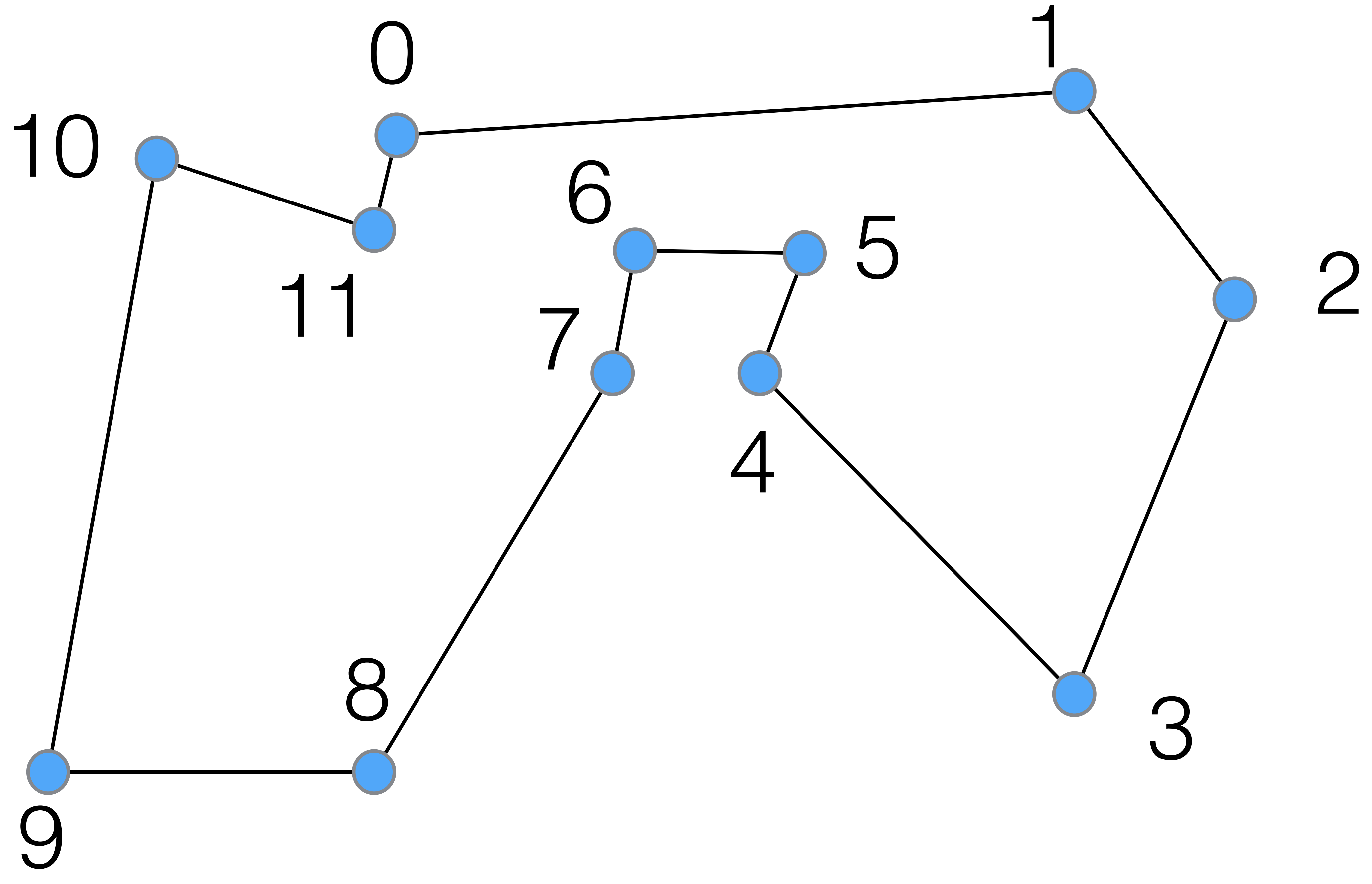
totalCost = 2056

Symmetric Distance Matrix

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	355	444	453	225	221	136	166	326	372	118	45
1	355	0	135	311	212	160	242	278	509	636	473	365
2	444	135	0	219	243	223	312	321	511	658	559	441
3	453	311	219	0	232	269	324	288	367	527	546	429
4	225	212	243	232	0	63	92	78	297	426	330	210
5	221	160	223	269	63	0	90	118	356	476	336	219
6	136	242	312	324	92	90	0	66	306	407	247	129
7	166	278	321	288	78	118	66	0	244	358	260	141
8	326	509	511	367	297	356	306	244	0	160	335	281
9	372	636	658	527	426	476	407	358	160	0	327	331
10	118	473	559	546	330	336	247	260	335	327	0	120
11	45	365	441	429	210	219	129	141	281	331	120	0

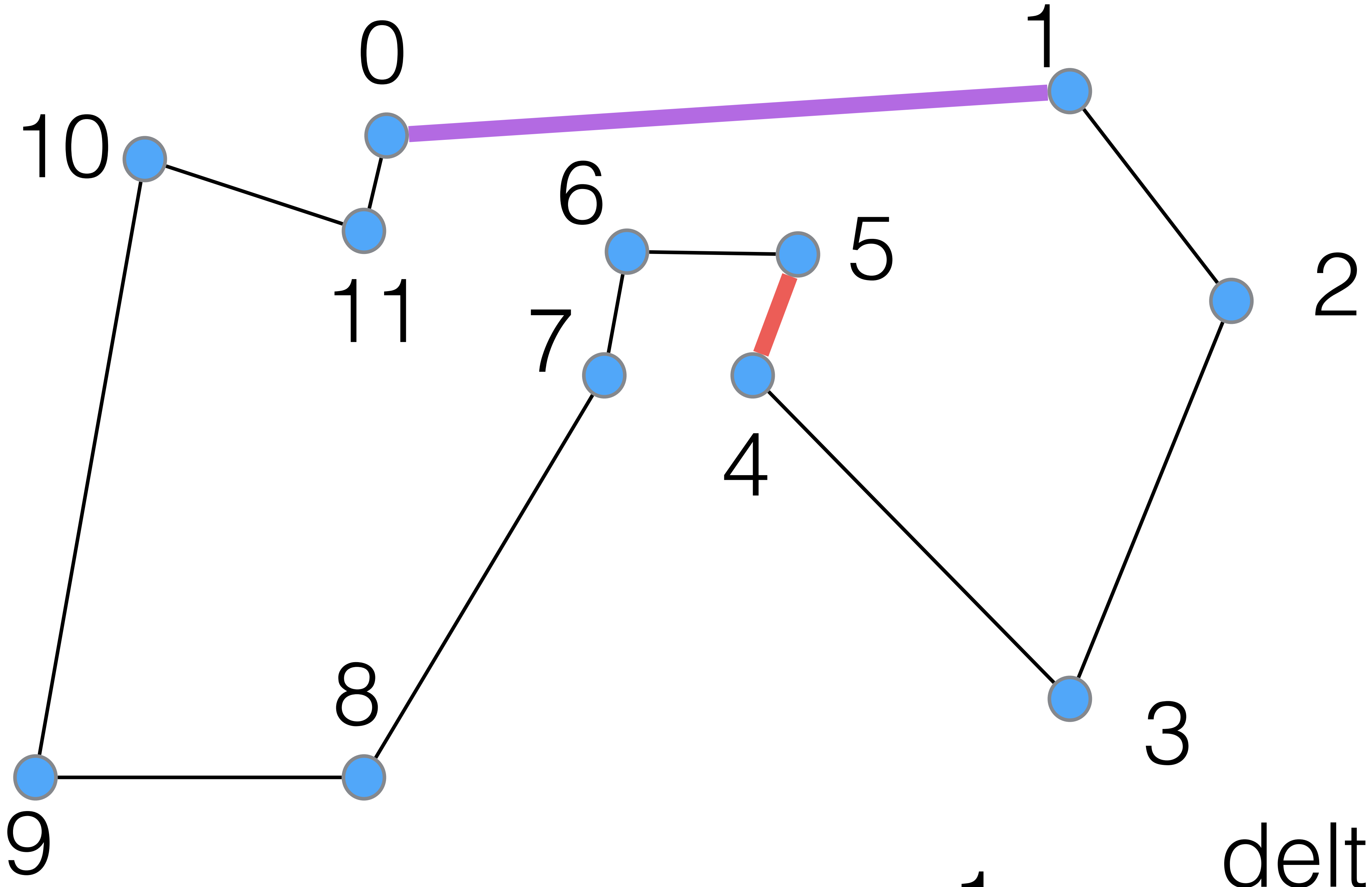
Question

- We know that In a Euclidien TSP, if the current solution has two crossing edges, it can be improved with a single 2-OPT move.
- But if the current solution has no crossing edges, does it mean that it can't be improve by a 2-OPT move?



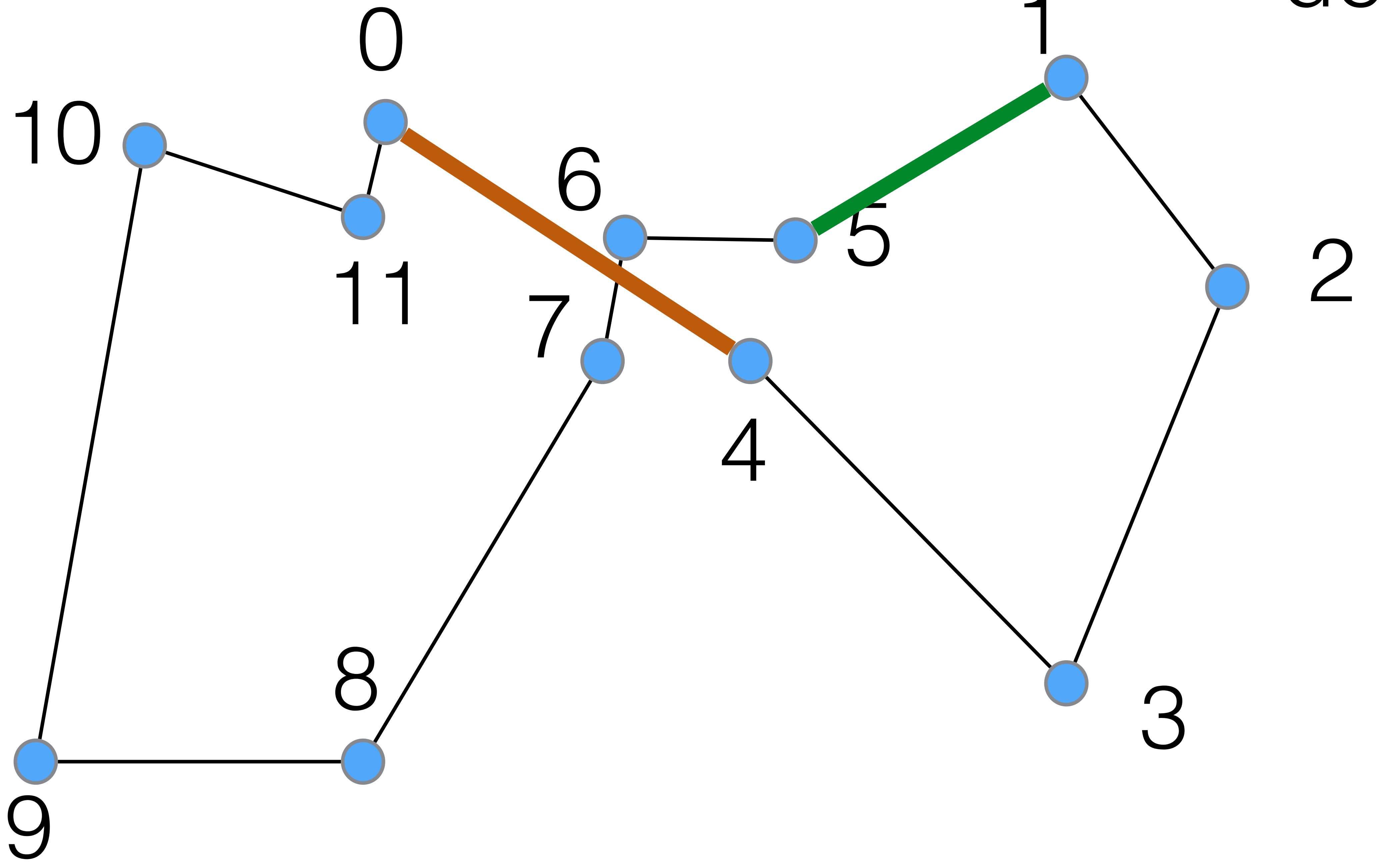
totalCost = 2056

Answer



totalCost = 2056

delta = **-355-63+225+160**

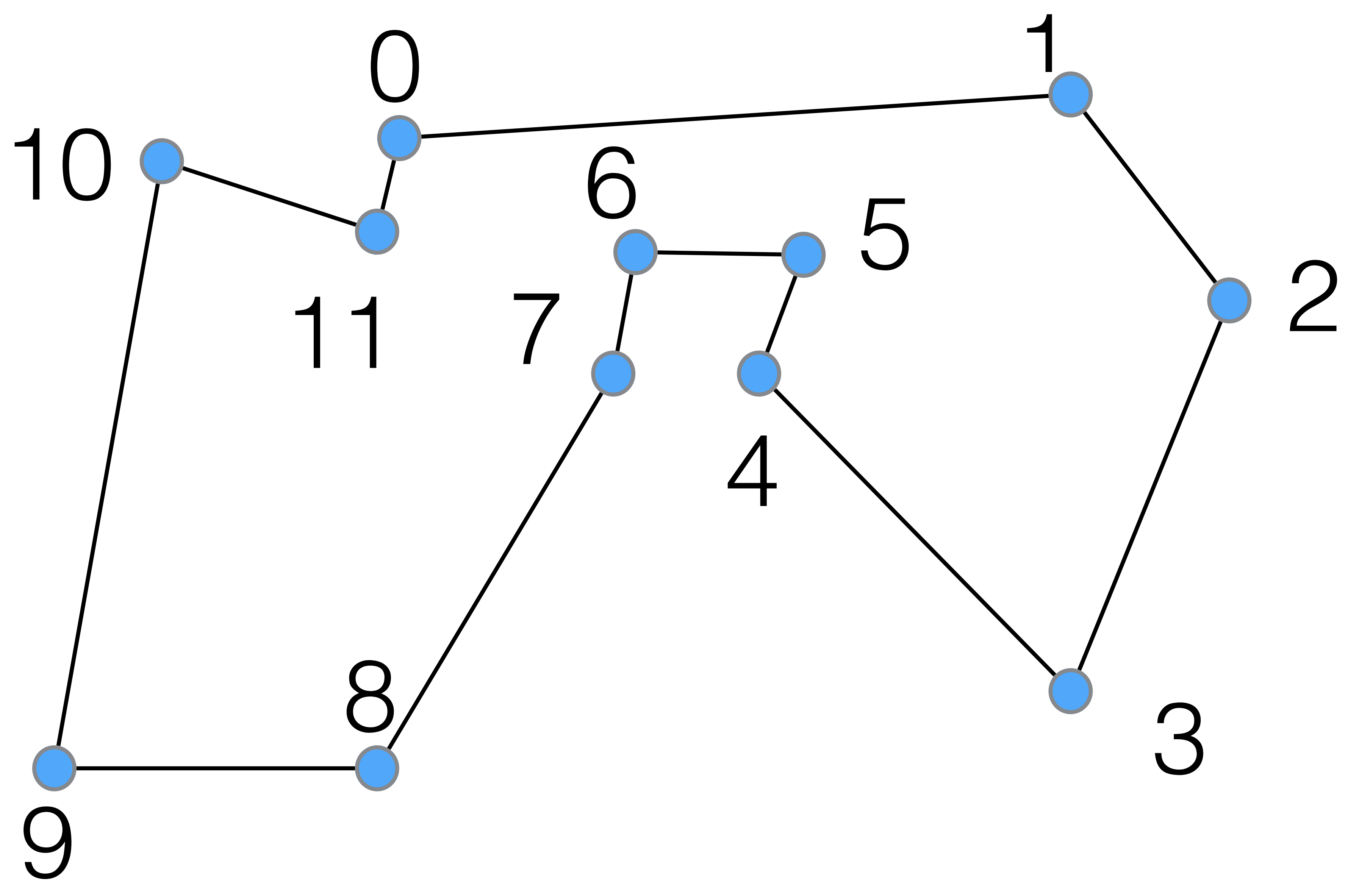


totalCost = 2023

Implementing 2-Opt: Generic Framework

```
static abstract class TSPLocalSearch {
    int n;
    int [][] dist;
    int [] tour;
    int [] tourSaved;

    TSPLocalSearch(TSPInstance data) {
        this.n = data.n;
        this.dist = data.distanceMatrix;
        this.tour = new int[data.n+1]; // first and last node are the same
        for (int i = 0; i < n; i++) {
            tour[i] = i;
        }
        tourSaved = Arrays.copyOf(tour,n+1);
    }

    public int[] currentTour() {
        return Arrays.copyOf(tour,n+1);
    }

    public void saveTour() {
        System.arraycopy(tour,0,tourSaved,0,n+1);
    }

    public void restoreSaved() {
        System.arraycopy(tourSaved,0,tour,0,n+1);
    }

    abstract boolean iteration();

    public void optimize() {
        int iter = 0;
        long t0 = System.currentTimeMillis();
        boolean improved = false;
        do {
            improved = iteration();
           iter += 1;
        } while (improved);
    }

}
```

0,1,2,3,4,5,6,7,8,9,10,11,0

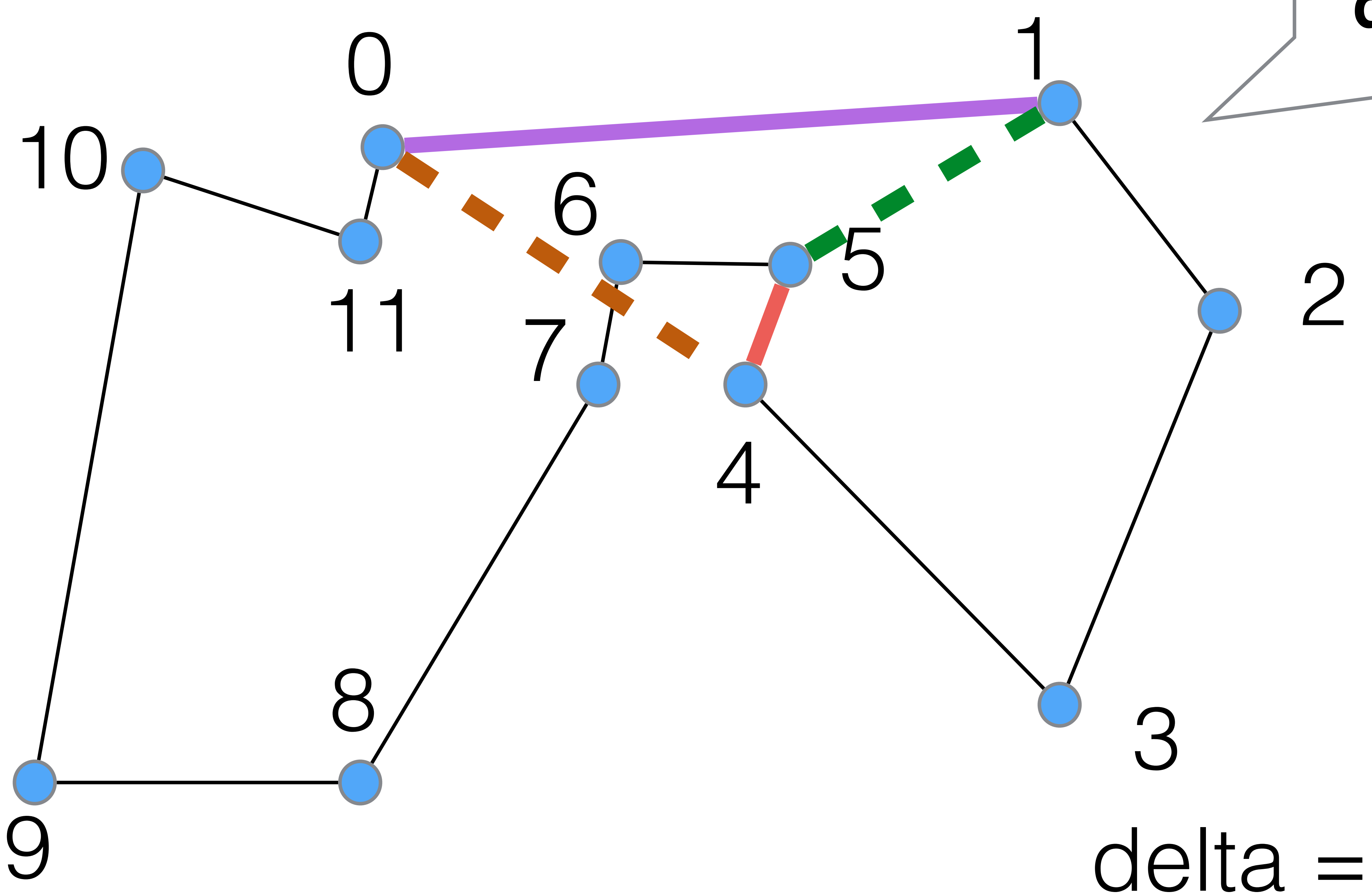


https://github.com/pschaus/linfo2266

delta2Opt(left,right)

Time Complexity ?

```
public int deltaTwoOpt(int left, int right) {
    int distLeft = dist[tour[left]][tour[left+1]];
    int distRight = dist[tour[right]][tour[right+1]];
    int distLeftNew = dist[tour[left]][tour[right]];
    int distRightNew = dist[tour[left+1]][tour[right+1]];
    return  distLeftNew + distRightNew - distLeft - distRight;
}
```



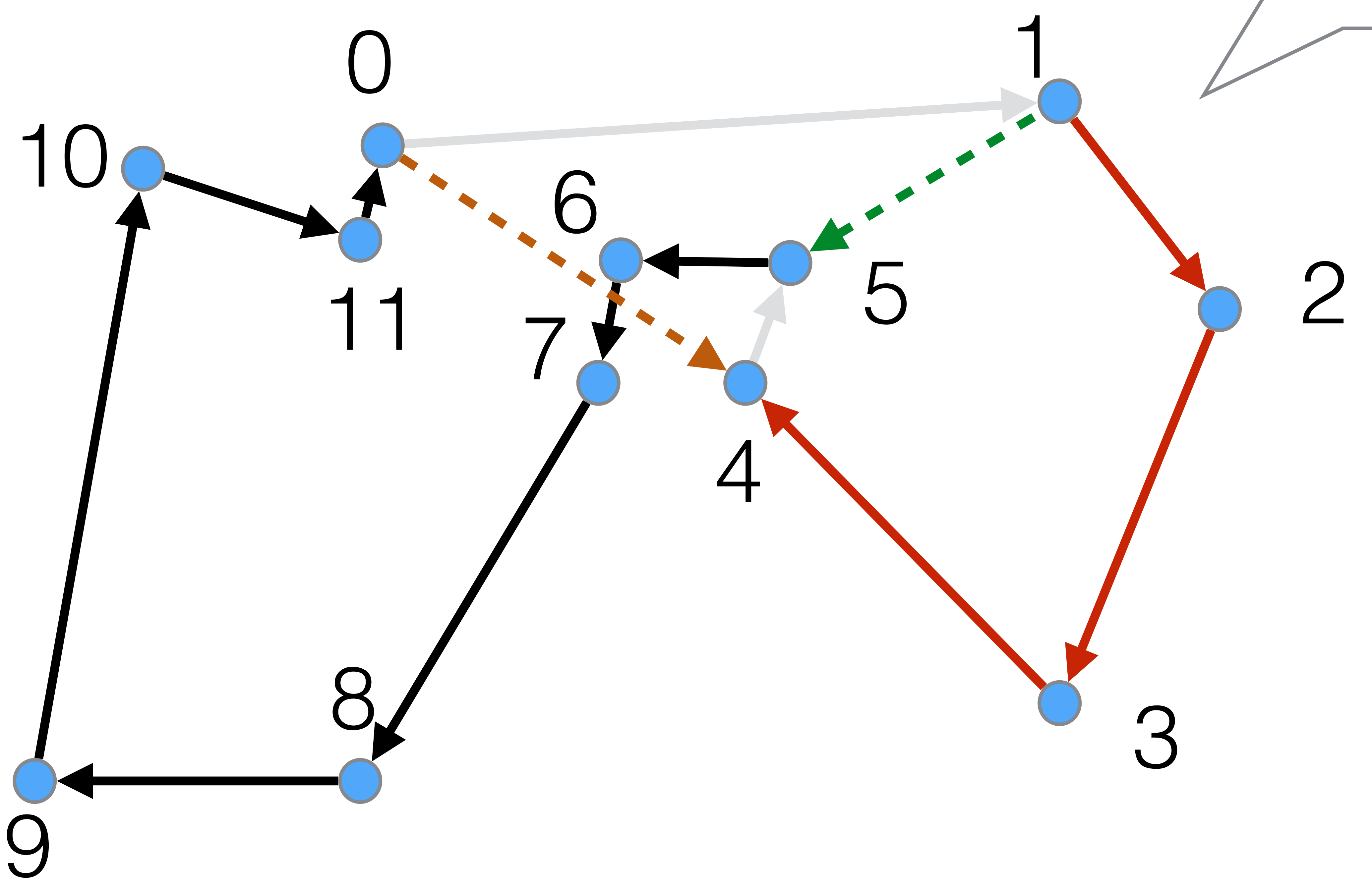
deltaTwoOpt(left=0,right=4)

totalCost = 2056

delta = **-355-63+225+160** = -33

twoOpt(left,right)

- Our tour representation is (artificially) « oriented » in our implementation



The red path must be flipped

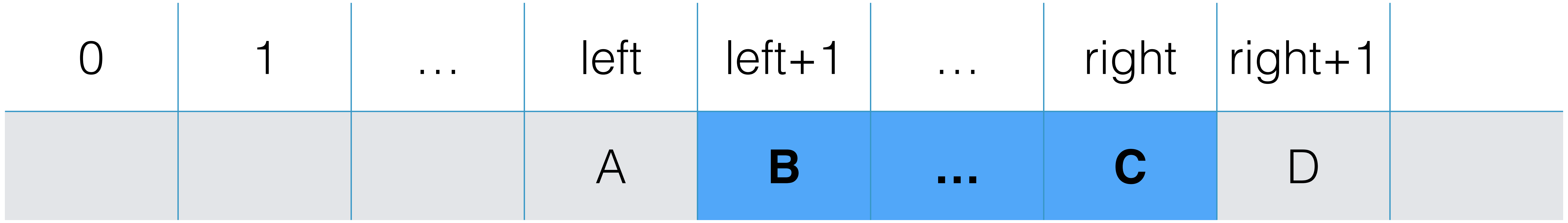
0,**1,2,3,4**,5,6,7,8,9,10,11,0

twoOpt(0,4)

0,**4,3,2,1**,5,6,7,8,9,10,11,0

twoOpt(left,right)

```
public void twoOpt(int left, int right) {
   for (int k = 0; k < (right - left + 1) / 2; k++) {
        int tmp = tour[left + 1 + k];
        tour[left + k + 1] = tour[right - k];
        tour[right - k] = tmp;
    }
}
```



0	1	…	left	left+1	…	right	right+1	
			A	**B**	**…**	**C**	D	

Effect of twoOpt is to swap this sub-array time: O(n)

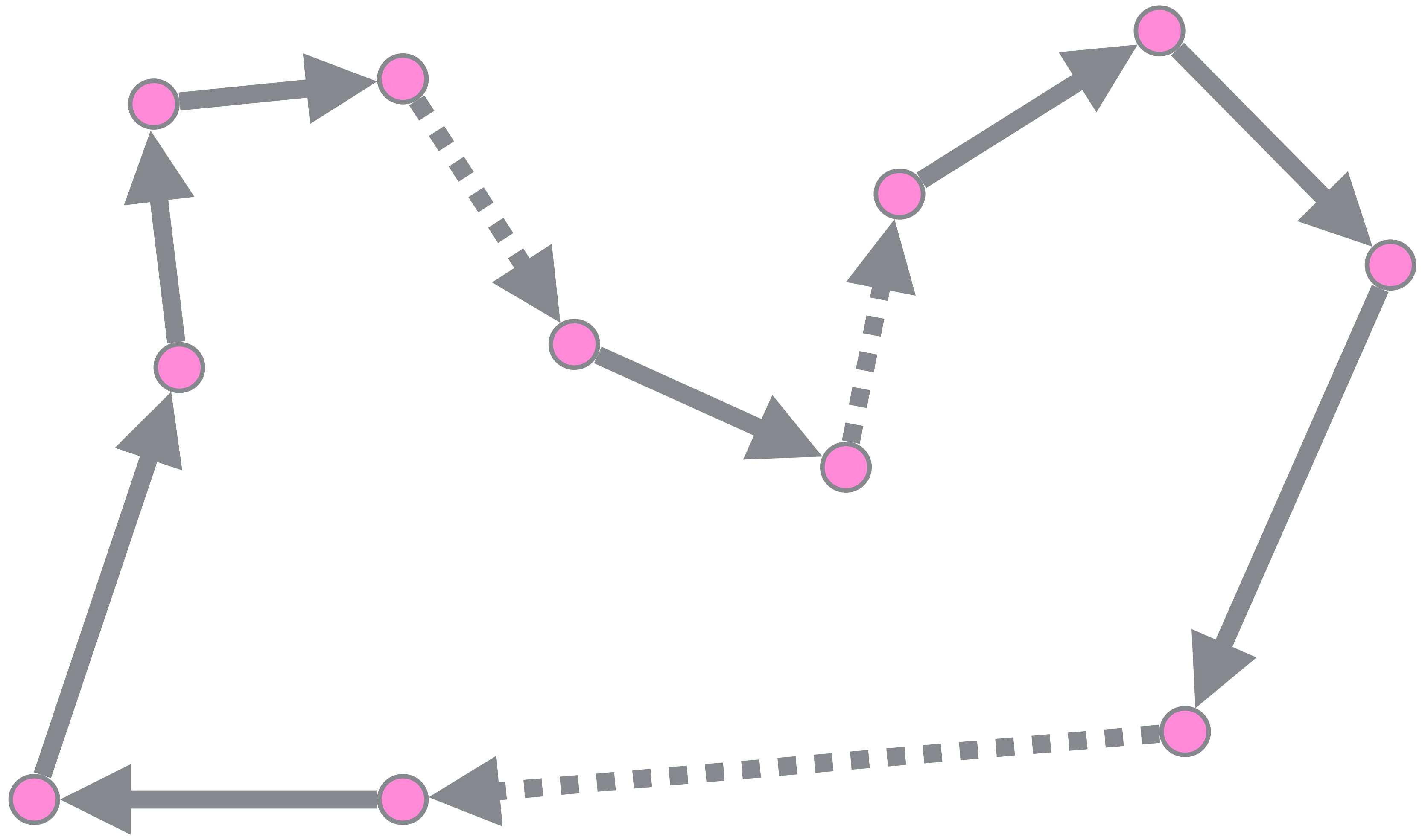
TSP2Opt

```
boolean iteration() {
    int bestLeft = 0, bestRight = 0, bestDelta = 0;
    // 2-opt move
    for (int left = 0; left < n; left++) {
        for (int right = left+1; right < n ; right++) {
            int delta = deltaTwoOpt(left,right);
            if (delta < bestDelta) {
                bestDelta = delta;
                bestLeft = left;
                bestRight = right;
            }
        }
    }
    twoOpt(bestLeft,bestRight);
    return bestDelta < 0;
}
```

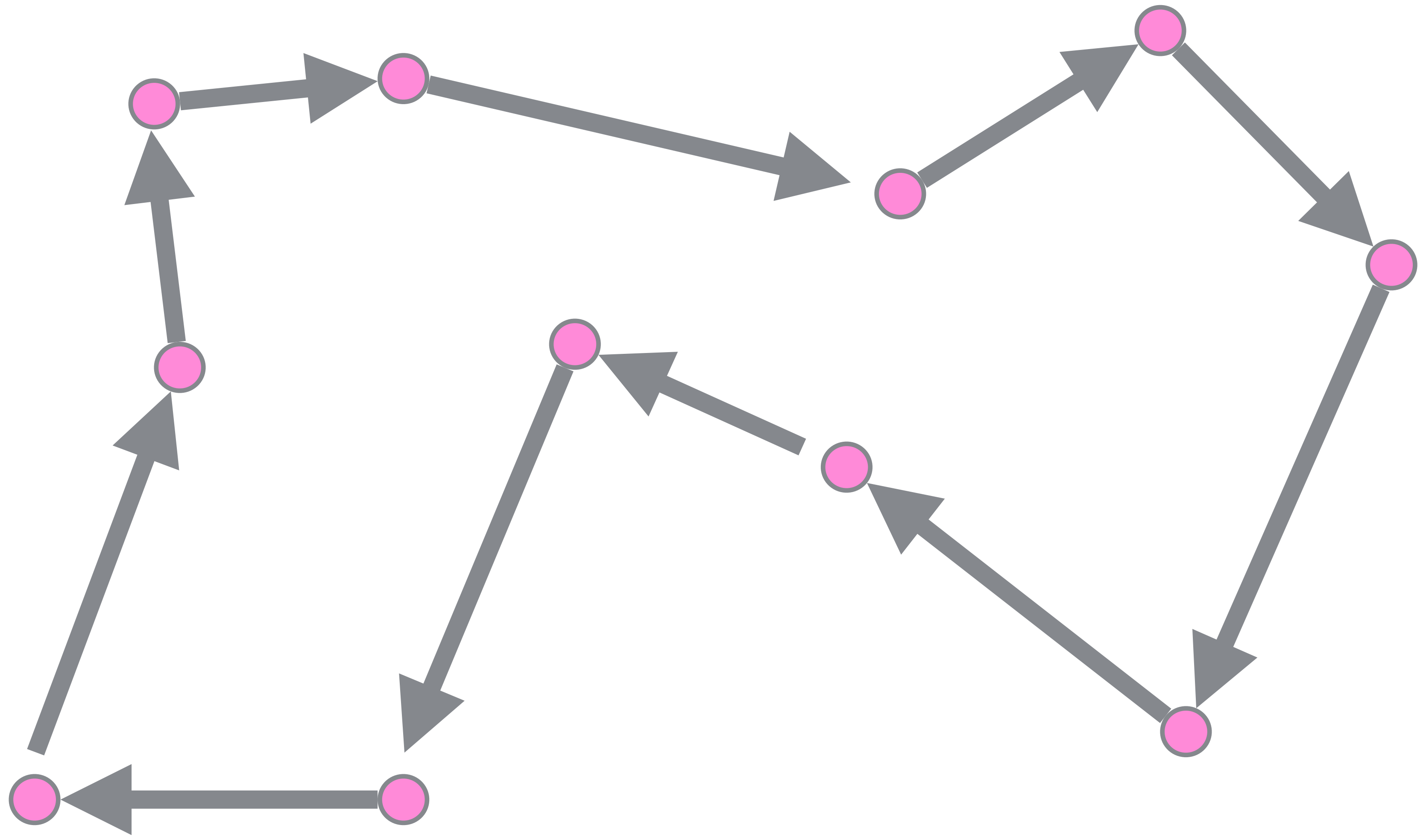
Time ?

3 Opt

- The neighbourhood is the set of all tours that can be obtained by removing 3 edges.



Problem: the neighborhood to explore becomes huge O(n^3). While 3-Opt can still be useful, 4-Opt almost never pays off.



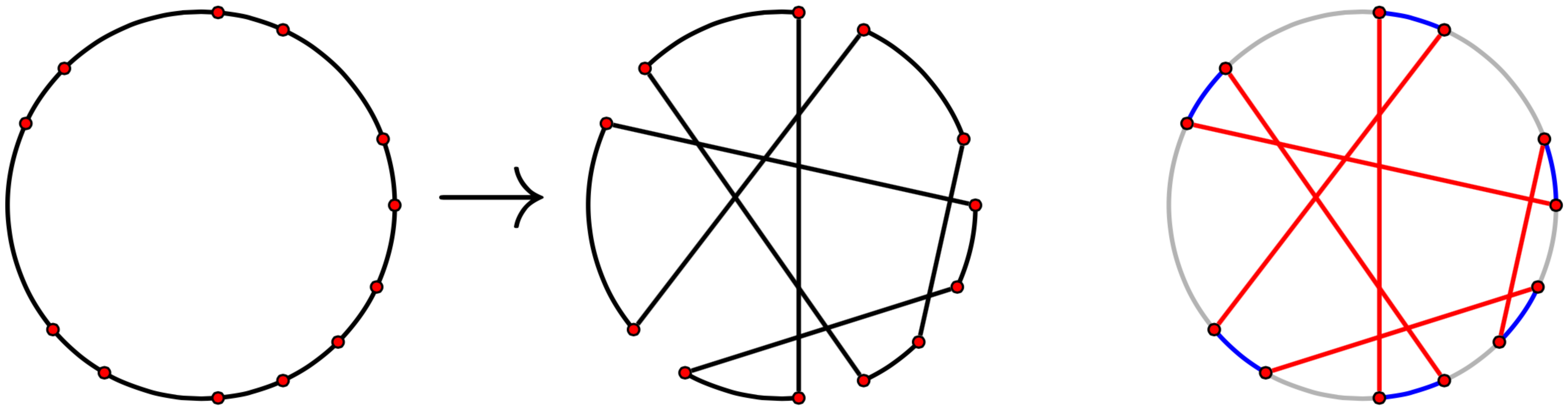
Let us dream …

- What about a K-Opt but the K can choose it-self.
- Sometime K can be 2, sometimes 5, etc.
- Would it be tractable?

Efficient K-Opt (Lin-Kernighan)

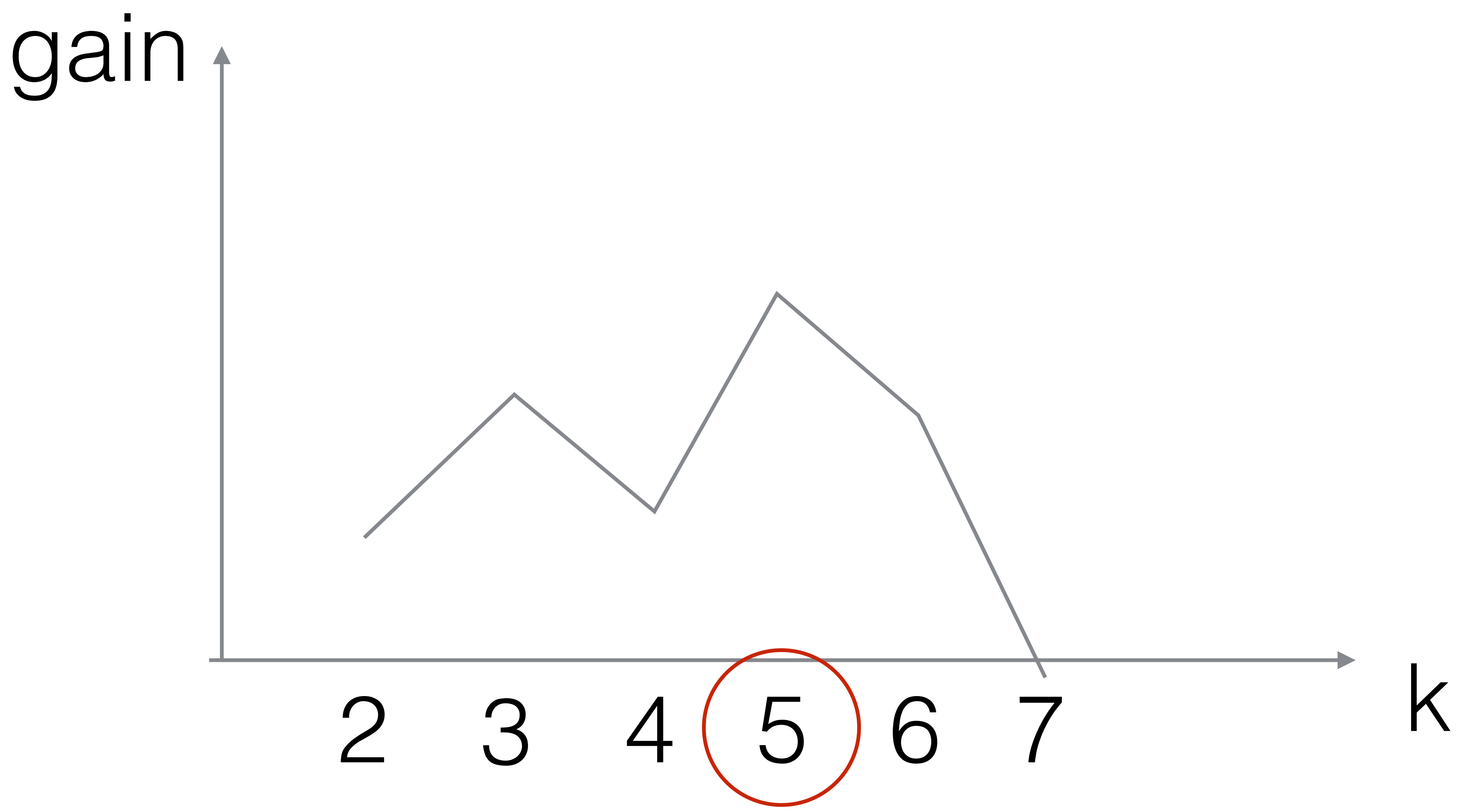
- NO: General K-Opt is computationally too intensive
- But we can limit our-self to a particular form of k-Opt moves: ***Sequential K-Opt moves***
- A K-Opt move is called sequential if it can be described by a **path alternating between deleted and added edges.**

Example: sequential 6-Opt



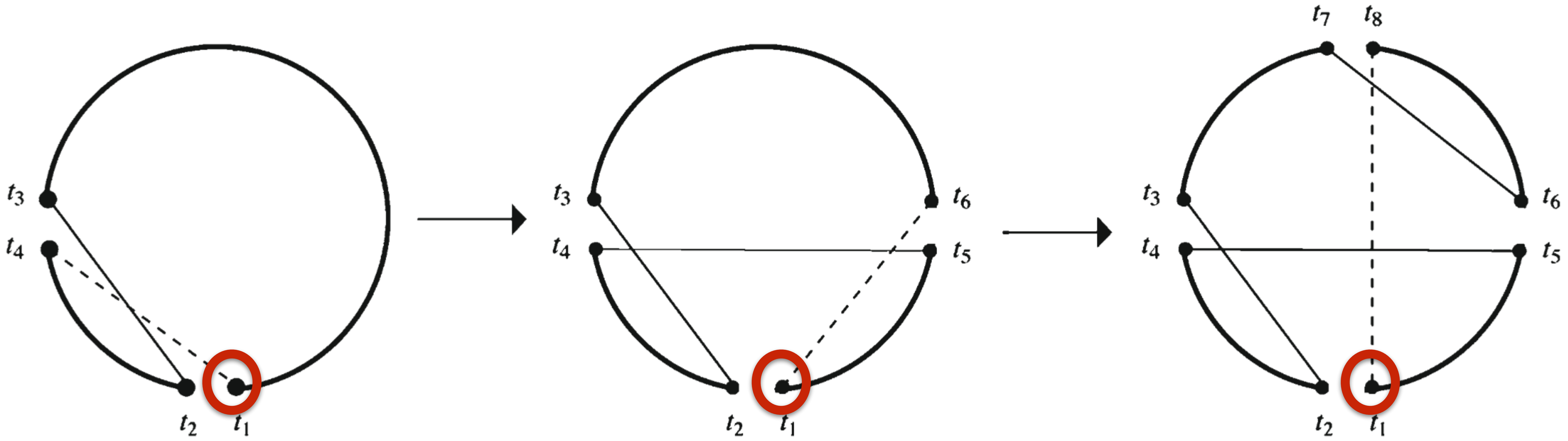
Lin-Kernighan

- Sequential K-Opt moves: What values for k ?
- The idea is to build greedily a K-Opt move
- At each step we compute the « gain » of applying it.
  - The (k+1)-sequential move is an extension of the k-sequential move, etc
- Then we select (retrospectively) the k that gives the best « gain »



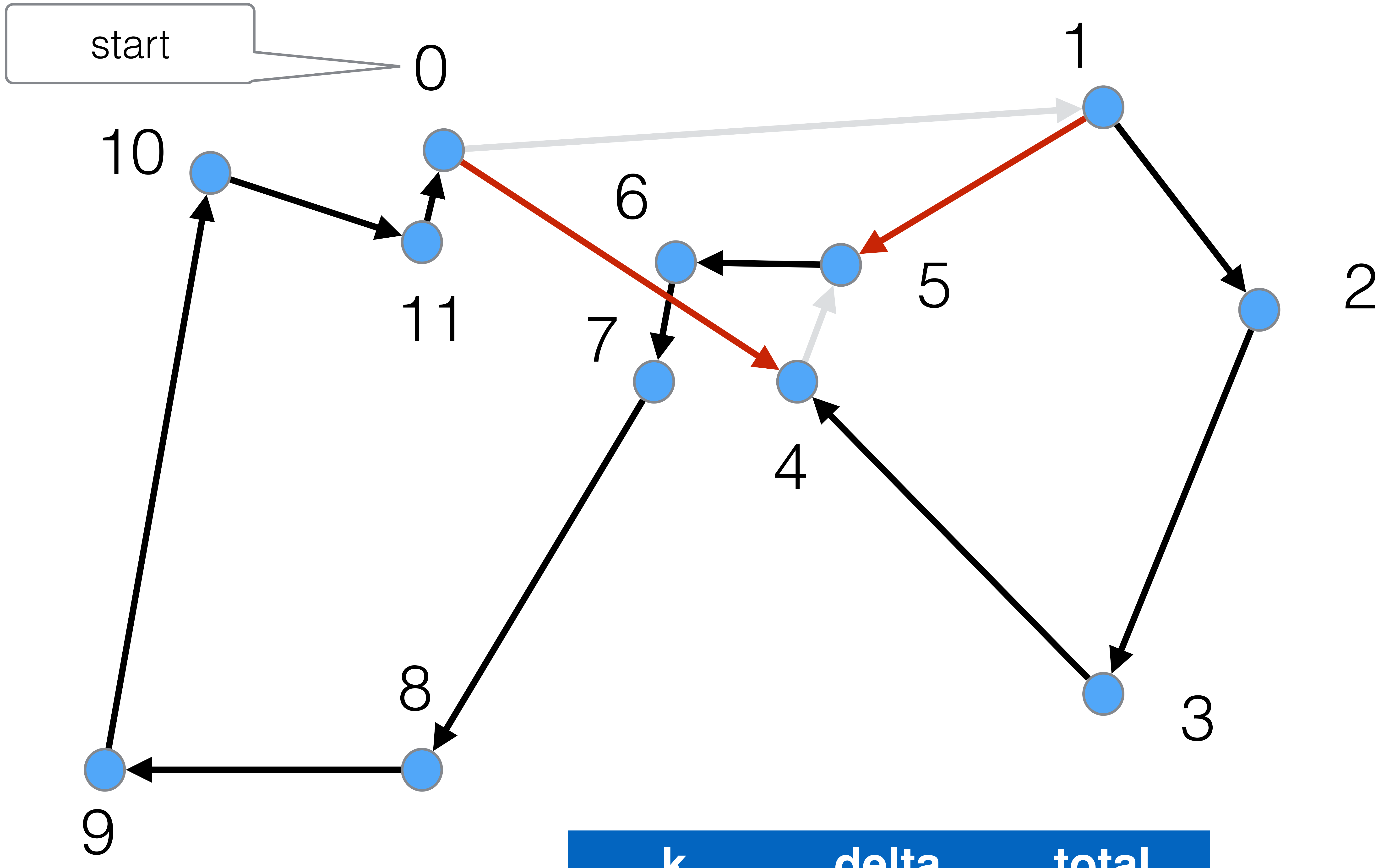
Observation:

- A sequential k-Opt move has the same effect as k-1 2Opt Moves.
- Example: sequential 4-Opt = 2Opt + 2Opt + 2Opt



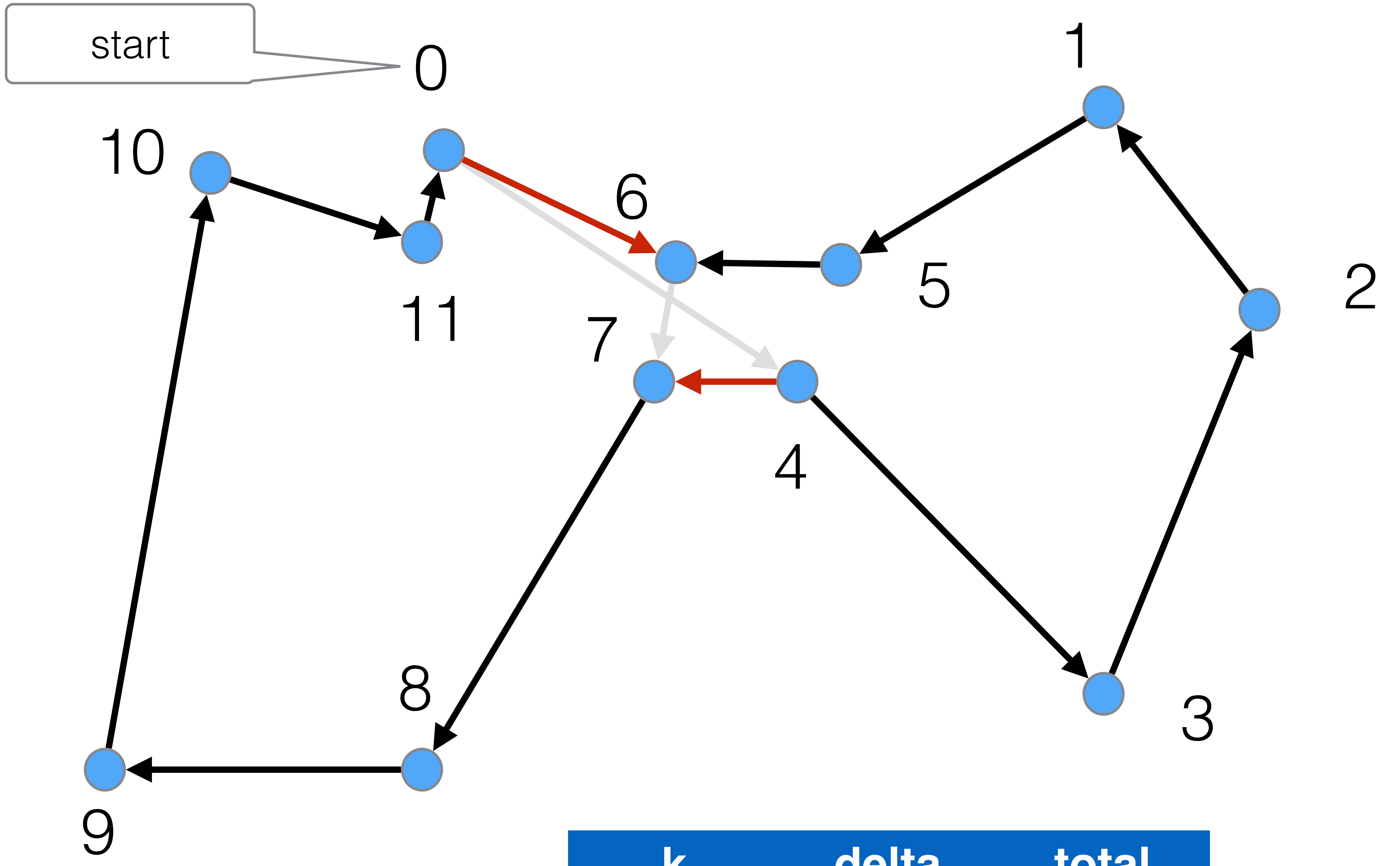
Sequential 4-opt move performed by three 2-opt moves. Close-up edges are shown by *dashed lines*

K-Opt starting at 0



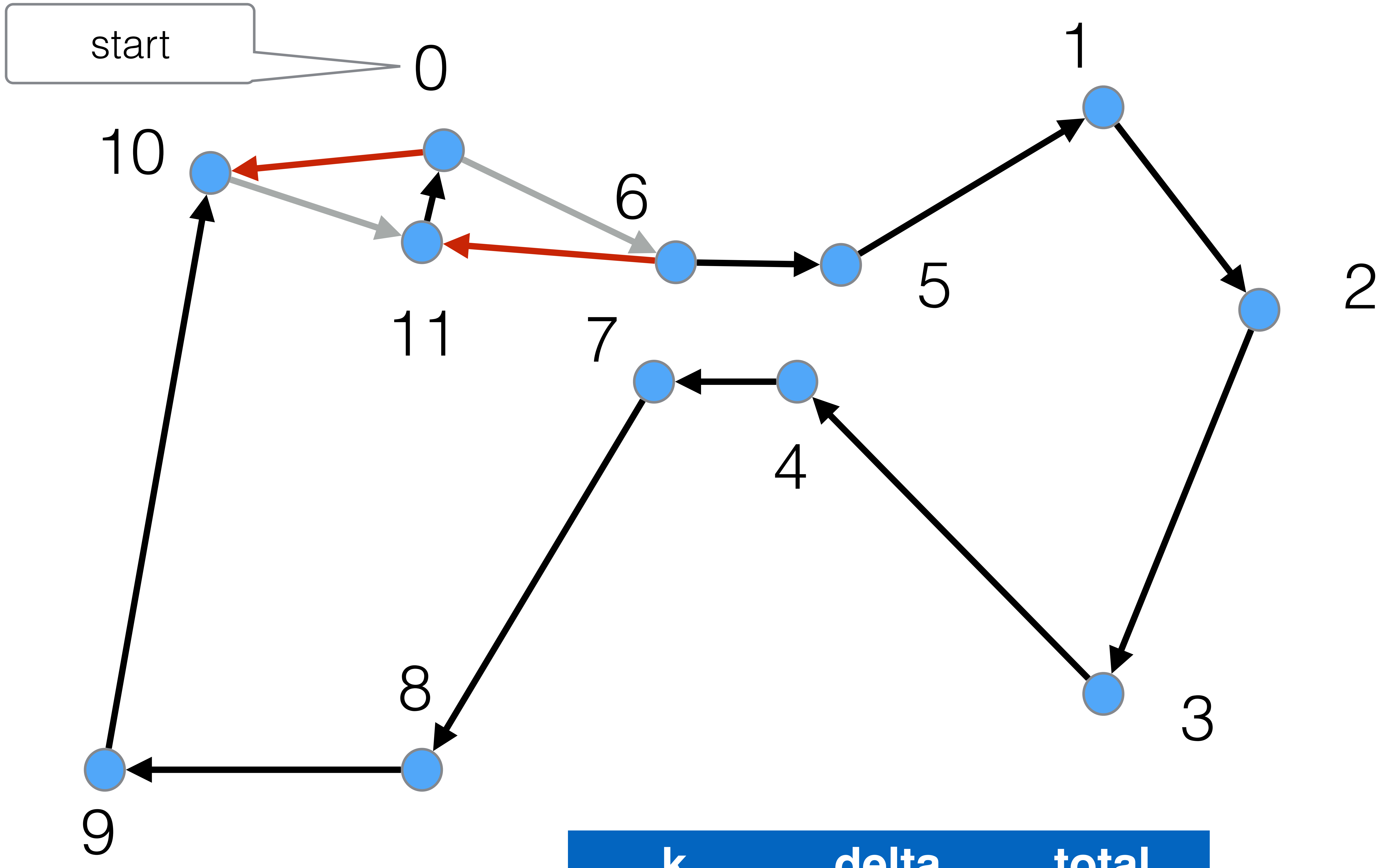
k	delta	total
2	-33	-33

K-Opt starting at 0



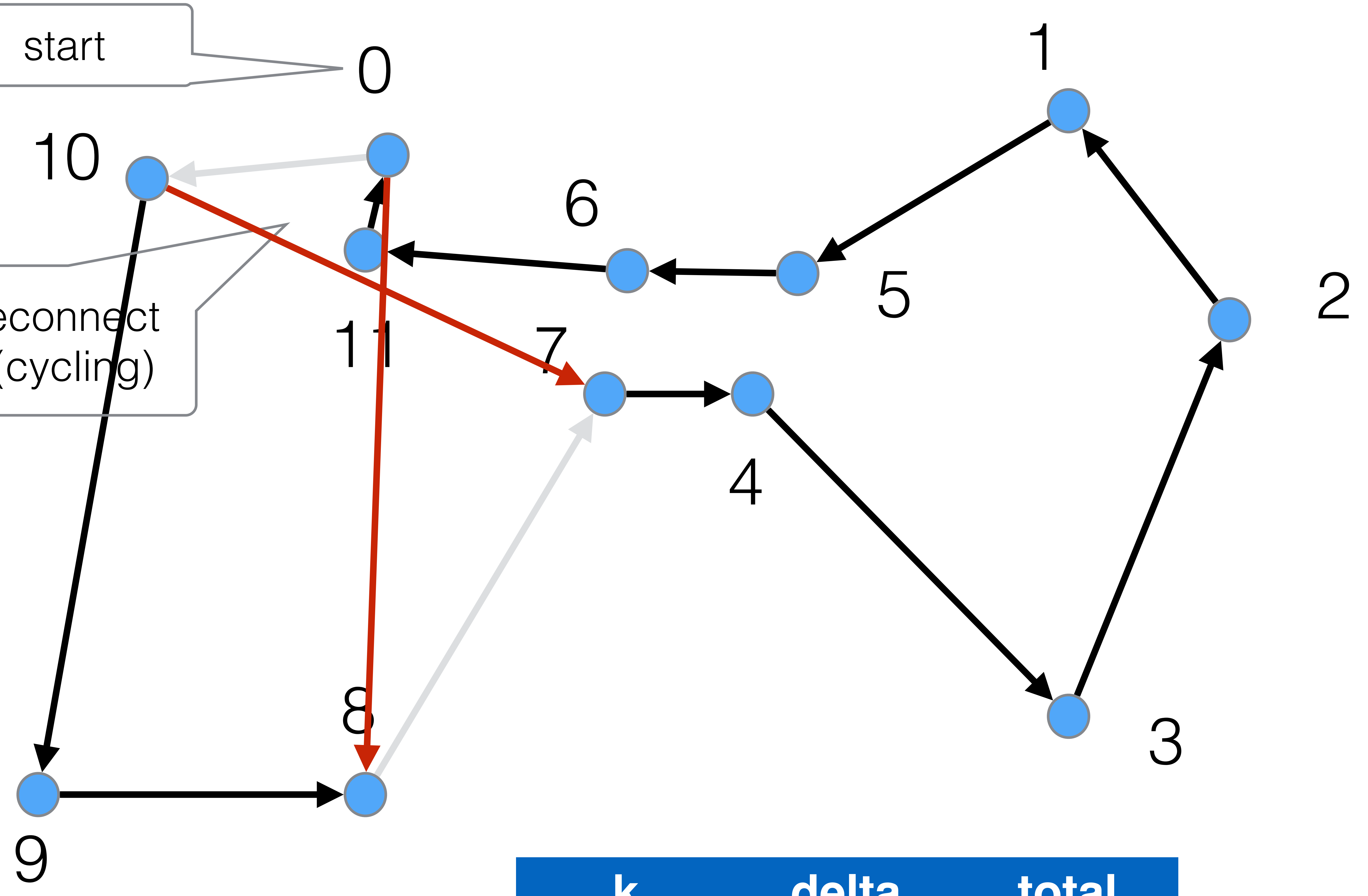
k	delta	total
2	-33	-33
3	-77	-110

K-Opt starting at 0



k	delta	total
2	-33	-33
3	-77	-110
4	-9	-119

K-Opt starting at 0



start

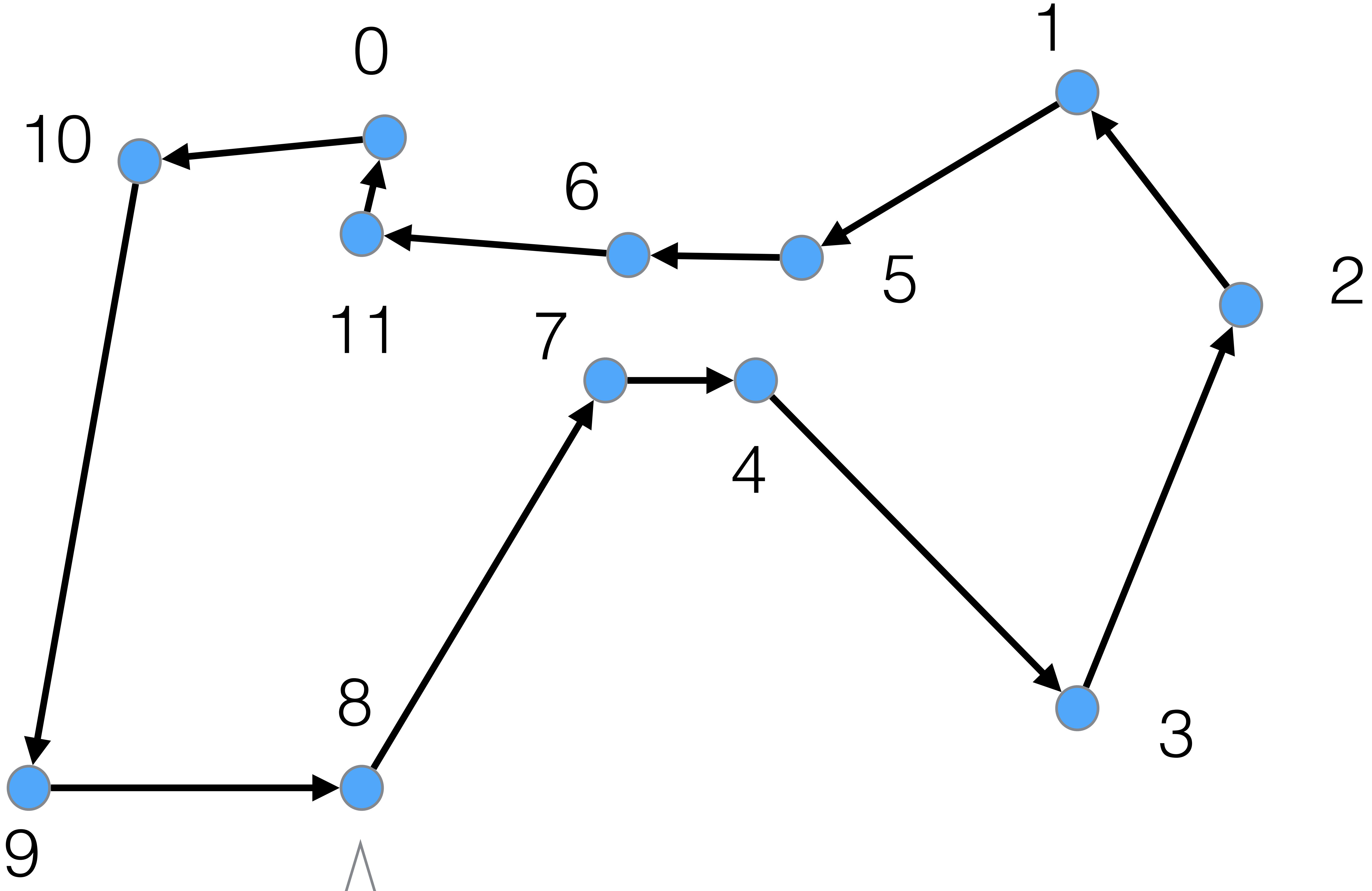
don’t reconnect 10-11 (cycling)

k	delta	total
2	-33	-33
3	-77	-110
4	-9	-119
5	224	105

restore this solution

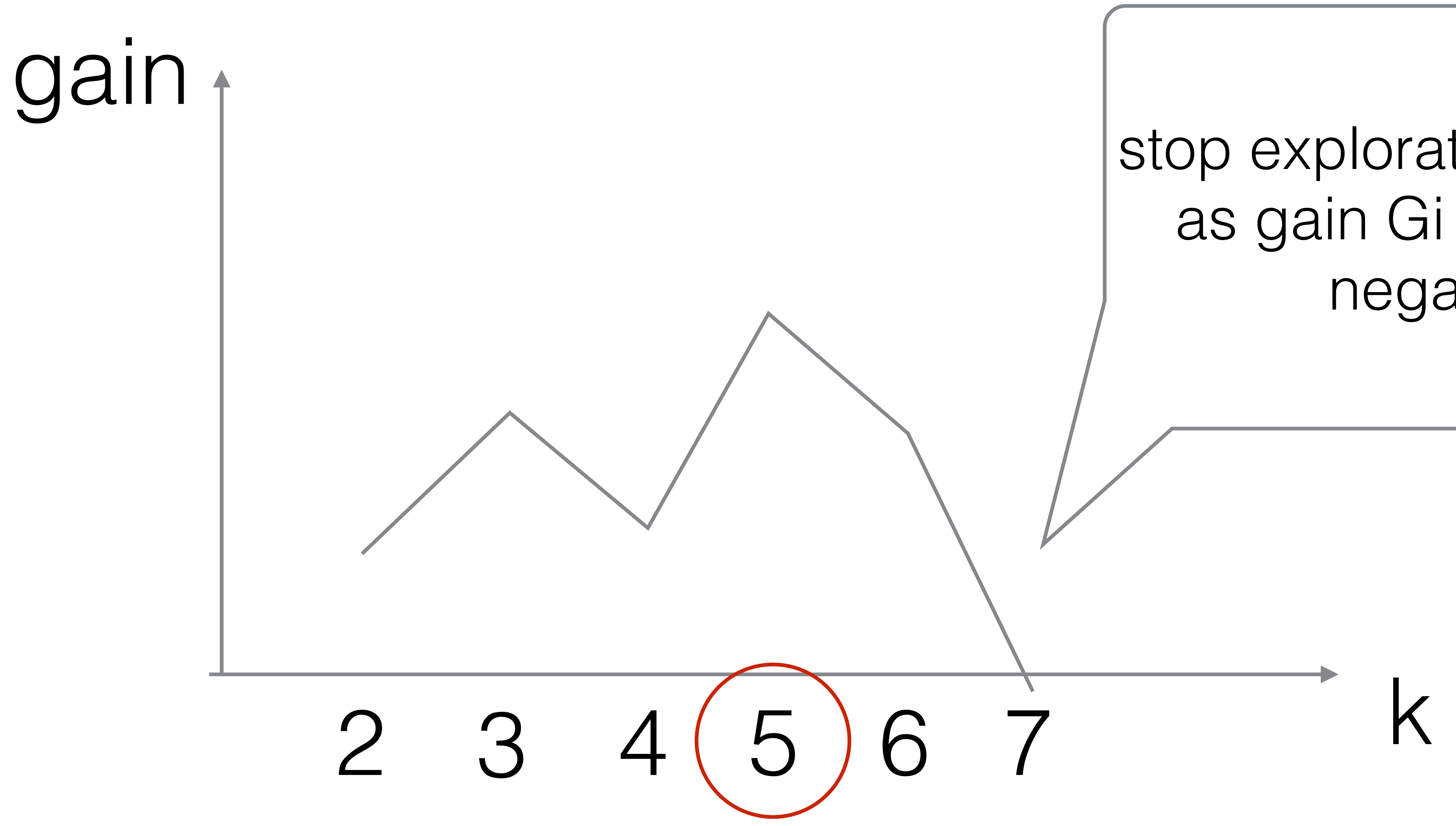
stop!

K-Opt starting at 8



Do a K-Opt iteration starting from 8, do at-least 2, you should see something very nice after two iterations (that 2-Opt alone cannot see)

Positive gain stoping criterion for one iter



stop exploration as soon as gain Gi becomes negative

Implementing K-Opt

```
static class TSPKOpt extends TSPLocalSearch {

    int K;

    TSPKOpt(TSPInstance data, int K) {
        super(data);
        this.K = K;
    }

    public boolean kOptFrom(int i) {
        …
    }

    @Override
    boolean iteration() {
        var found = false;
        for (int i = 0; i < n; i++) {
           if (kOptFrom(i)) {
                found = true;
            }
        }
        return found;
    }
}
```

Implementing K-Opt:

```
public boolean kOptFrom(int i) {
    saveTour();
    int cumulatedDelta = 0;
    int bestCumulatedDelta = cumulatedDelta;
    int bestK = 0;
    int k = 0;
    int prev = -1;
    do {
        k += 1;
        int bestDelta = Integer.MAX_VALUE;
        int bestj = -1;
        for (int j = 0; j < n; j++) {
            int dist = Math.abs(j-i);
            if (1 < dist && dist < n-1 && j != prev) { // no stupid 2-opt or direct cycling
                int delta = deltaTwoOpt(i,j);
                if (delta < bestDelta) {
                    bestDelta = delta;
                    bestj = j;
                }
            }
        }
        prev = bestj;
        twoOpt(i,bestj);
        cumulatedDelta += bestDelta;
        if (cumulatedDelta < bestCumulatedDelta) {
            bestCumulatedDelta = cumulatedDelta;
            bestK = k;
            saveTour();
        }
    } while (k < K);
    restoreSaved();
    return bestCumulatedDelta < 0;
}
```

look for the next 2-opt move

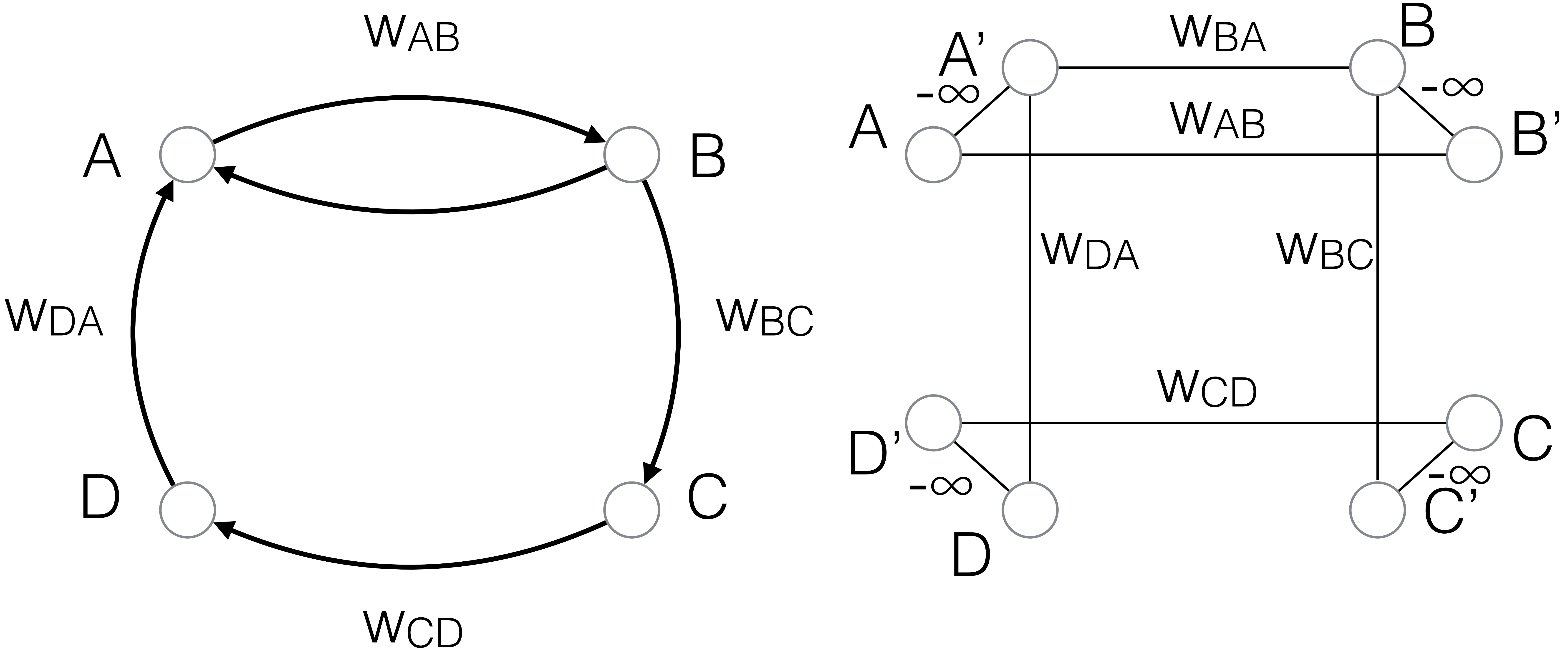
only store the solution if it is the best so-far (cumulatedDelta)

positive gain stopping criterion, not sure it is a good idea

restore the solution corresponding to the best k (possibly the initial one)

ATSP reduction to TSP

- The distance matrix is not always symmetric (city: one-way roads, etc).
- Possible to transform an ATSP into a TSP by doubling the number of nodes.

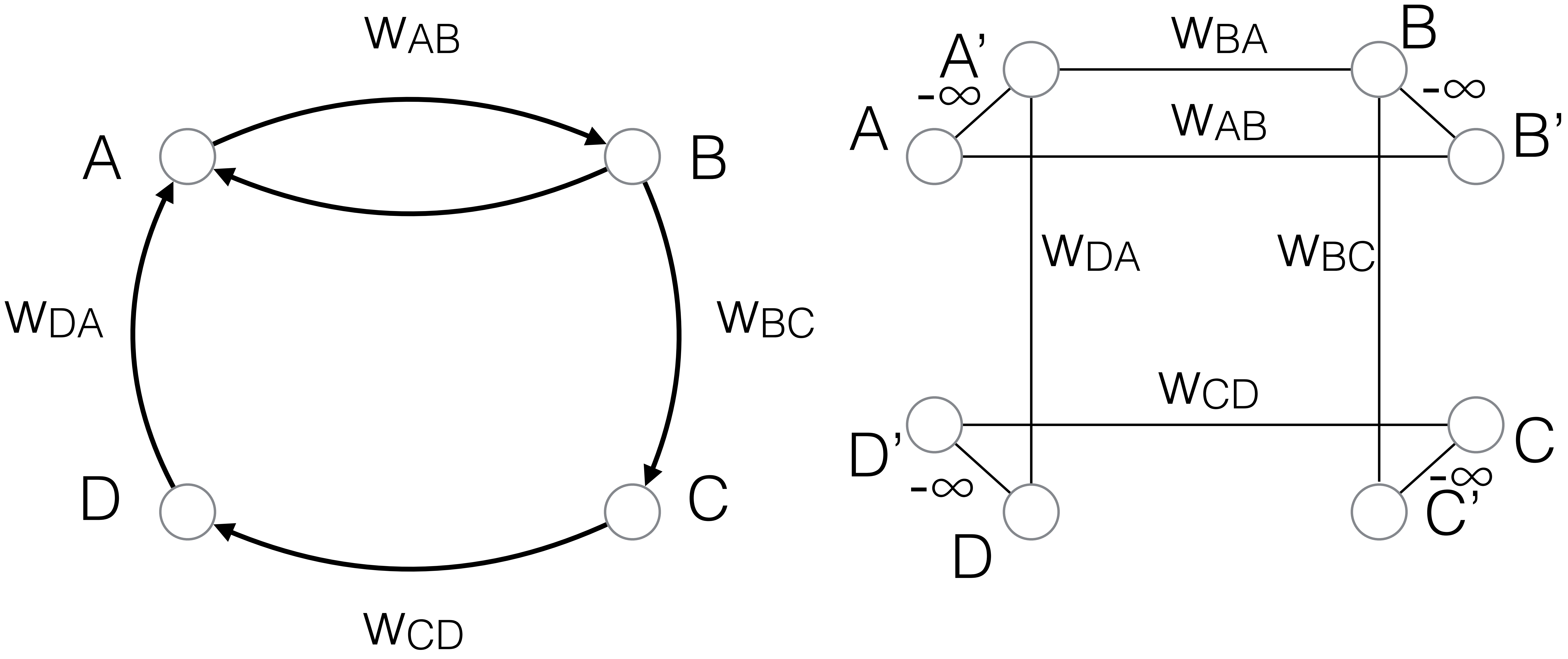


1 to 1 correspondance

Jonker, Roy; Volgenant, Ton. "Transforming asymmetric into symmetric traveling salesman problems". *Operations Research Letters*, 1983

ATSP reduction to TSP

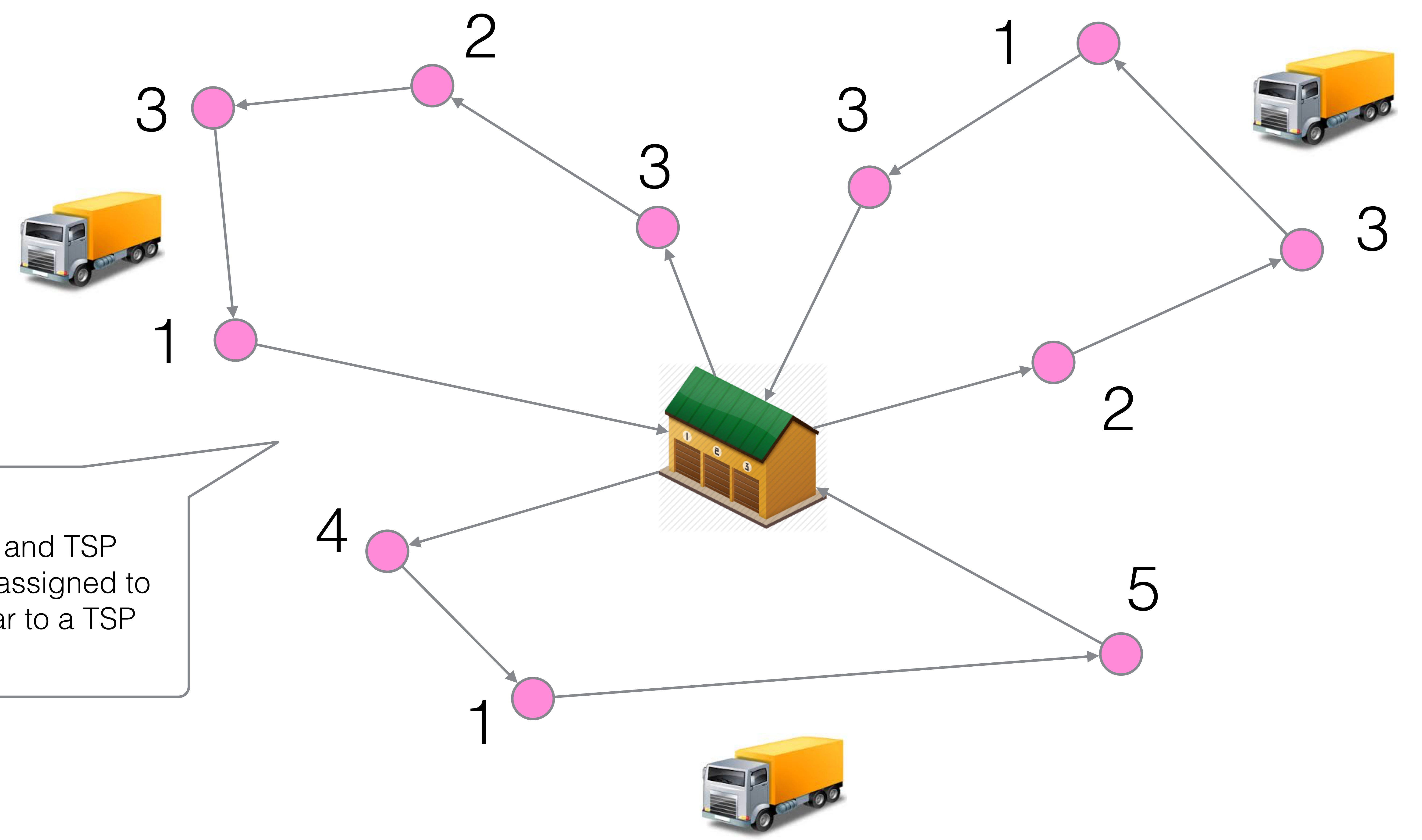
- Idea: N-N’ are so attractive (large negative number) that they are part of any optimal TSP. Hence A’B and AB’ cannot be both selected otherwise there would be a sub-tour (not hamiltonian circuit).



1 to 1 correspondance

Jonker, Roy; Volgenant, Ton. "Transforming asymmetric into symmetric traveling salesman problems". *Operations Research Letters*, 1983

Vehicle Routing



Mix of bin-packing and TSP
Once customers are assigned to vehicles, quite similar to a TSP

- Given vehicles starting from a depot each having a fixed capacity C
- How to visit each customer once without exceeding the capa (we charge a quantity at each customer) while minimizing the total distance?

Two strategies for initialization

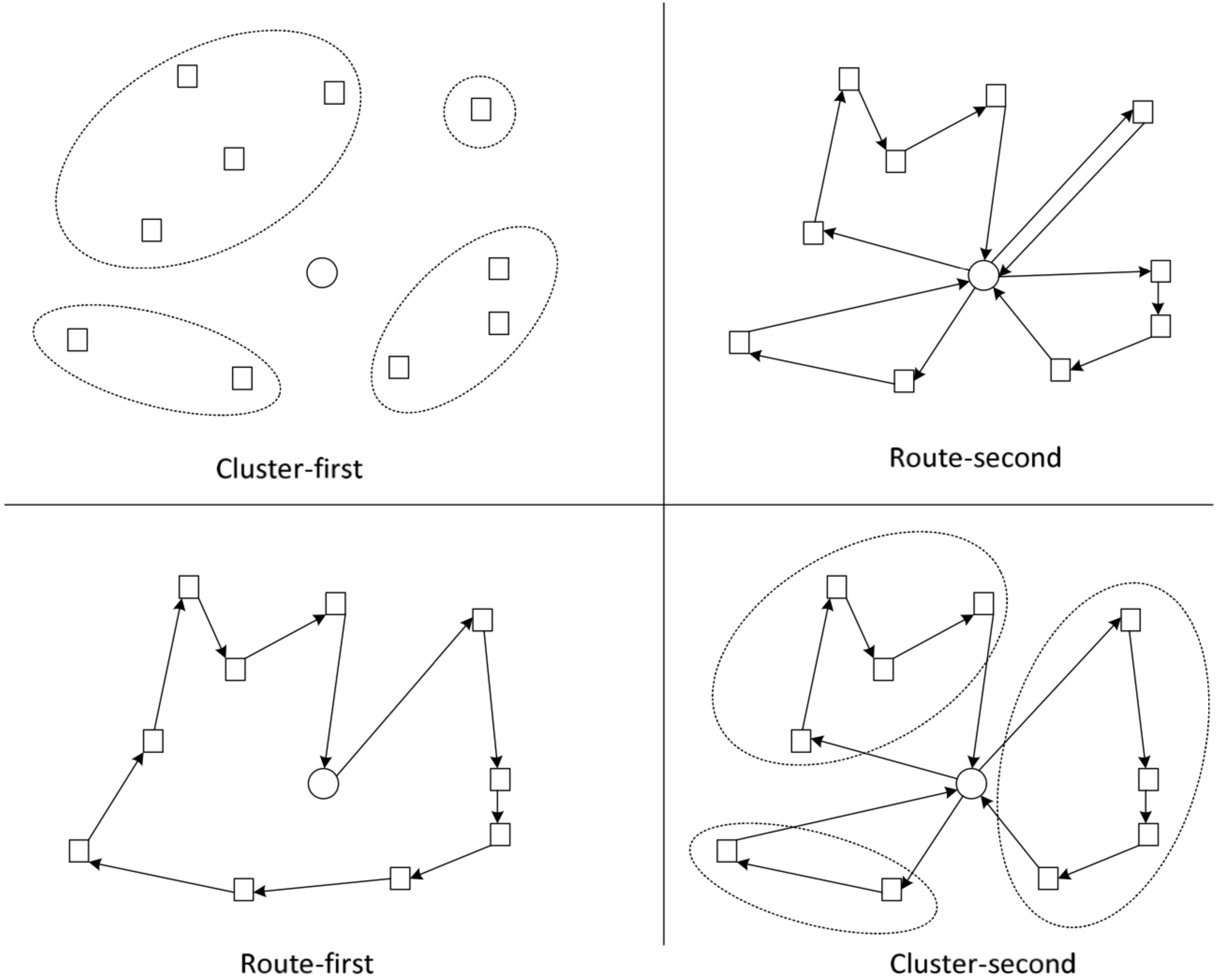
- VRP = partitioning (trucs) + sequencing (circuits)

- We usually use two strategies:

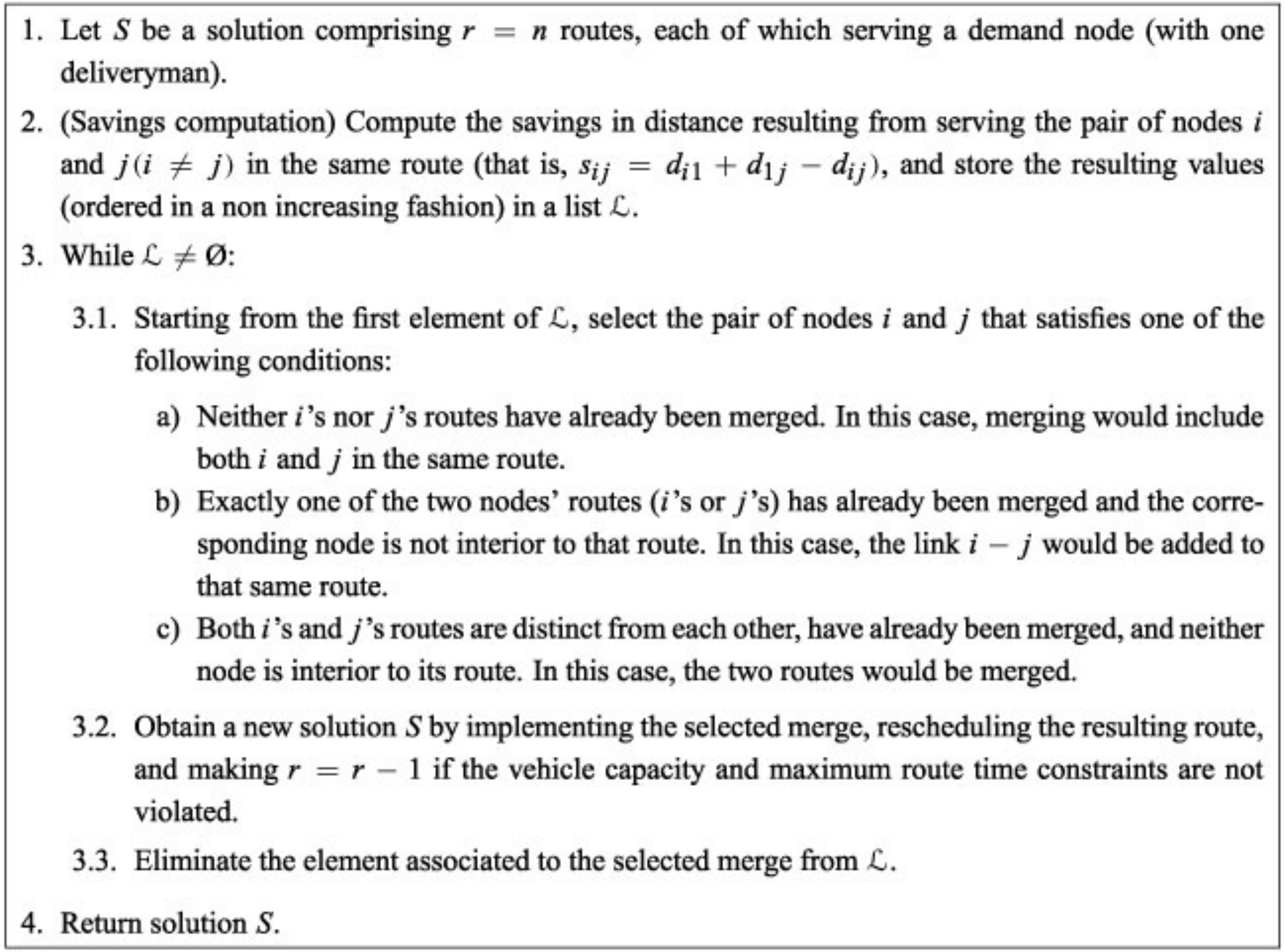
- Partition first:
  - Build one group of nodes per vehicle (clustering)
  - Solve the TSP on each group

- Sequence first:
  - Relax the vehicle capacity and solve a *giant* TSP
  - Split the giant TSP into trip satisfying the capacity constraints.

Partition vs Sequence first



VRP Initialization: Saving Heuristics [Clarke and Wright 1964]

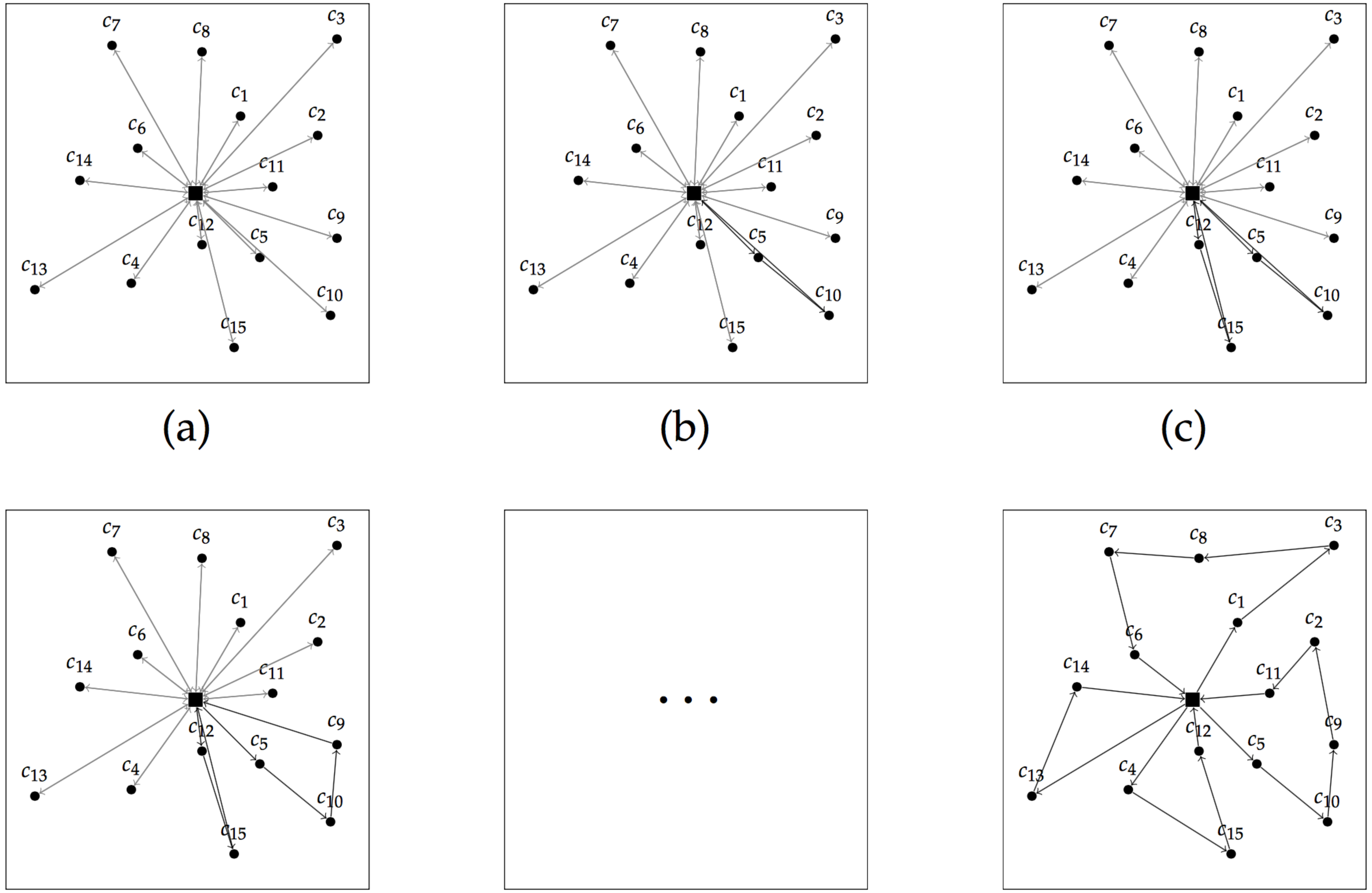


1. Let S be a solution comprising $r = n$ routes, each of which serving a demand node (with one deliveryman).
2. (Savings computation) Compute the savings in distance resulting from serving the pair of nodes i and $j (i \neq j)$ in the same route (that is, $s_{ij} = d_{i1} + d_{1j} - d_{ij}$), and store the resulting values (ordered in a non increasing fashion) in a list $\mathcal{L}$.
3. While $\mathcal{L} \neq \emptyset$:
    - 3.1. Starting from the first element of $\mathcal{L}$, select the pair of nodes i and j that satisfies one of the following conditions:
        - a) Neither i's nor j's routes have already been merged. In this case, merging would include both i and j in the same route.
        - b) Exactly one of the two nodes' routes (i's or j's) has already been merged and the corresponding node is not interior to that route. In this case, the link $i - j$ would be added to that same route.
        - c) Both i's and j's routes are distinct from each other, have already been merged, and neither node is interior to its route. In this case, the two routes would be merged.
    - 3.2. Obtain a new solution S by implementing the selected merge, rescheduling the resulting route, and making $r = r - 1$ if the vehicle capacity and maximum route time constraints are not violated.
    - 3.3. Eliminate the element associated to the selected merge from $\mathcal{L}$.
4. Return solution S.

Figure 1 – Sequential savings algorithm (SAV).

VRP Initialization: Saving Heuristics [Clarke and Wright 1964]

- Start with as many vehicles as the number of customers
- Insert customers and merge routes such that the capa is not violated and the distance is decreased the most

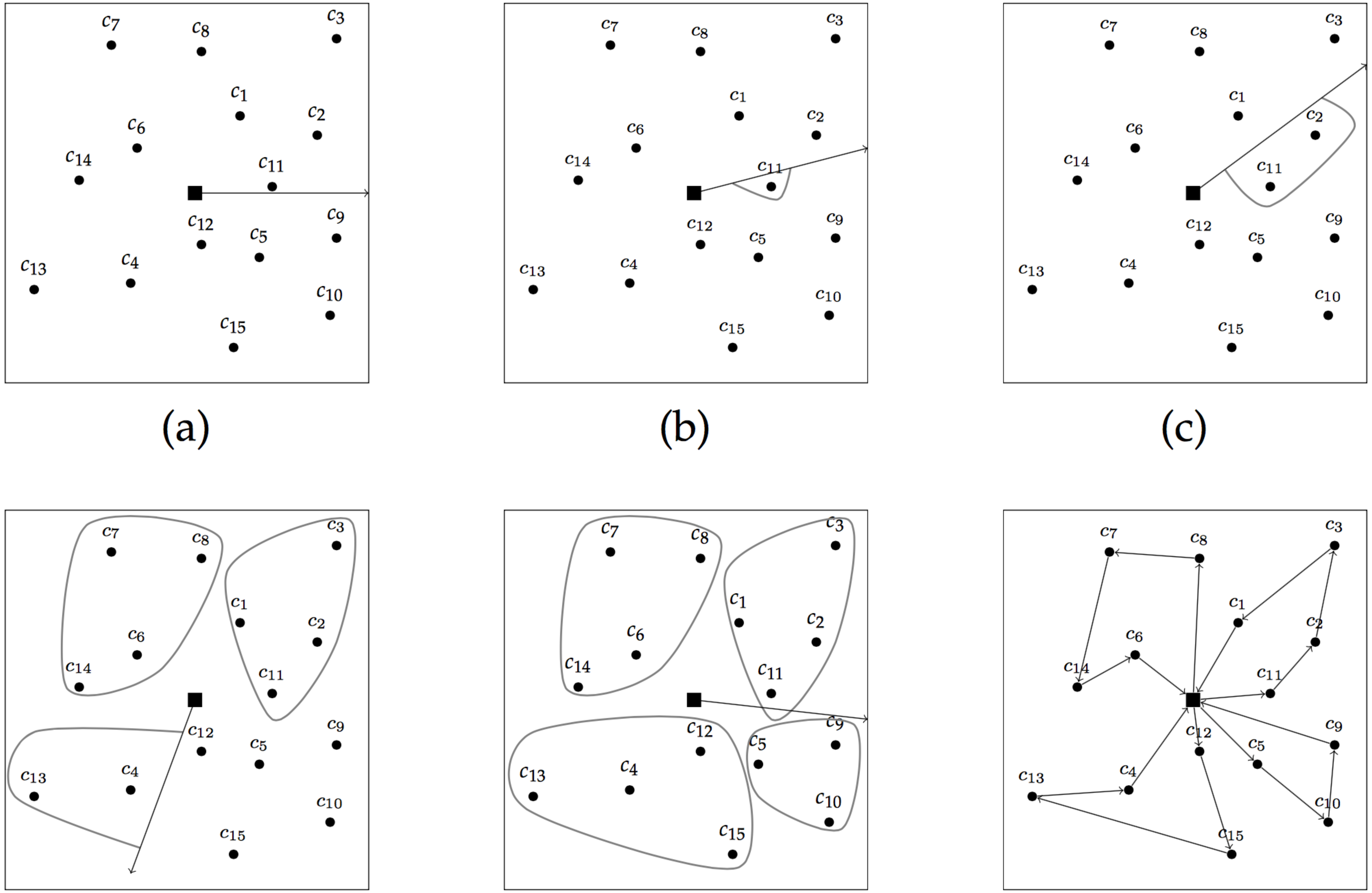


(a) (b) (c)

(d) (e) (f)

VRP Initialization: Sweep Heuristic

- A ray centered at the depot performs a full rotation, collecting customers into clusters not violating the capacity



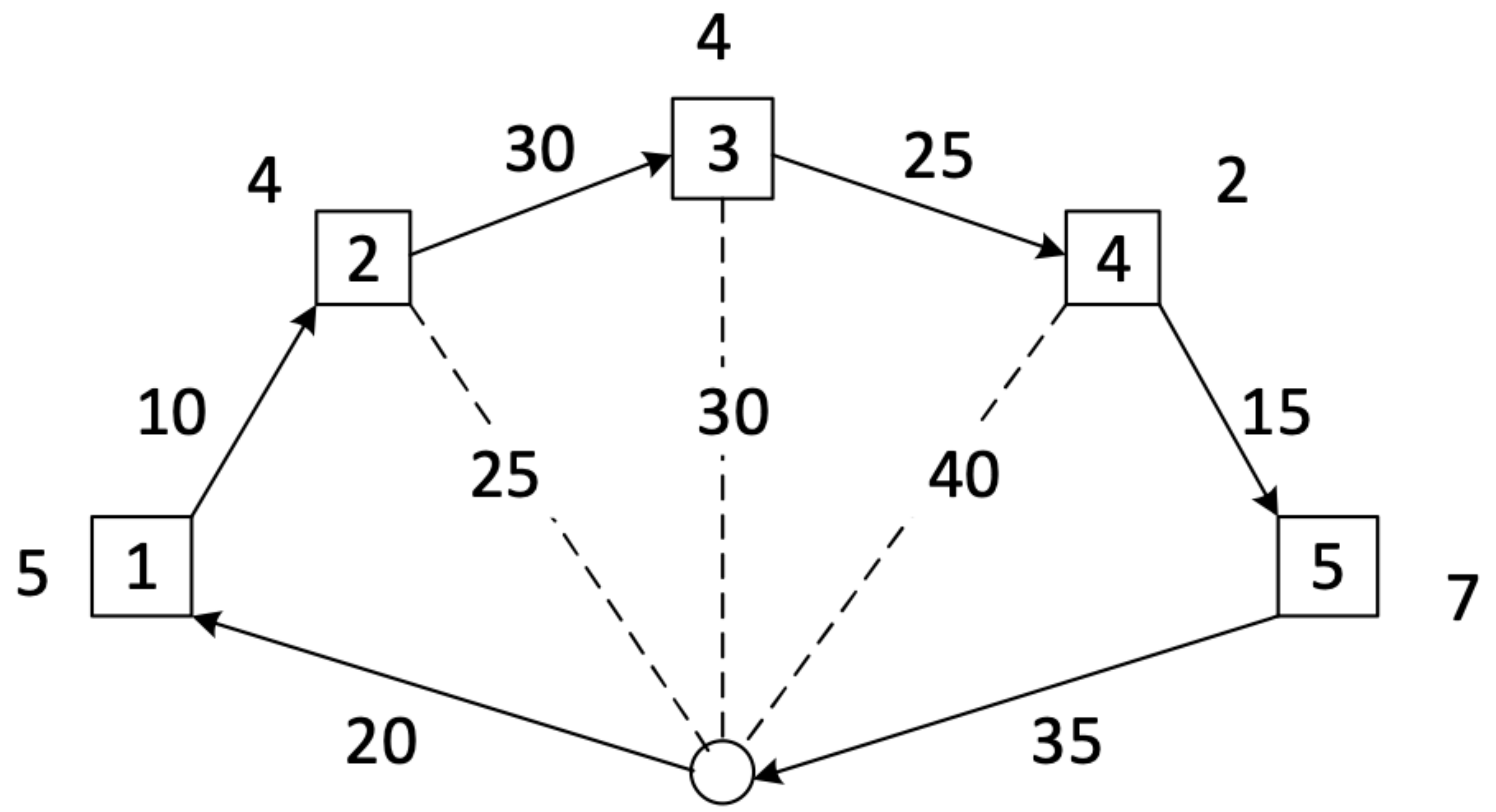
(a) (b) (c)

(d) (e) (f)

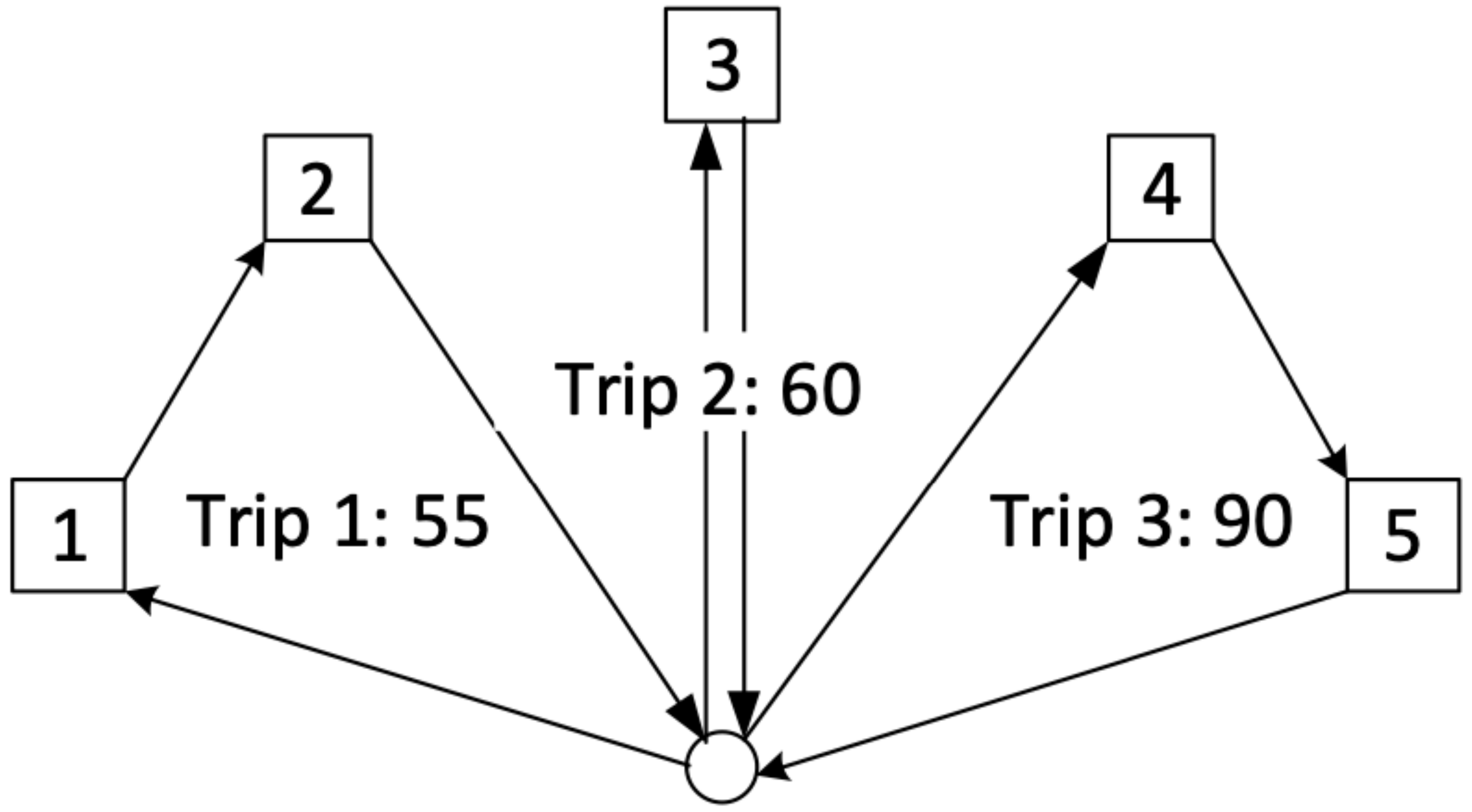
Split Heuristic

Initial TSP Giant Tour that violates the capacity 10 for one truck

How to discover this split optimally ?

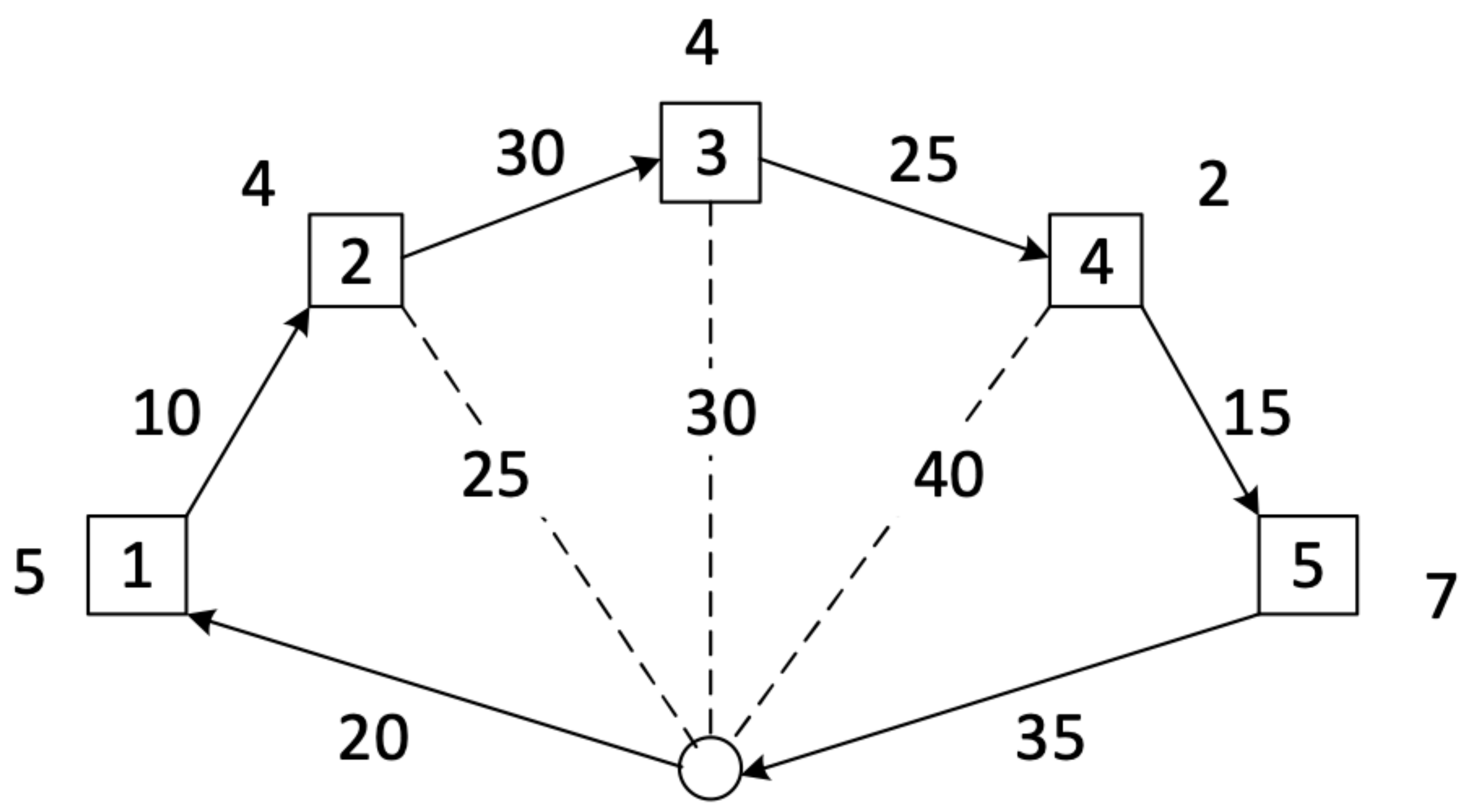


1. Giant tour *T* = (1, 2, 3, 4, 5) with demands

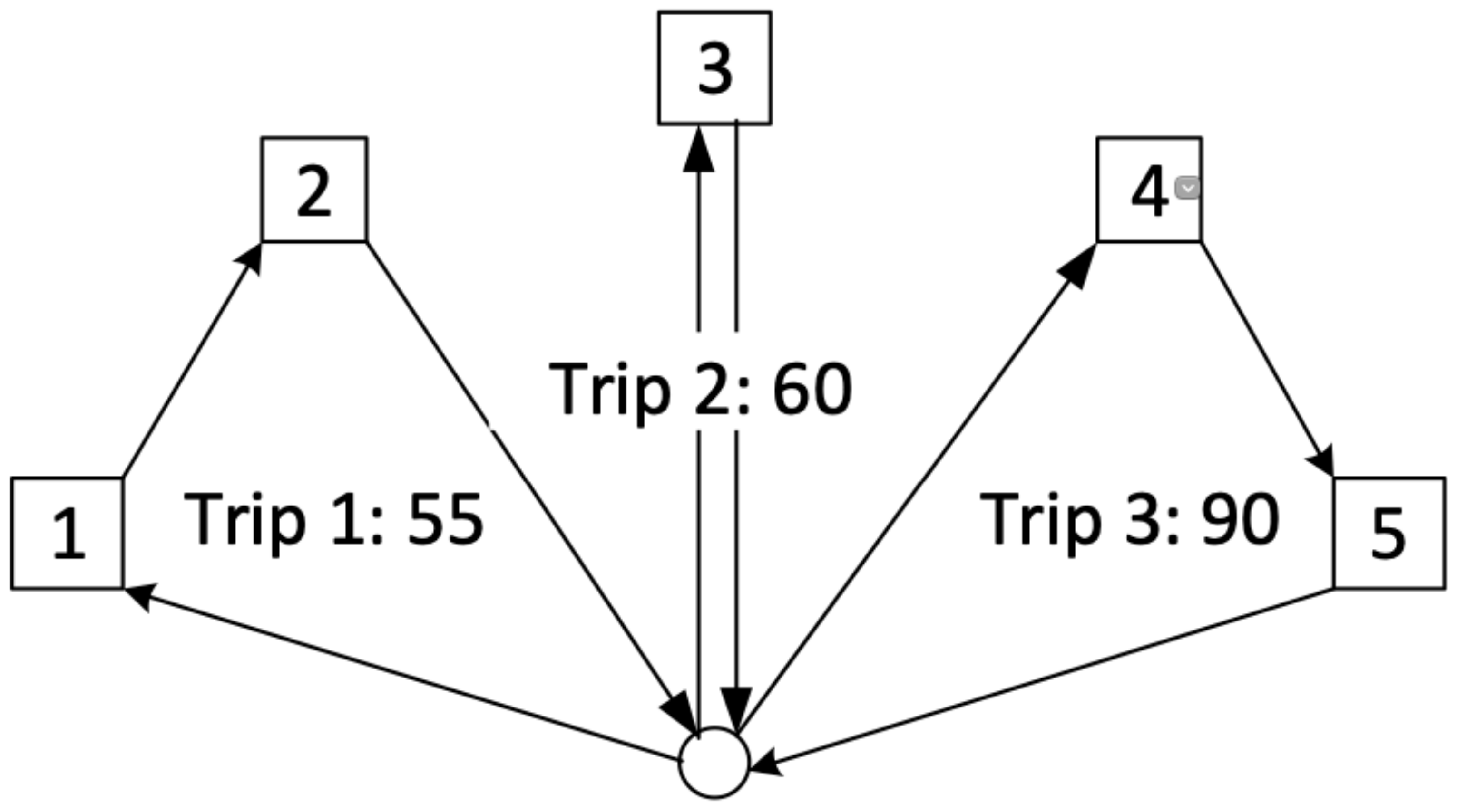


3. Optimal splitting, cost 205

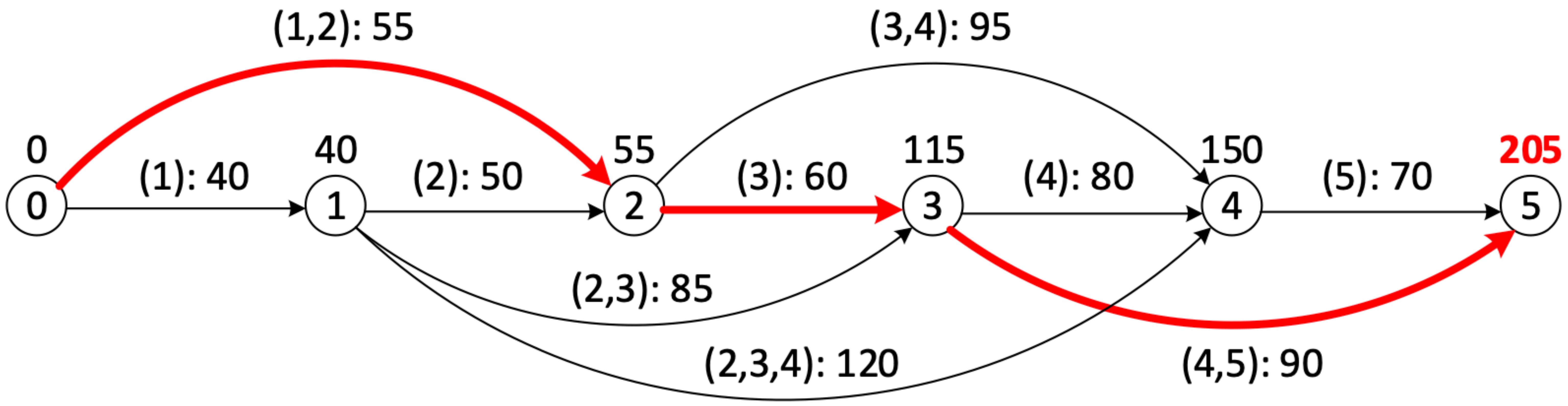
Answer: Dynamic Programming



1. Giant tour *T* = (1, 2, 3, 4, 5) with demands



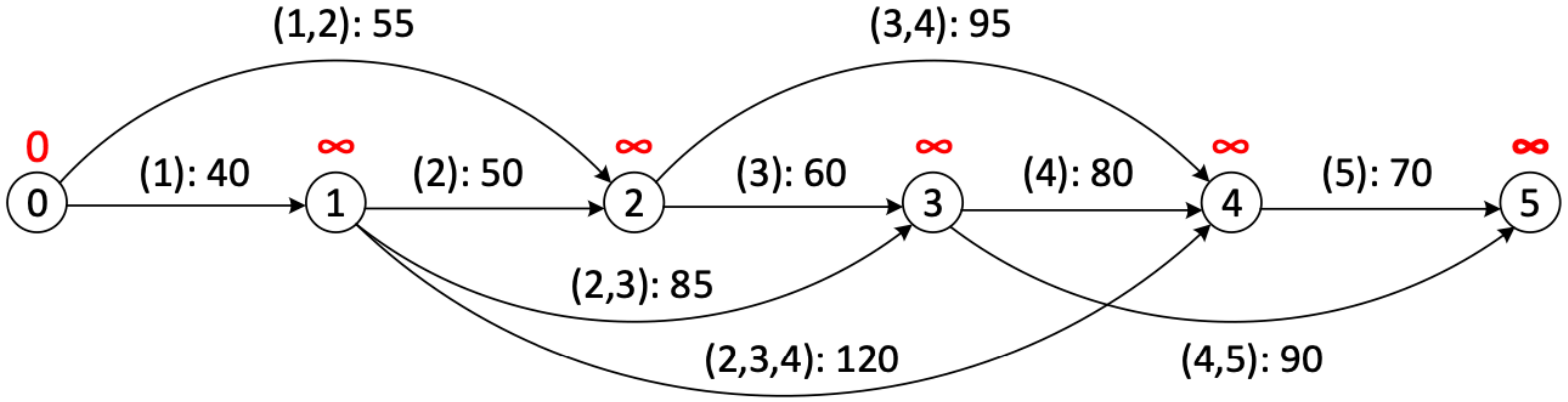
3. Optimal splitting, cost 205

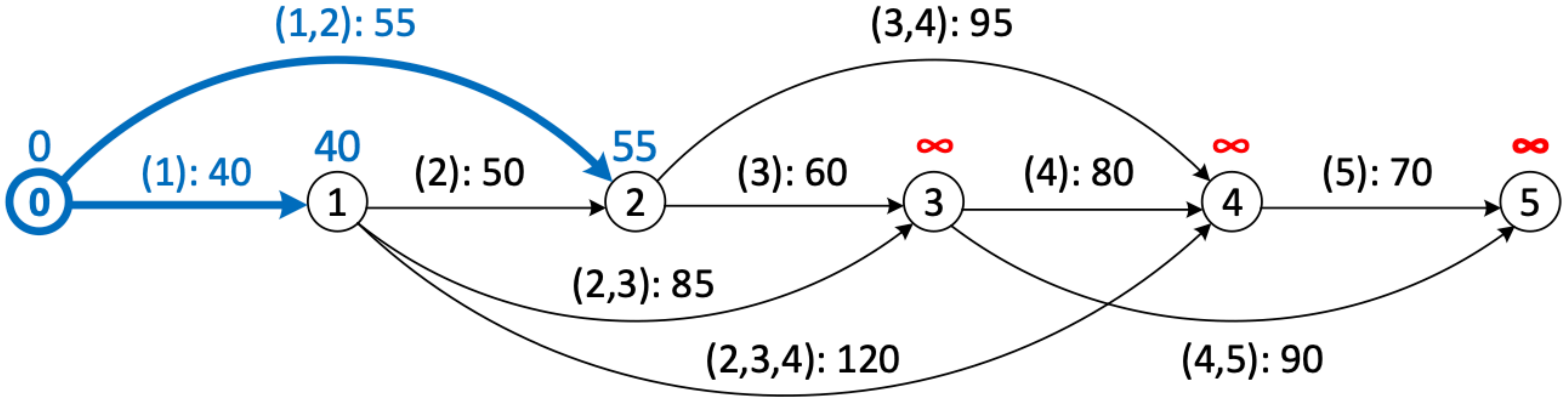


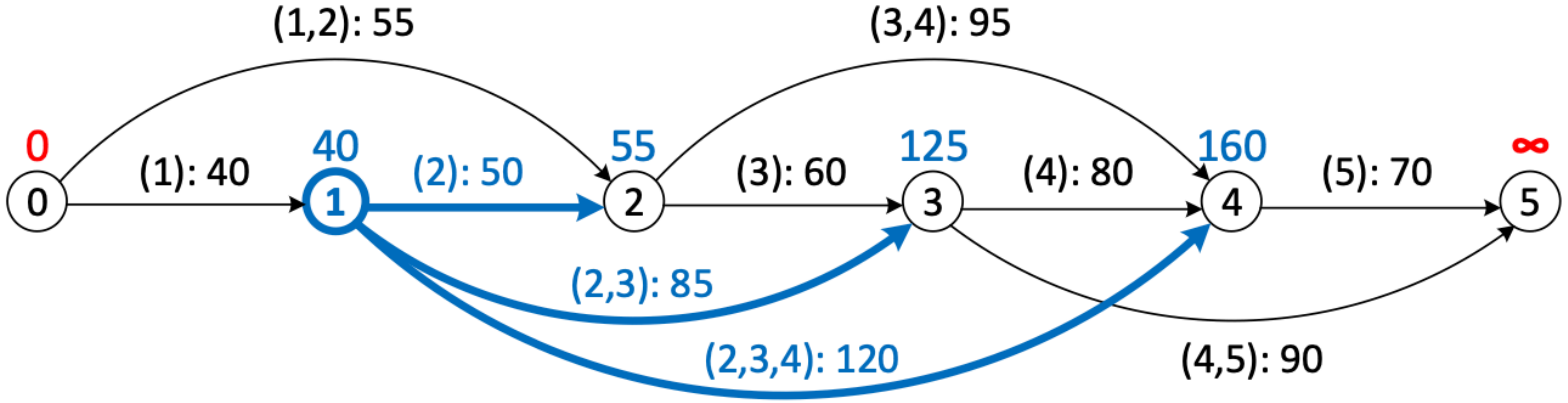
2. Auxiliary graph *H* of possible trips for *Q* = 10 – Shortest path in bold

Prins, C., Labadi, N., & Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, *47*(2), 507-535.

Shortest Path: Bellman Algo







Implementation

Auxiliary graph $H = (X, A, Z)$ with $n + 1$ nodes numbered rom 0.
A feasible route $(T_{i+1}, \dots, T_j)$ is modelled by arc $(i - 1, j)$.

Bellman's algorithm for directed acyclic graphs (DAGs).
Compact form with implicit auxiliary graph (Prins, 2004):

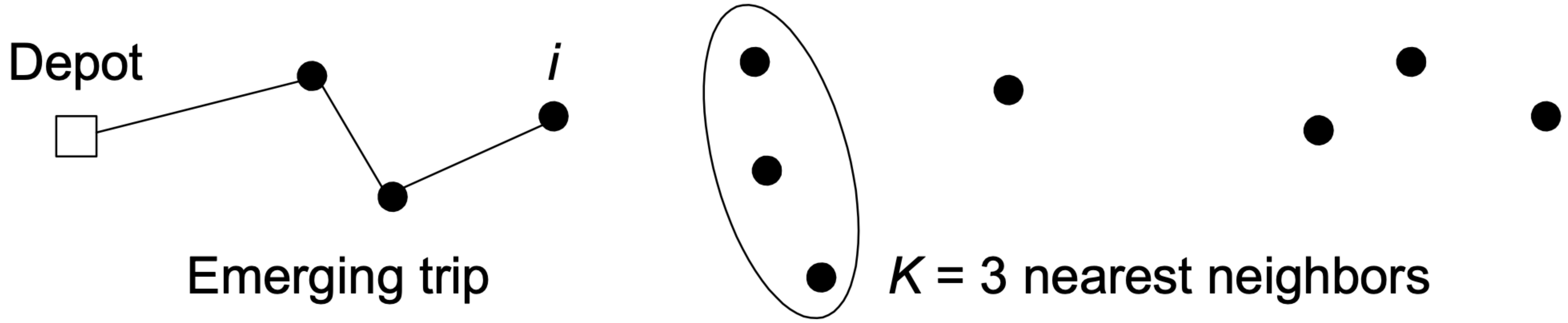
```
set $V_0$ to 0 and other labels $V_i$ to $\infty$ (cost of path to node $i$)
for $i \leftarrow 1$ to $n$ do
   for $j \leftarrow i$ to $n$ while subsequence/route $(T_i, T_{i+1}, \dots, T_j)$ feasible
      compute route cost, i.e., cost $z_{i-1,j}$ of arc $(i - 1, j)$
      if $V_{i-1} + z_{i-1,j} < V_j$ then
         $V_j \leftarrow V_{i-1} + z_{i-1,j}$
      endif
    endfor
endfor
```

Remark

- *The giant tour T can be built using any TSP algorithm.*
- *Optimal TSP tours do not necessarily lead to optimal CVRP solutions after splitting, good tours are enough.*
- *However, Split is optimal, subject to the ordering of T*
- *O(n^2) time complexity overall (capacity check in O(1))*

Initialiaztion with randomised TSP Giant tour

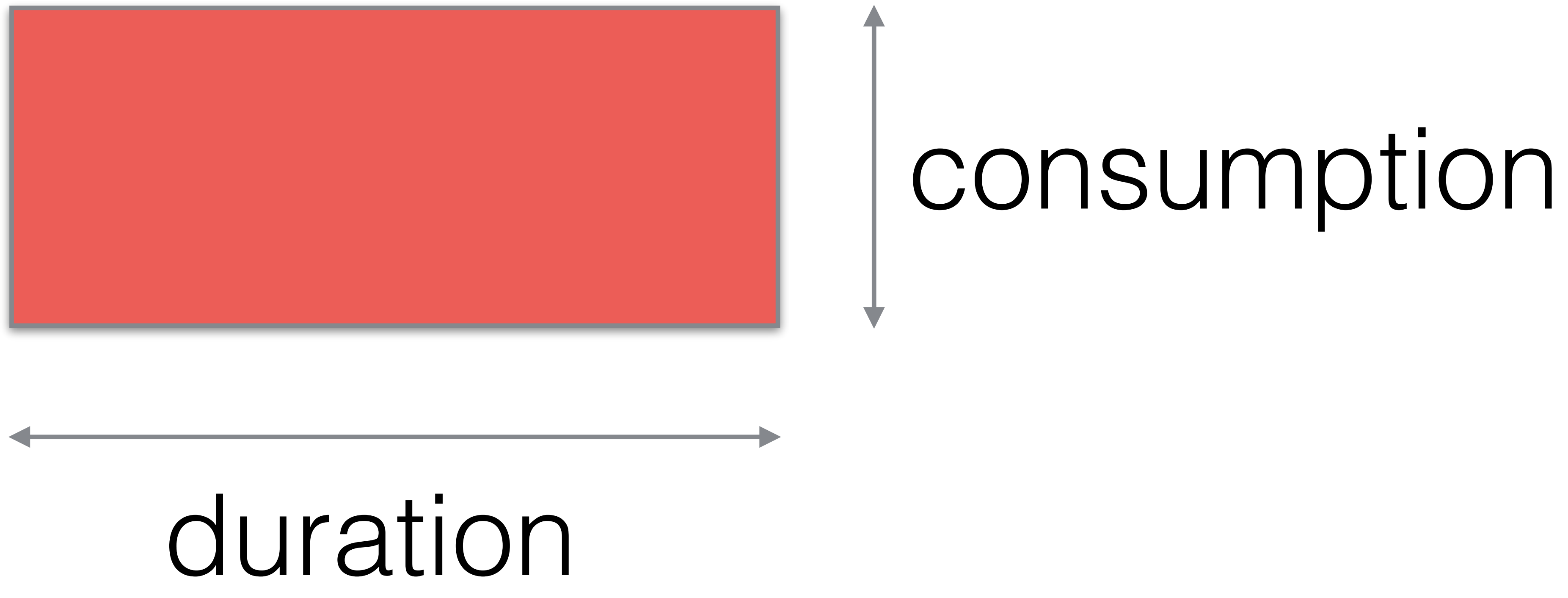
Nearest Neighbor heuristic (NN), well known for the TSP.
Randomized version:

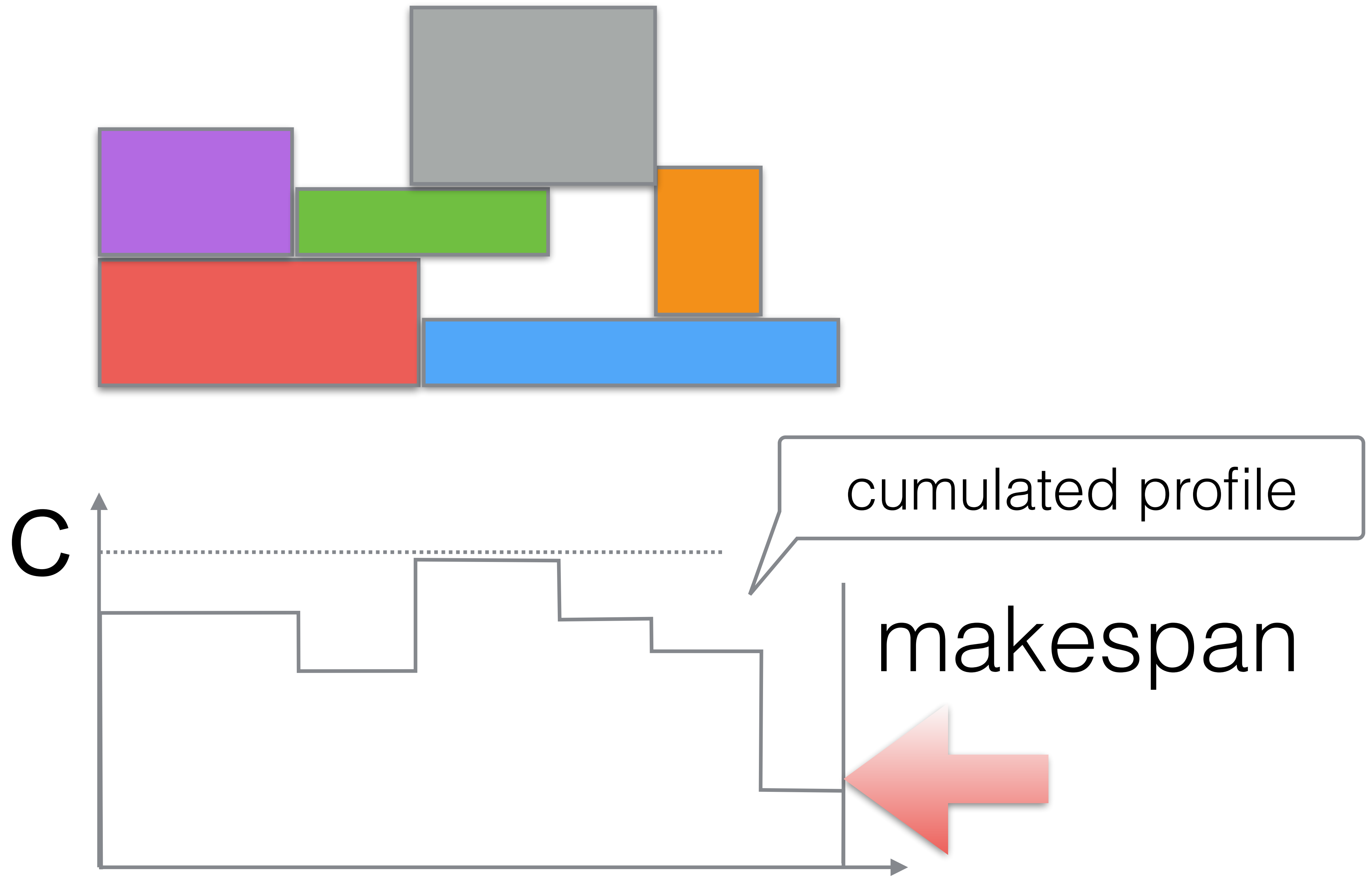


Draw the next client *j* among the *K* nearest ones.

Scheduling Moves

- Given a set of non preemptive activities (cannot be interrupted)
- A resource with capa C
- How to schedule them to minimize the total duration (makespan) without exceeding the capacity of the resource?



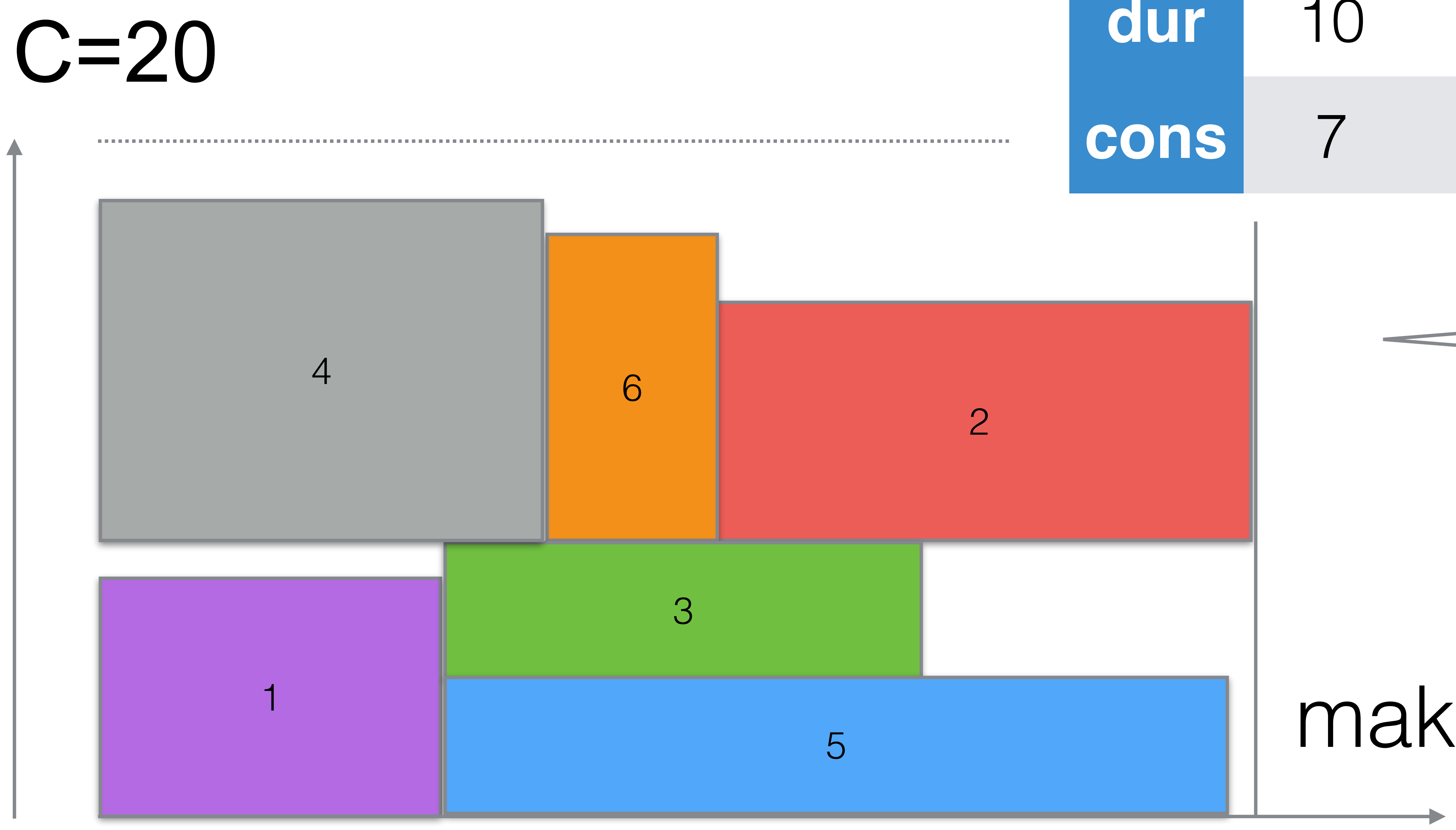


Scheduling with capa: IFlat-IRelax algorithm

- Iterate between two steps:
  1. flatten = add strong precedences constraints until the capacity constraint is satisfied (assuming each activity starts as soon as possible while satisfying the precedence constraints)
  2. relax = remove some precedences randomly on the critical path

Example

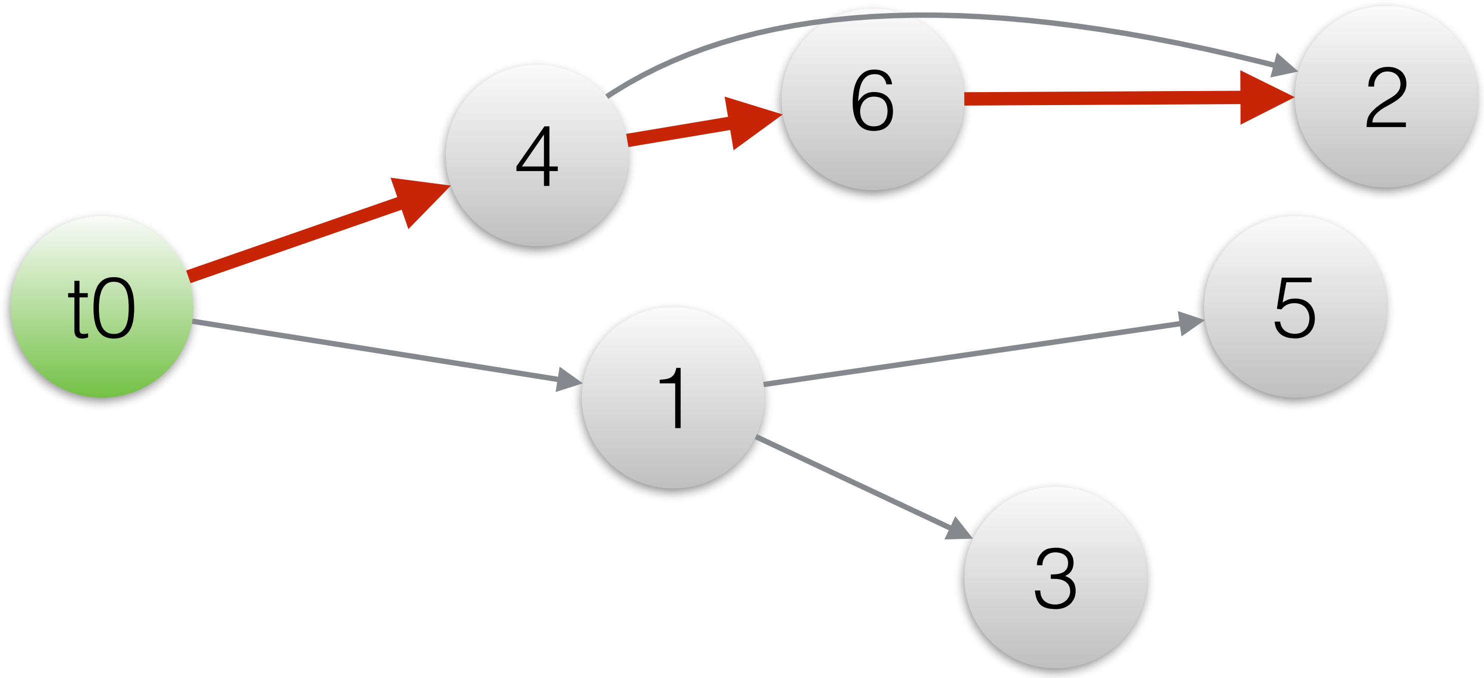
	1	2	3	4	5	6
dur	10	16	14	13	21	5
cons	7	7	4	10	4	9



each activity scheduled at its earliest start while satisfying the precedences (dynamic programming)

makespan

current precedence graph

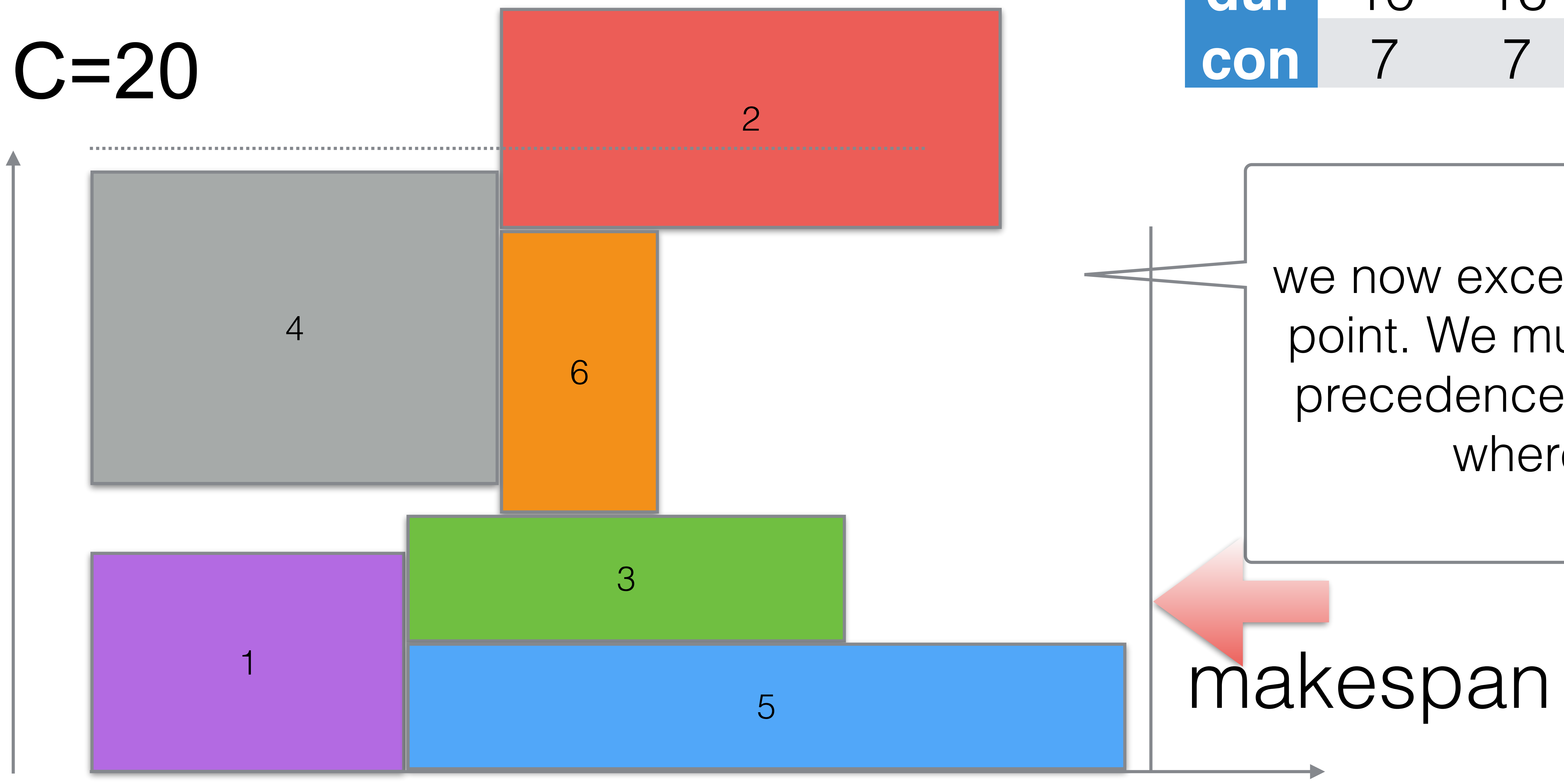


critical path = the path that causes the makespan value

to have a chance to decrease the makespan we have to relax precedences on the critical path (say we remove 6->2)

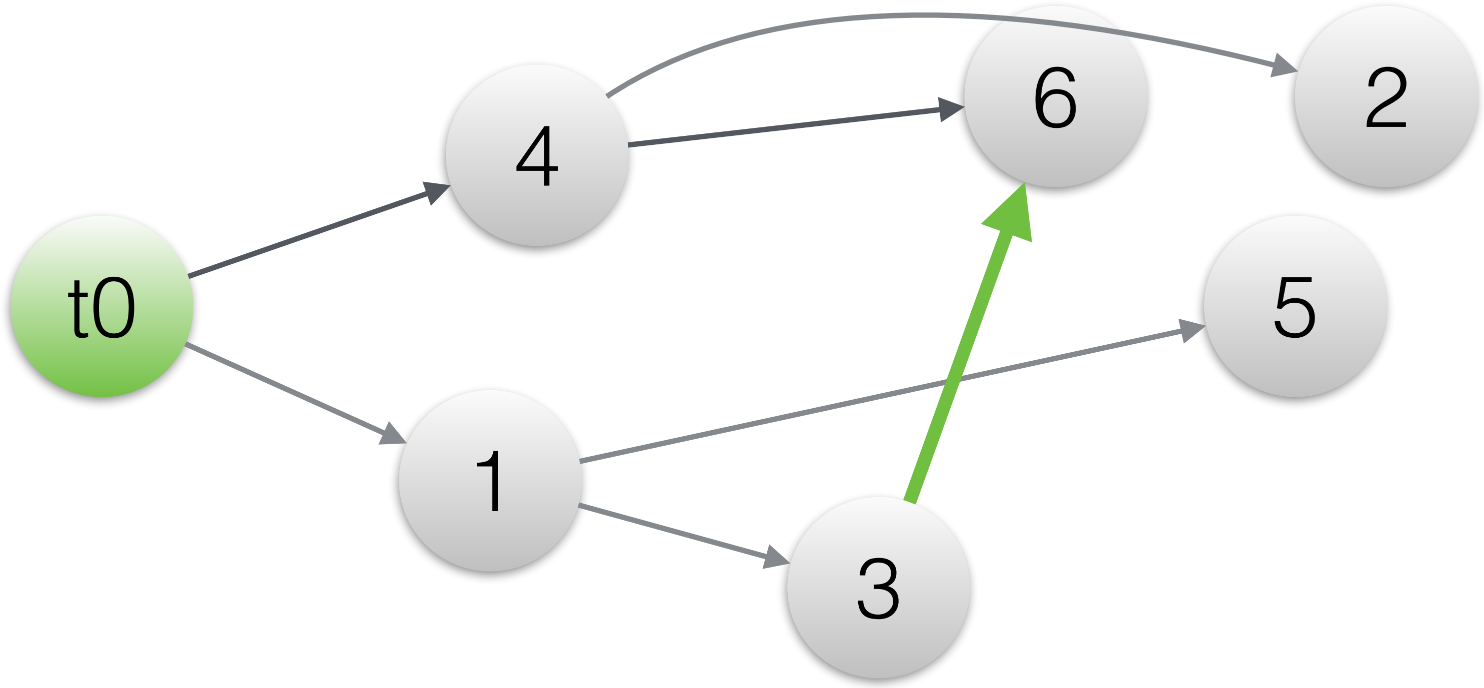
Example

	1	2	3	4	5	6
dur	10	16	14	13	21	5
con	7	7	4	10	4	9



we now exceed the capa at some point. We must flatten by adding precedences between activities where we exceed

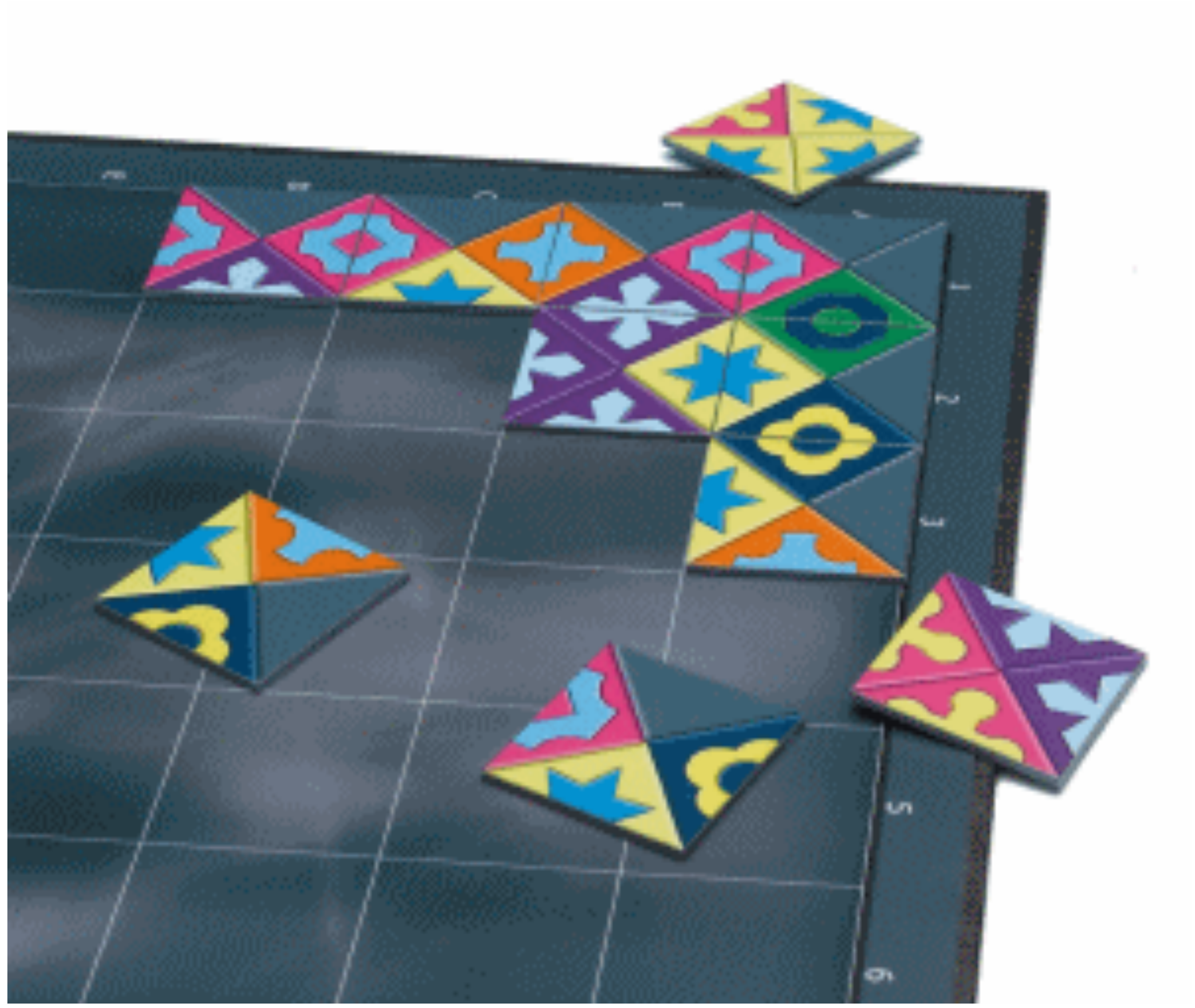
current precedence graph



critical path = the path that causes the makespan value

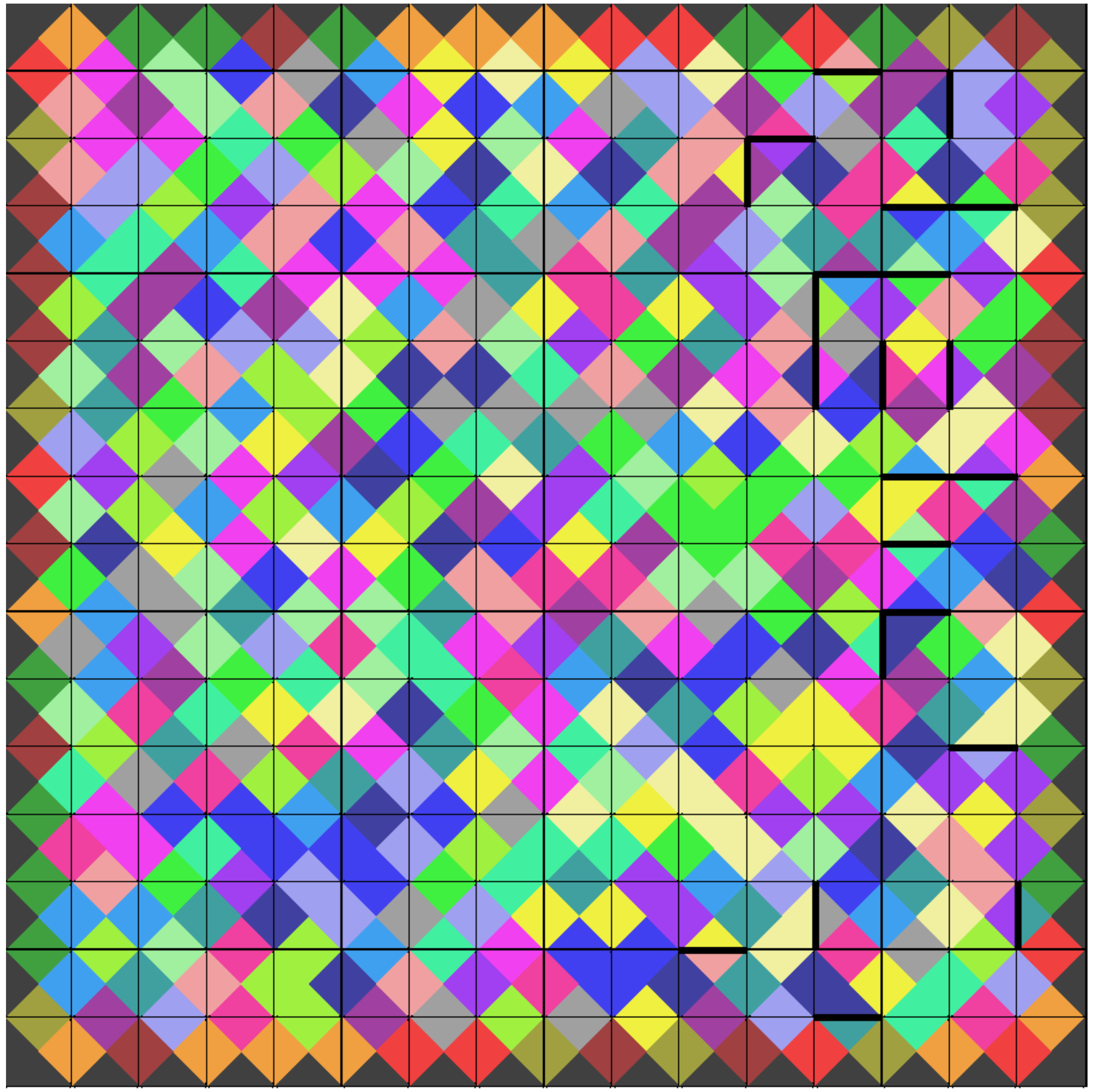
Eternity II

- 16x16 edge matching puzzle
- 2$ millions if you solve it …



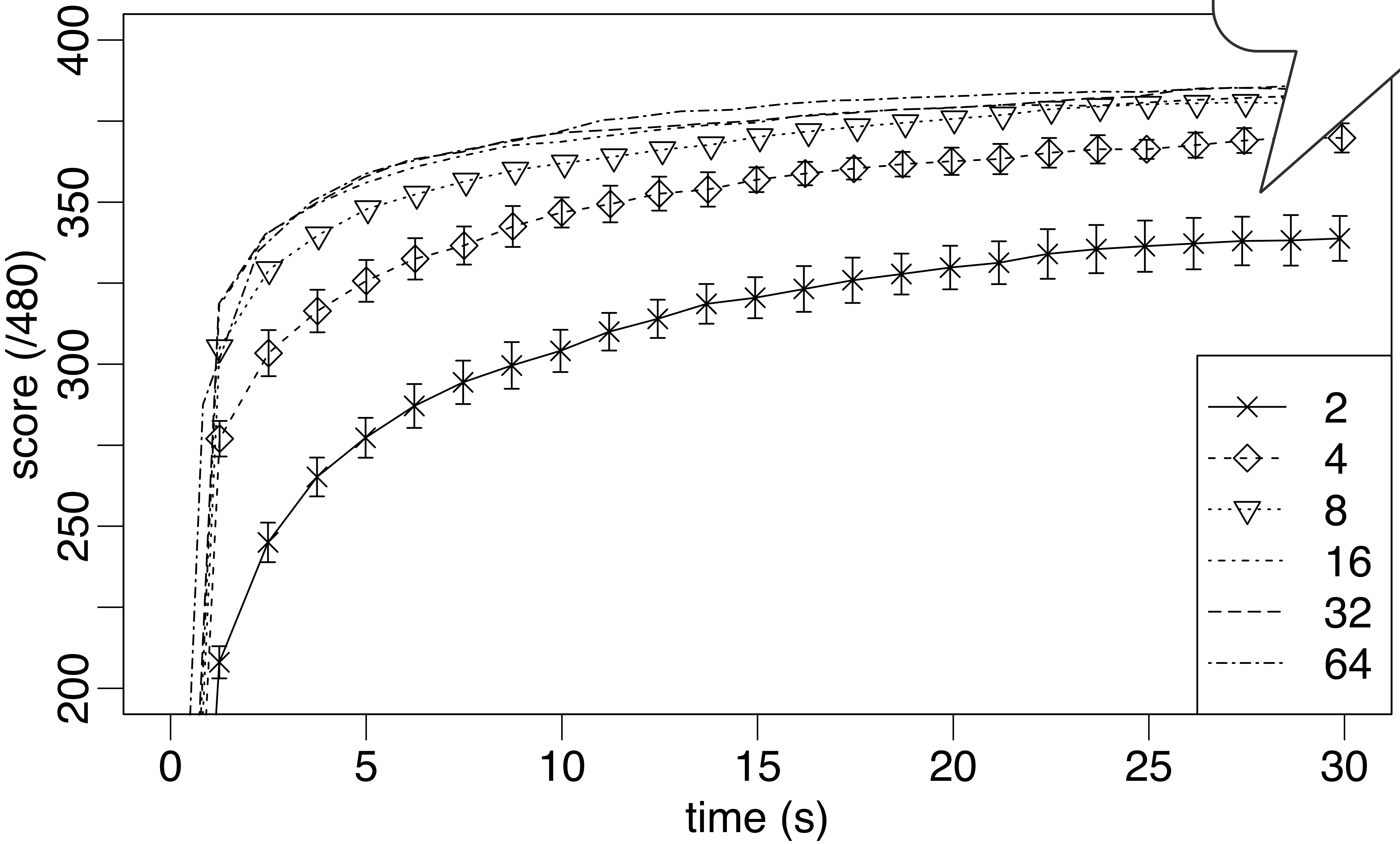
What do you suggest as neighborhood?

Objective: maximize # correct connections (480)



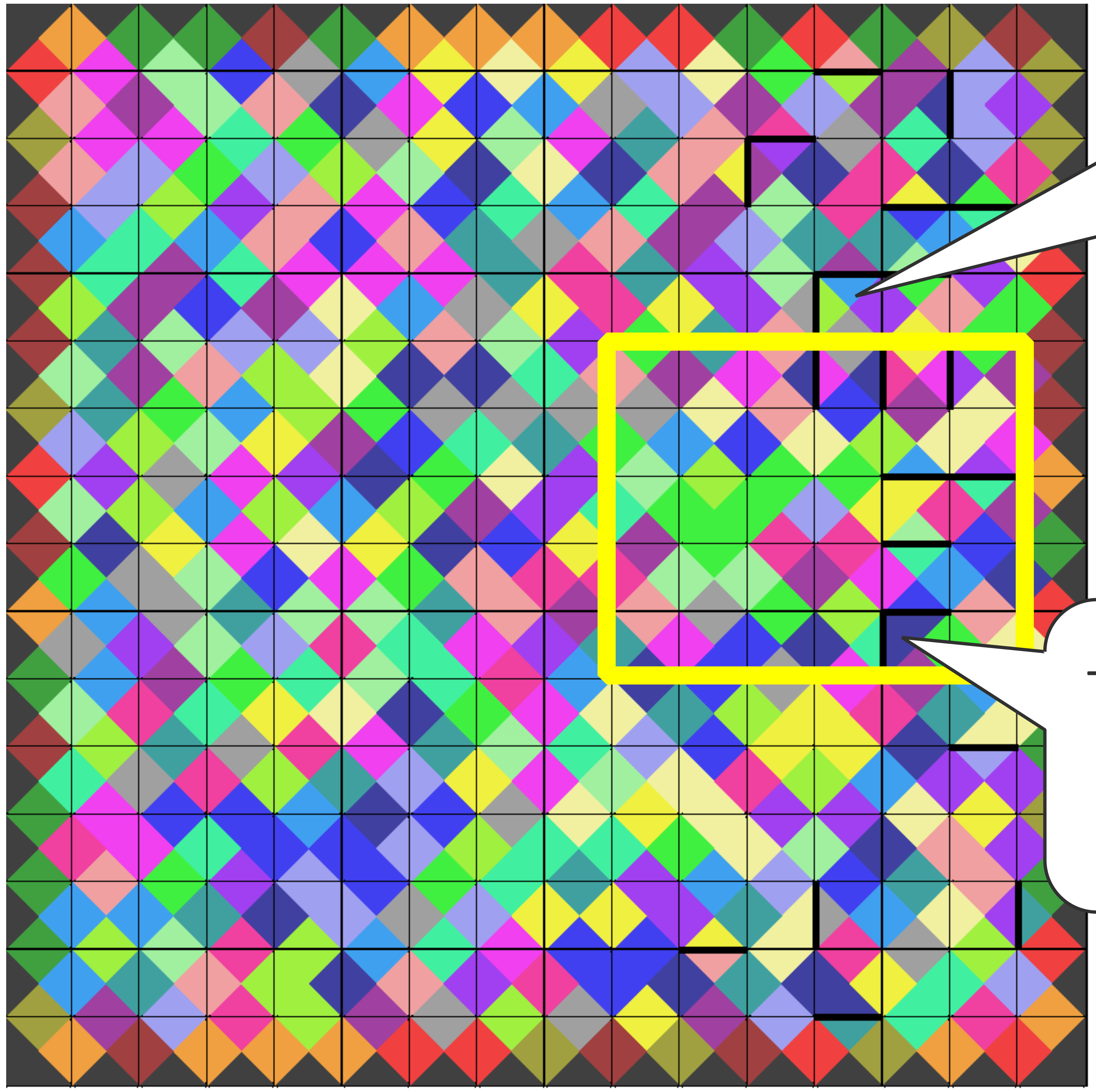
What do you suggest as neighborhood?

Swap and rotate pieces pairwise ?



You will not go much higher than 320

Let’s try to move >2 pieces at once



Try to improve optimize a full (small) region?

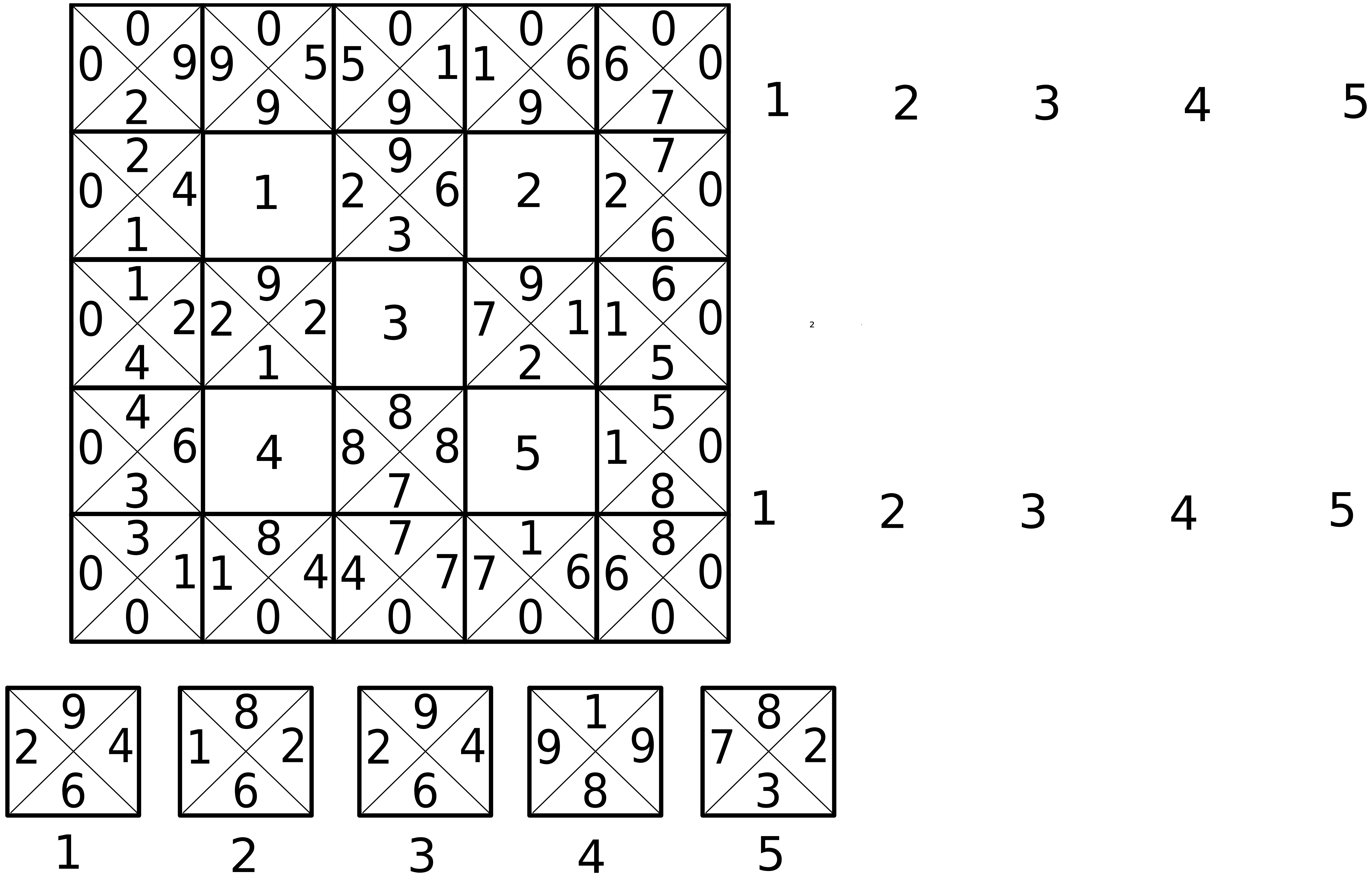
Too computational intensive (same intrinsic difficulty as original problem)

Generalization of Swap and Rotate

- Remove m pieces from **non edge adjacent** positions (up to $n^2/2$ = chessboard)..
- Replace them optimally.
- This neighborhood can be solved optimally and efficiently

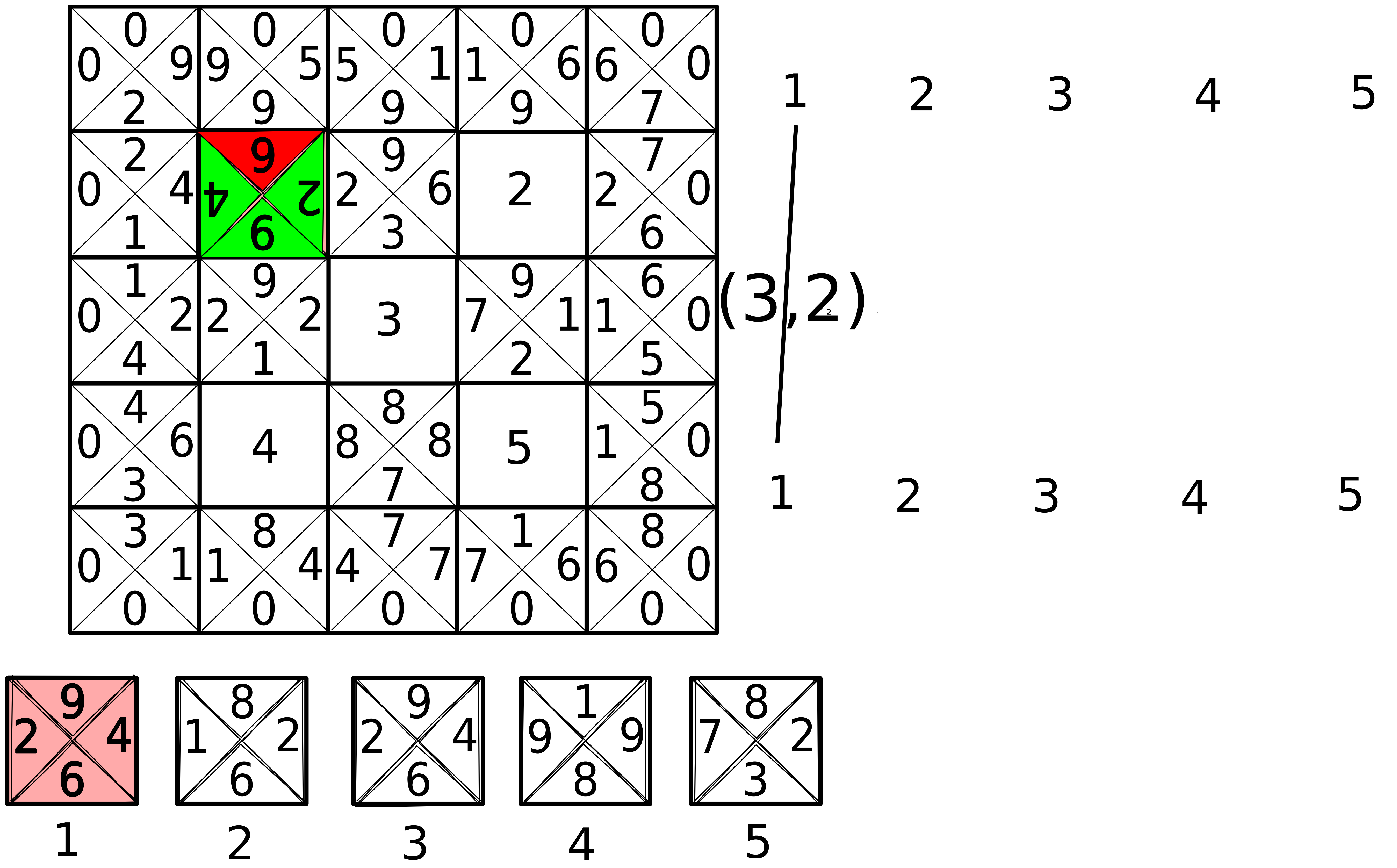
Eternity VLNS

- Let’s remove 5 non adjacent pieces



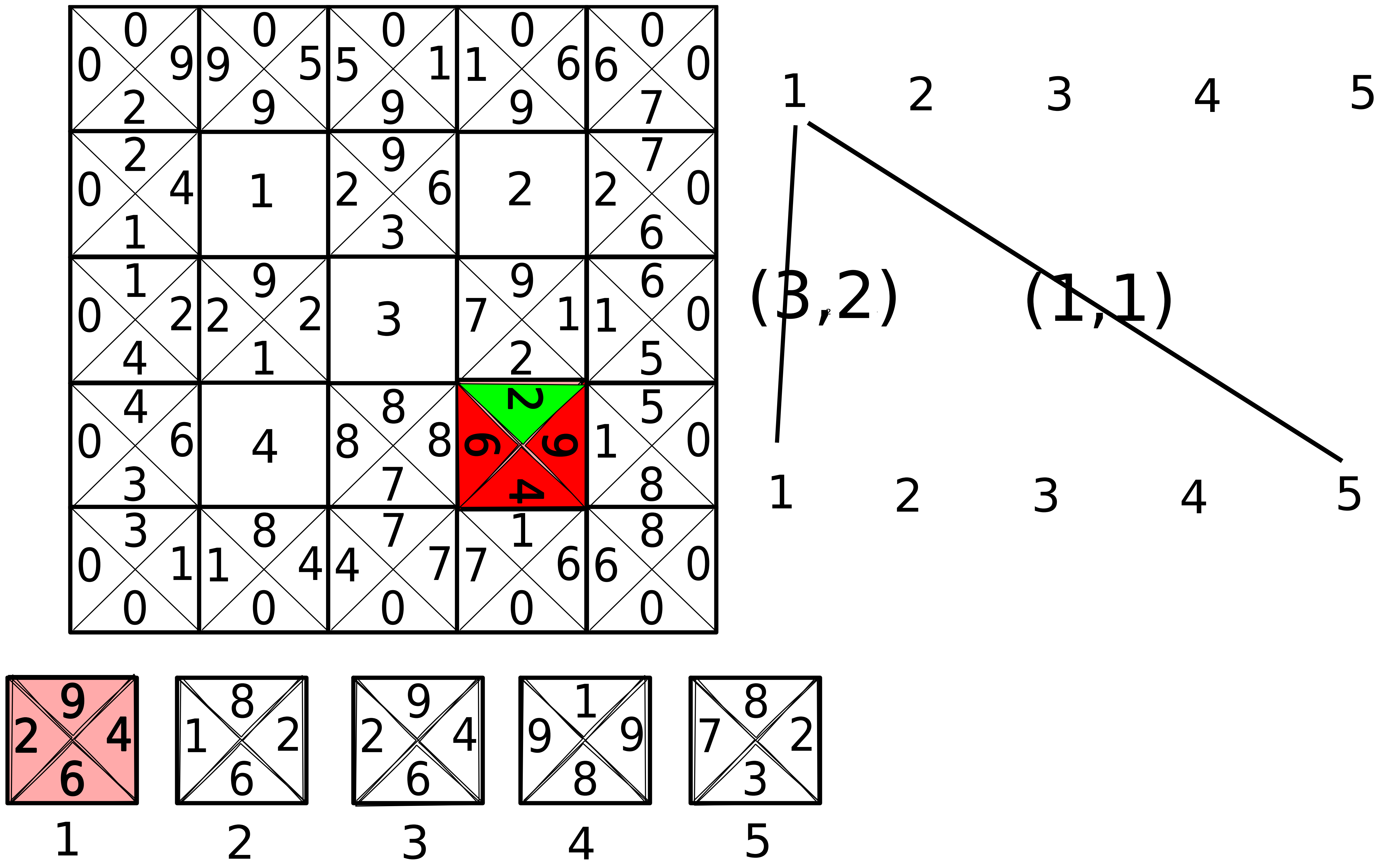
Eternity VLNS

- Let’s remove 5 non adjacent pieces



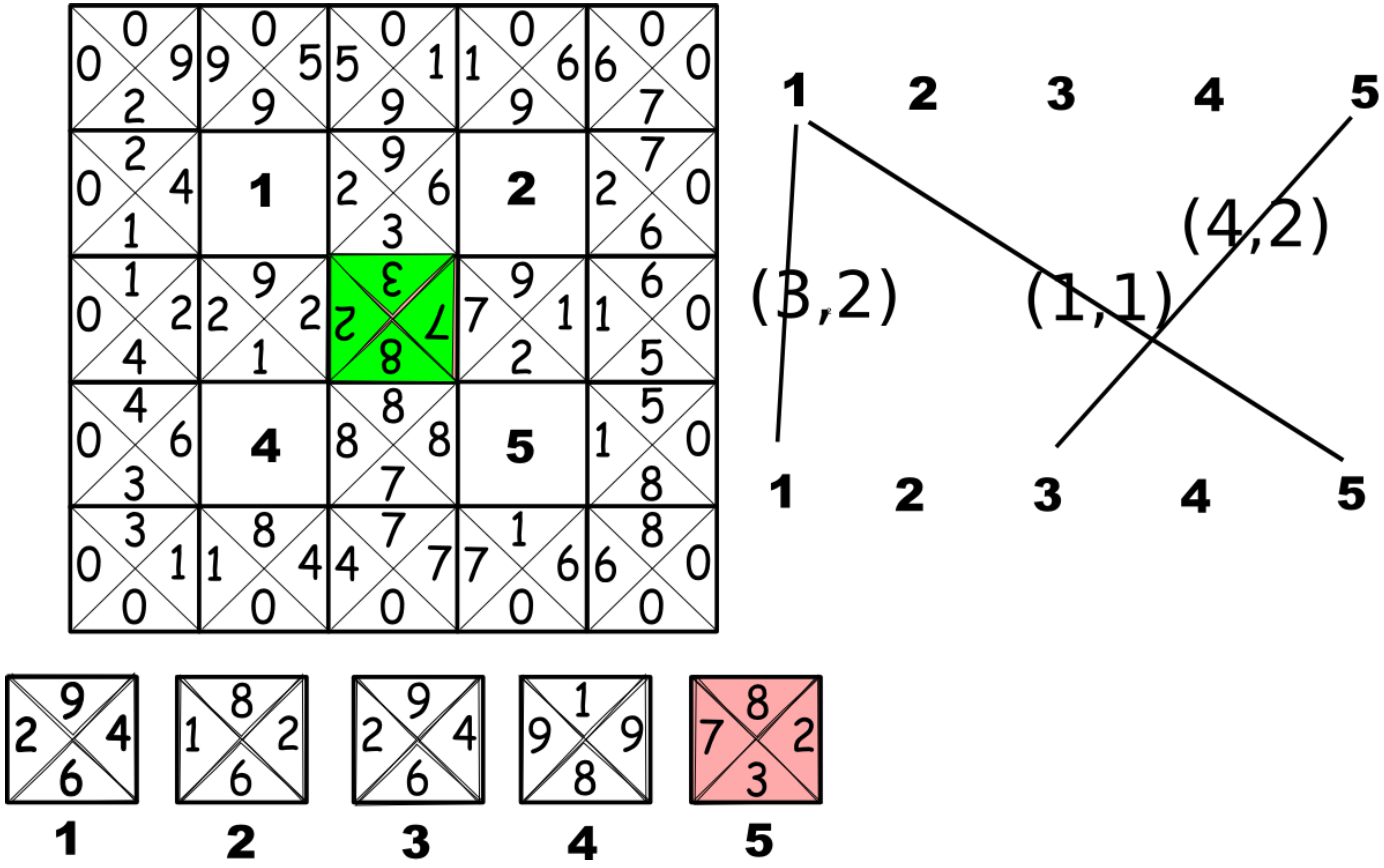
Eternity VLNS

- Compute score to place them each optimally in holes



Eternity VLNS

- Compute score to place them each optimally in holes



Eternity VLNS

- At the end, a complete weighted bipartite graph between removed pieces and holes.

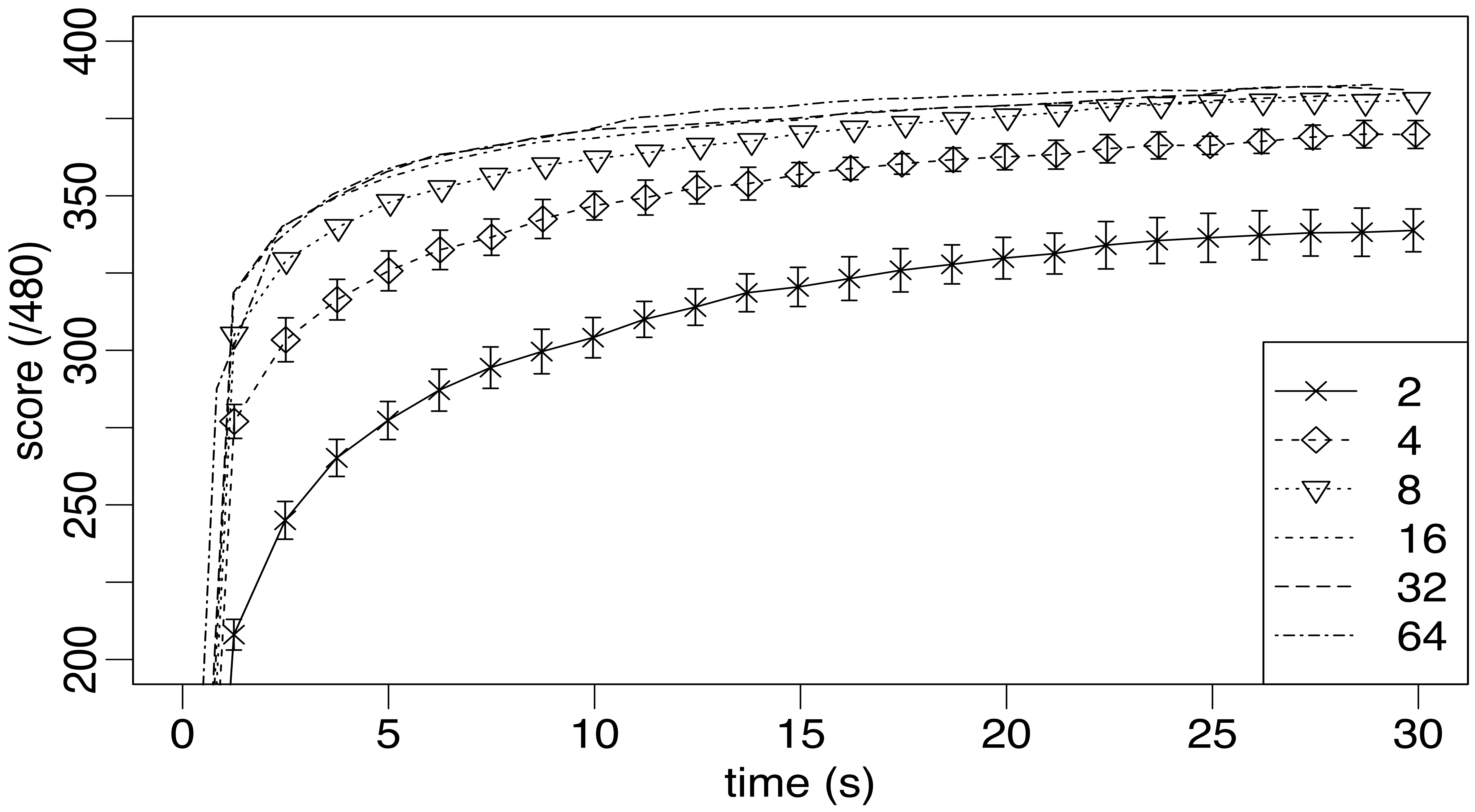
- solve a maximum assignment problem (Hungarian algorithm in $O(m^3)$).

- The arcs in the assignment and the label of the arcs tell us how to replace optimally the pieces in the holes.

- Neighborhood of size: $m!4^m$ (exponential) but it is optimally explored in polynomial time.

Effect of neighborhood size

- Each step, m non-adjacent positions are randomly chosen and the move is applied. During 30 seconds. Random initial positions of the pieces on the board.



Exam Questions

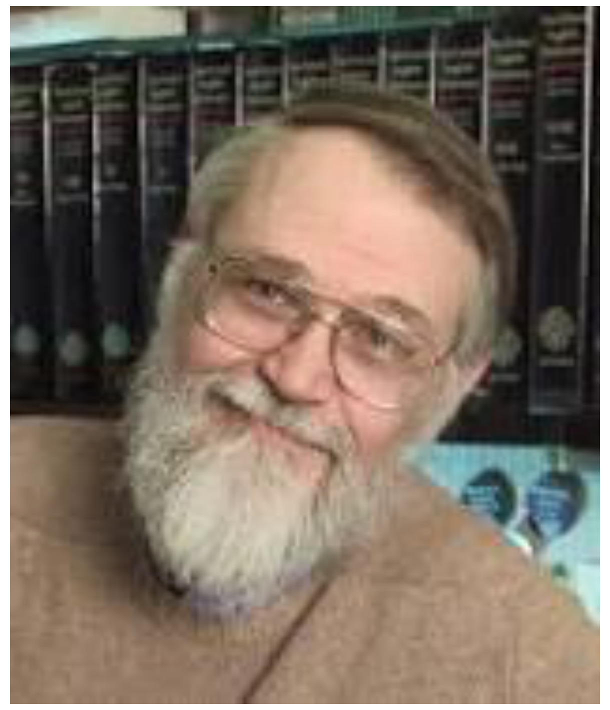
- Be able to explain the principle of local search
- Be able to implement a simple search with swap-moves and a tabu-search meta-heuristic
- Be able to suggest a neighborhood for a new problem and discuss/prove it it is connected or not.
- Be able to explain and apply moves for:
    - TSP (Lin-Kernighan) and vehicle routing,
    - scheduling,
    - eternity

Bibliography

- Implementing the Lin-Kernighan heuristic for the TSP, Markus Reuther
- General k-opt submoves for the Lin–Kernighan TSP heuristic, Keld Helsgaun
- Optimization approaches for the vip’s with black box feasibility. Florence Massen. PhD thesis 2013.
- In pursuit of the traveling salesman, William J. Cook, 2012.
- Constrained Based Local Search. P. Van Hentenryck and L. Michel. 2006.
- Hybrization of CP and VLNS for eternity II. P Schaus and Yves Deville.
- Iterative Relaxations for Iterative Flattening in Cumulative Scheduling. P. Van Hentenryck and L. Michel. 2006.
- Prins, C., Labadi, N., & Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, *47*(2), 507-535.

Inventors

Brian Wilson Kernighan



1942

also coauthor of the AWK and AMPL programming languages

Fred Glover

1937

Tabu Search meta-heuristic