University of Antwerp
Operations Research Group
ANT/OR

Practical recommendations for an efficient local search implementation: a CVRP case study

Daniel Palhazi Cuervo, Kenneth Sörensen and Peter Goos

EU/MEeting 2012 - Copenhagen, May 11th 2012

Outline

- Design phase
  - Dealing with constraints
  - Consideration of infeasible solutions
- Implementation phase
  - Neighbourhood calculation and update
  - Data structure (solution representation)
- Final remarks

Dealing with constraints

Alternatives:

- Discard infeasible solutions
  - Continual checking of constraint violations
- Penalize the cost function $C(s) = d(s) + \alpha I(s)$
  - Need of a measure to quantify infeasibility

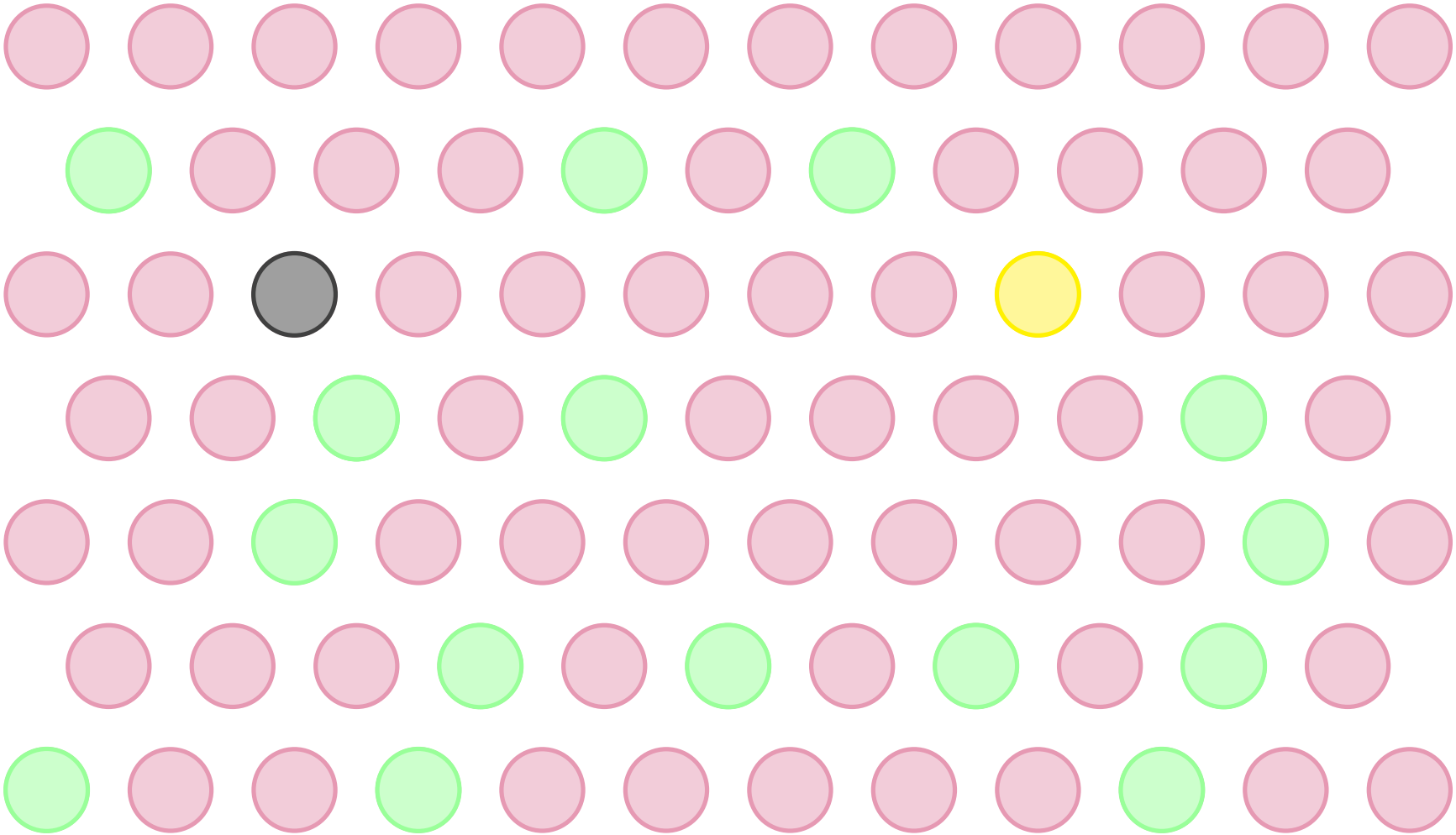
Only feasible

- Fixed α value (usually very large)

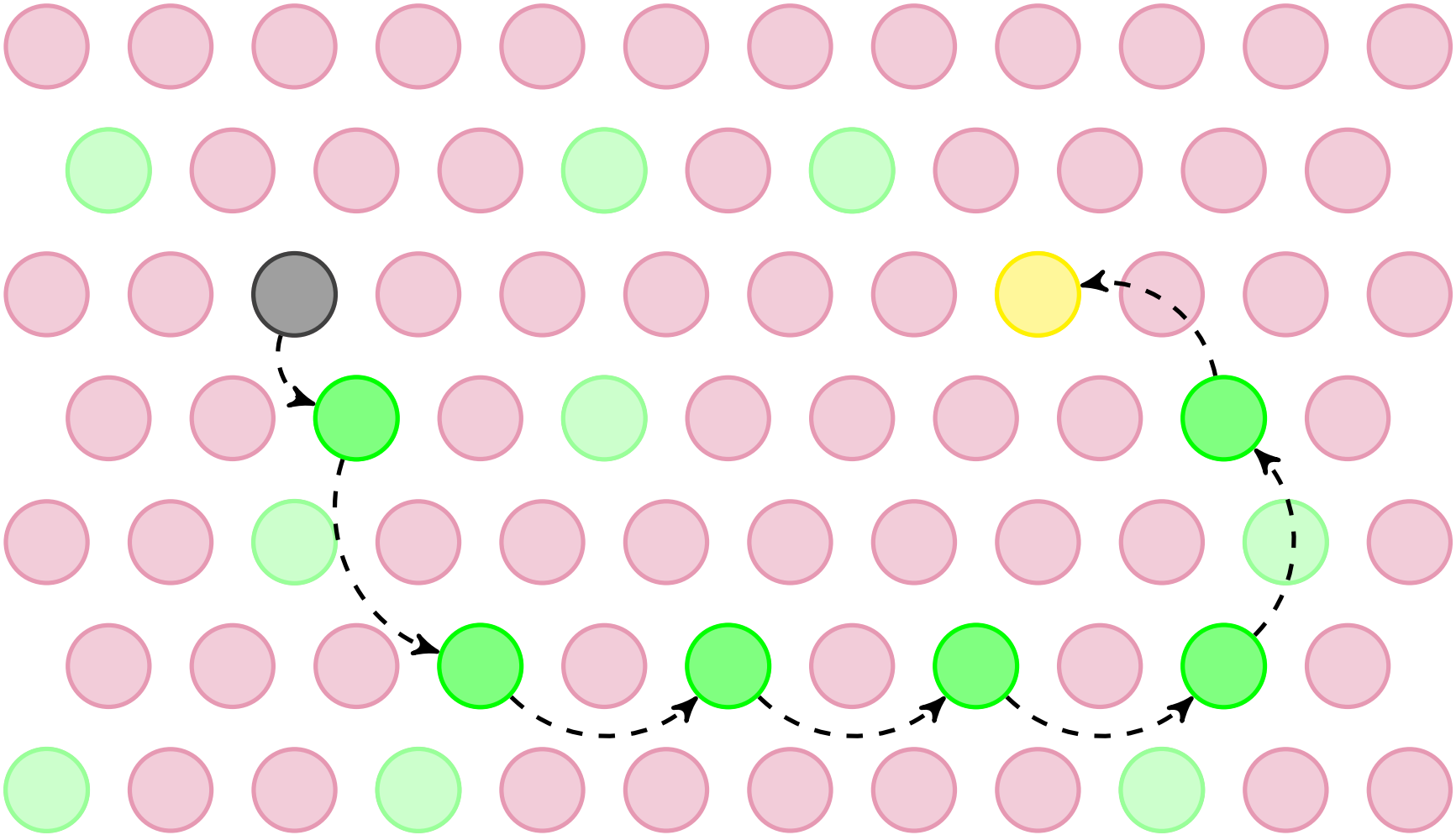
Feasible and infeasible

- Oscillation strategy (variation of α)
- Important: guarantee the transition between feasible and infeasible solution space

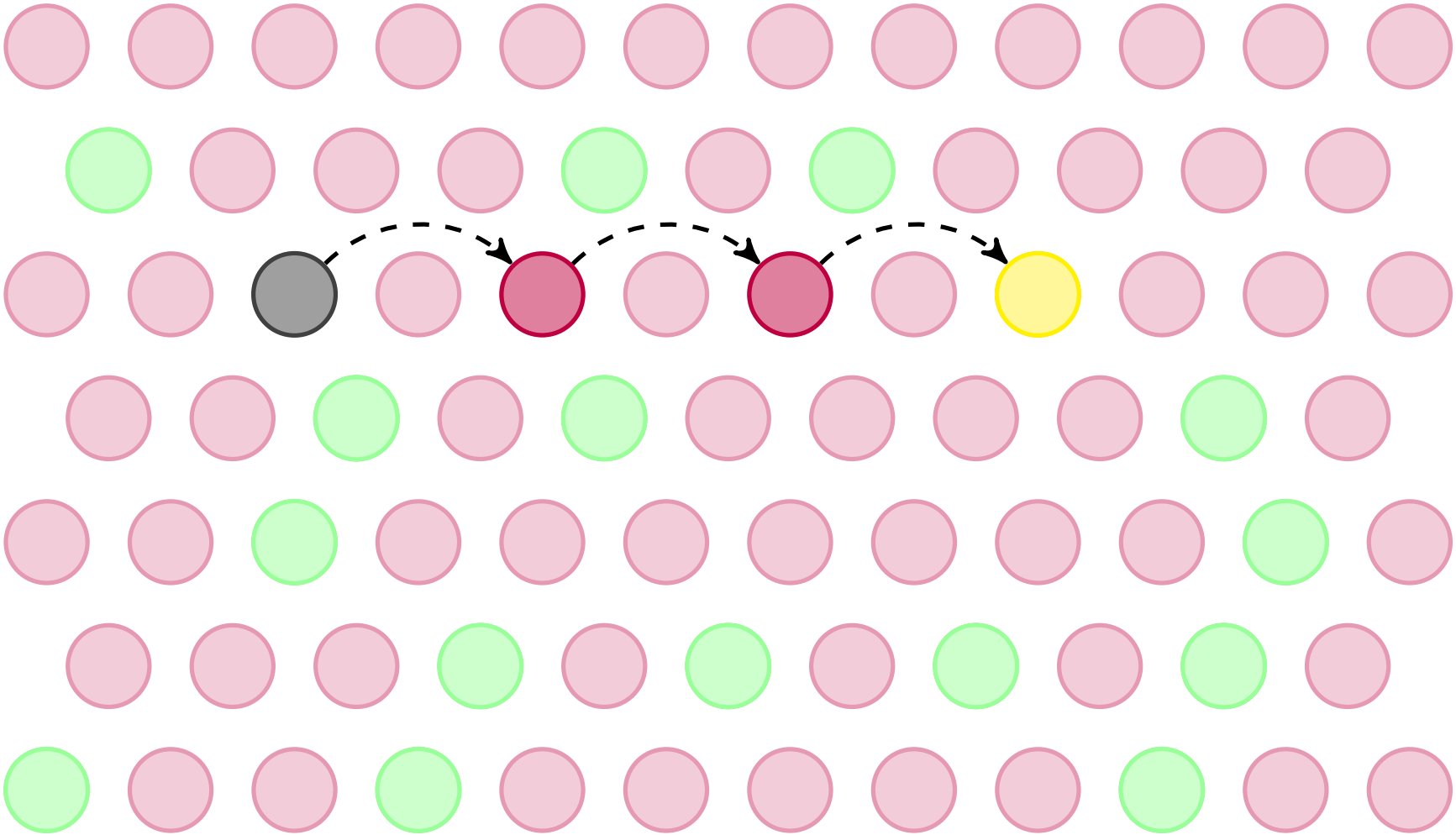
Consider infeasible solutions - Motivation



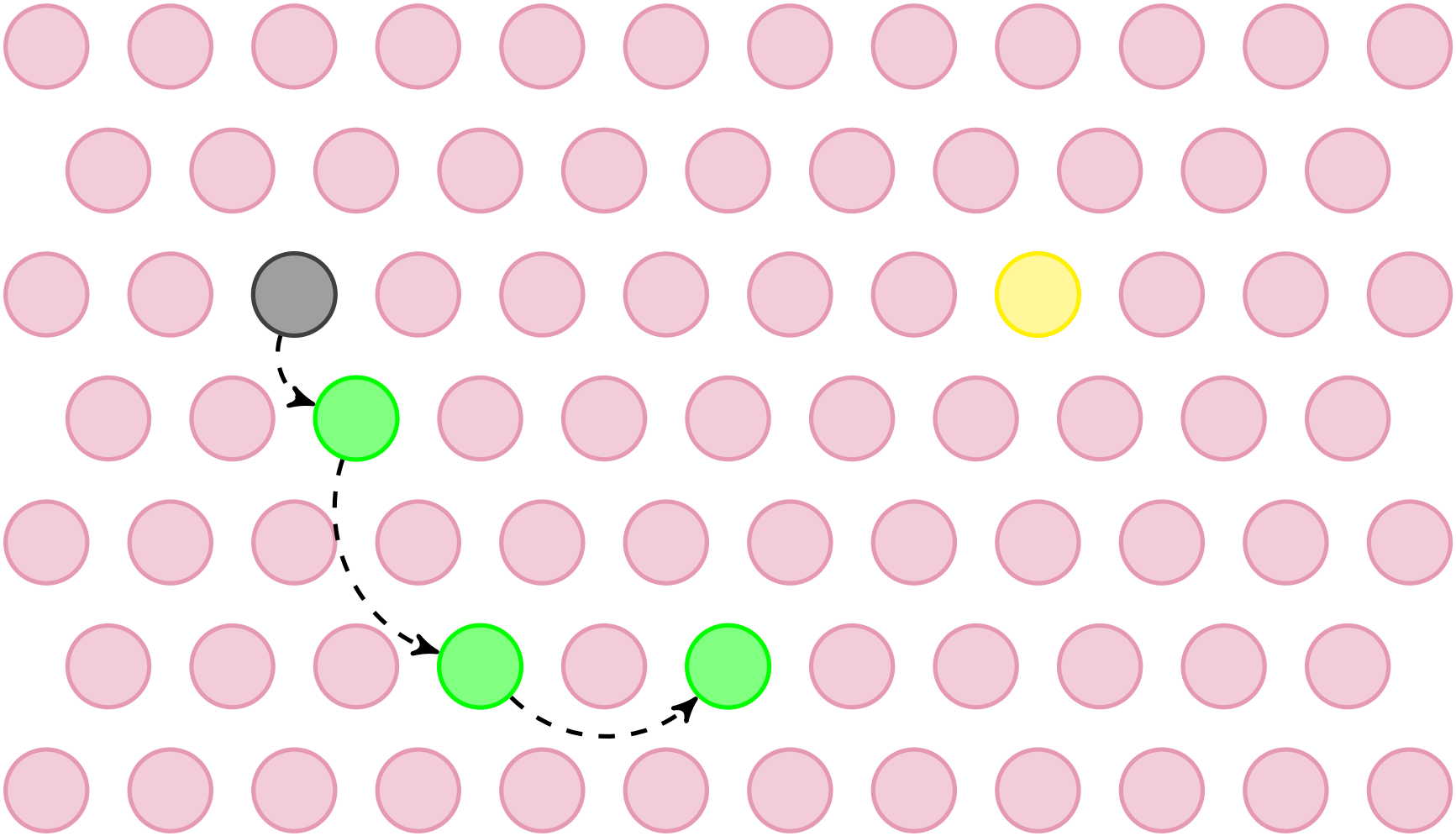
Consider infeasible solutions - Motivation



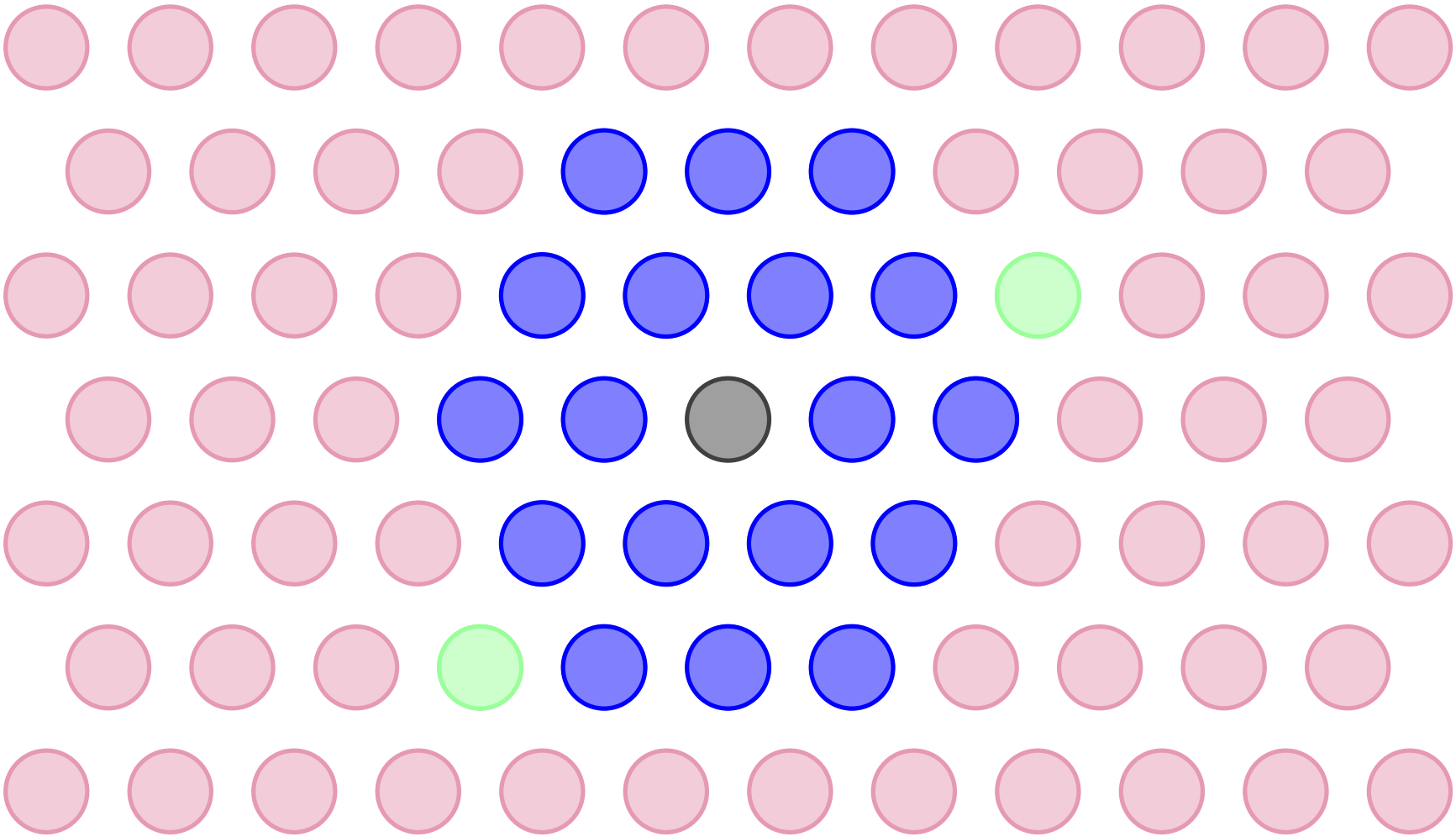
Consider infeasible solutions - Motivation



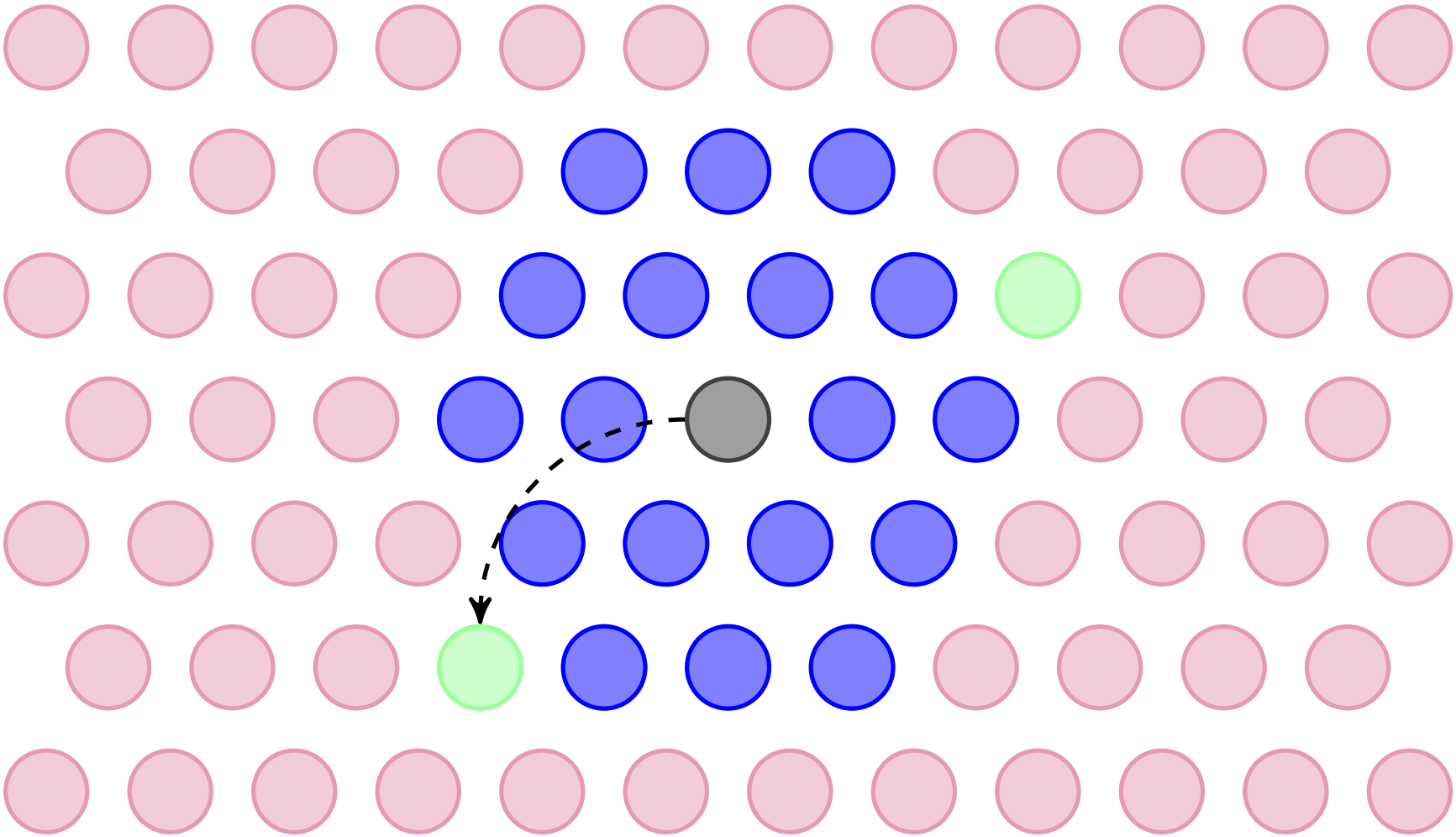
Consider infeasible solutions - Worst case



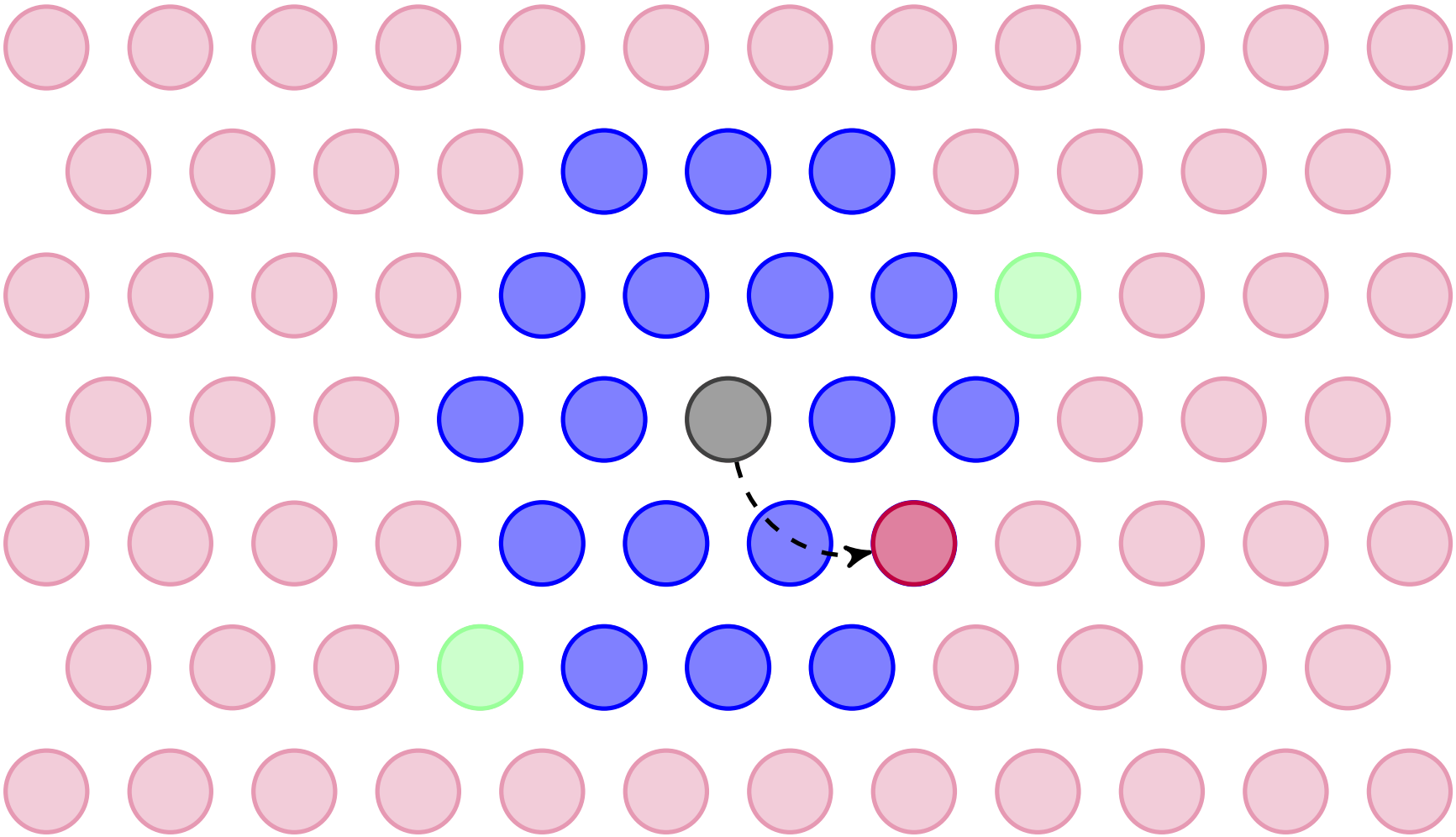
Consider infeasible solutions - Important case



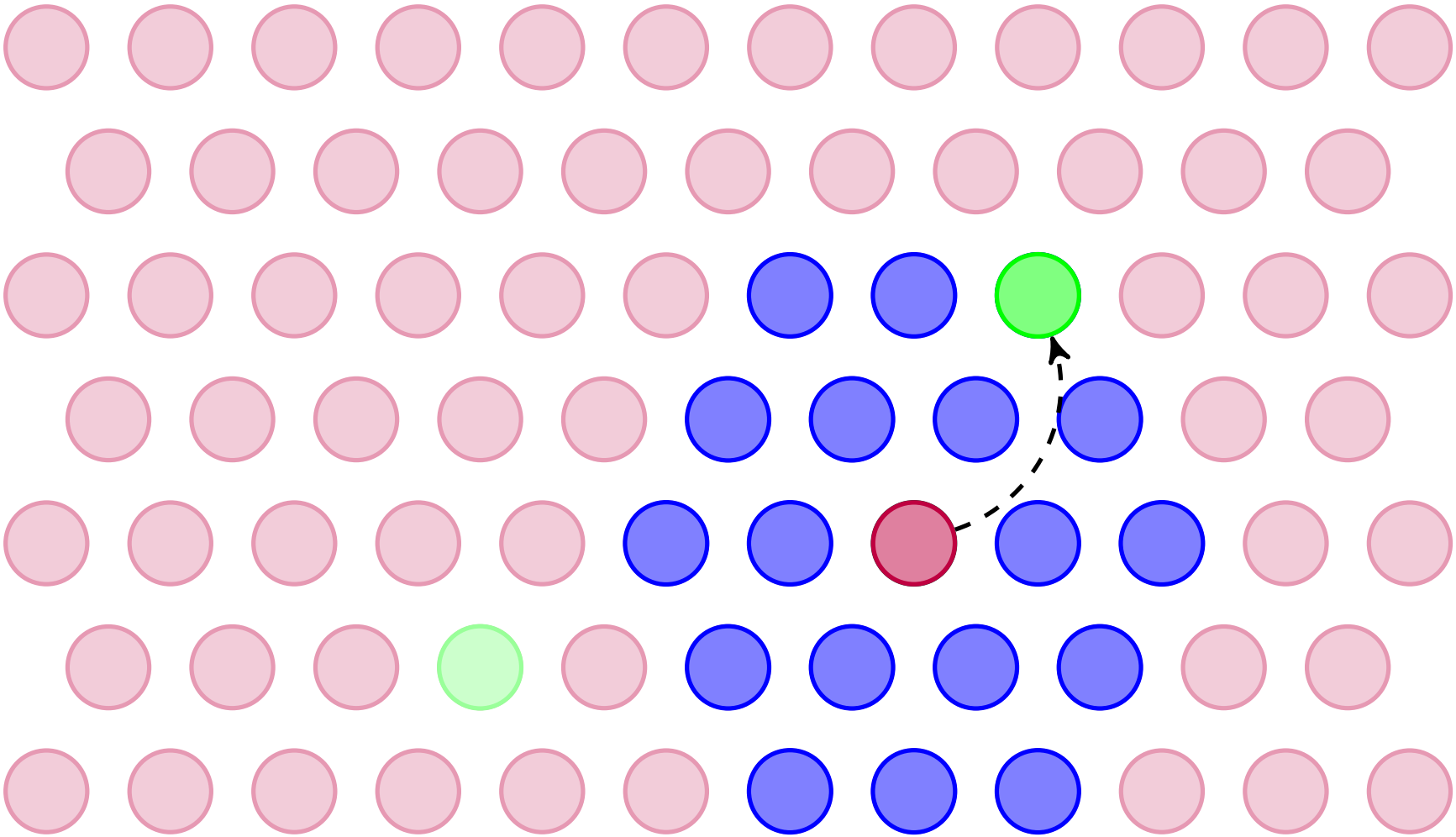
Consider infeasible solutions - Important case



Consider infeasible solutions - Important case



Consider infeasible solutions - Important case



Computational experiments

- Set of instances proposed by Augerat et al.
- Oscillation strategy:
  - Initial value of $\alpha = 1$
  - Increase α by 1 every time no better solution can be found
  - In case of getting stuck in the infeasible solution space, set value of $\alpha = 1$

Strategy	Set A	Set B	Set P
Only feasible sol.	10.13	-	-
Force feasibility	10.18	10.80	-
Oscillation	6.88	3.82	5.96

Table : Average solution gap

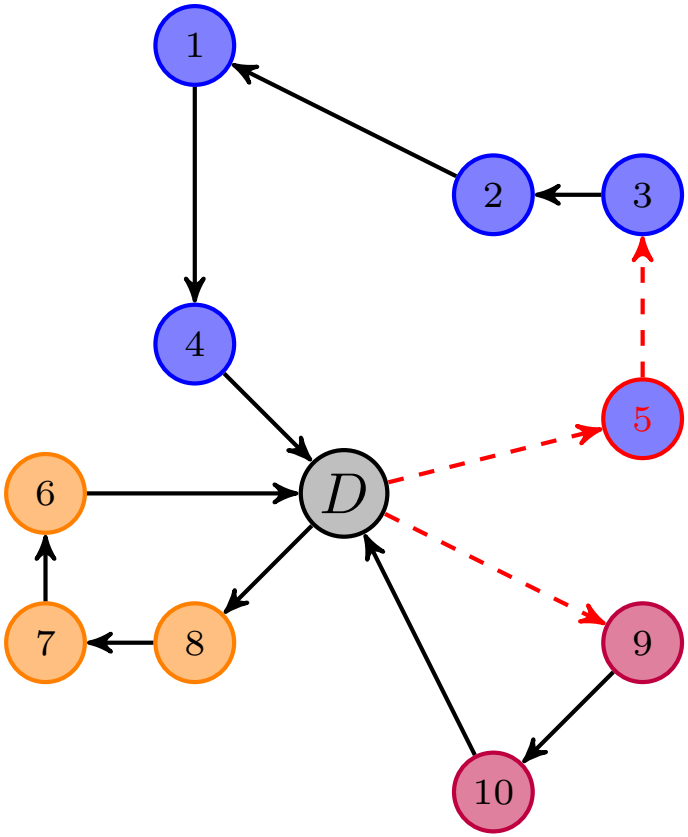
Implementation - Goal

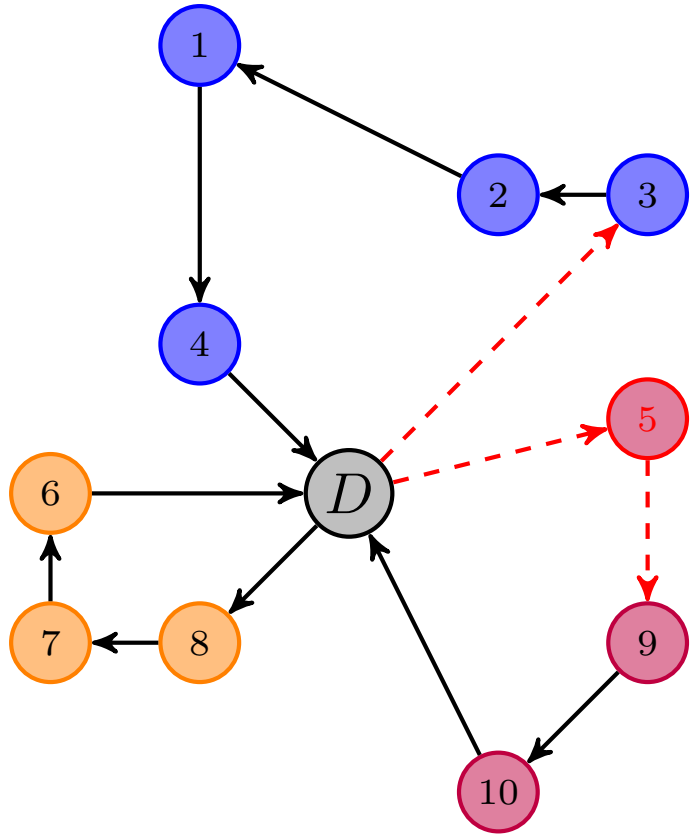
Be able to calculate/perform in the least order of time complexity:

- Differential cost of neighbouring solutions
- Update the neighbourhood set
- Apply the neighbourhood operators (neighbouring solution)

Differential cost of neighbouring solutions

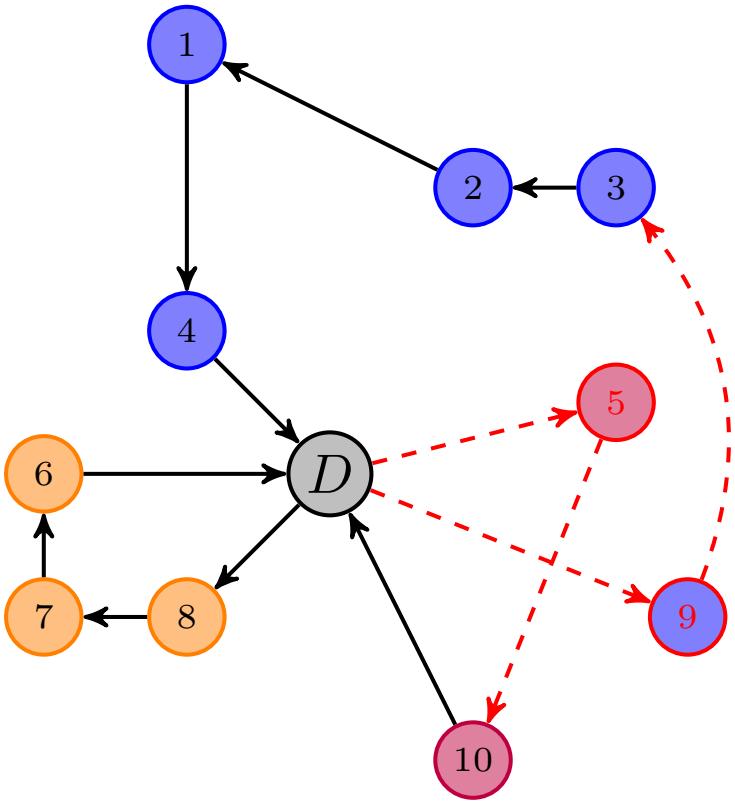
- Relocate (intra-route and inter-route)

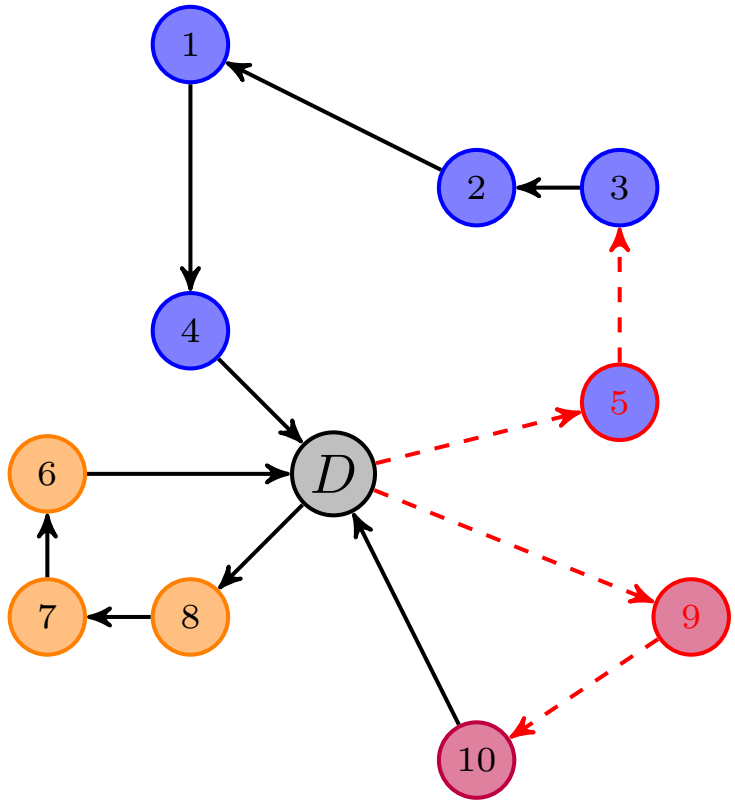




Differential cost of neighbouring solutions

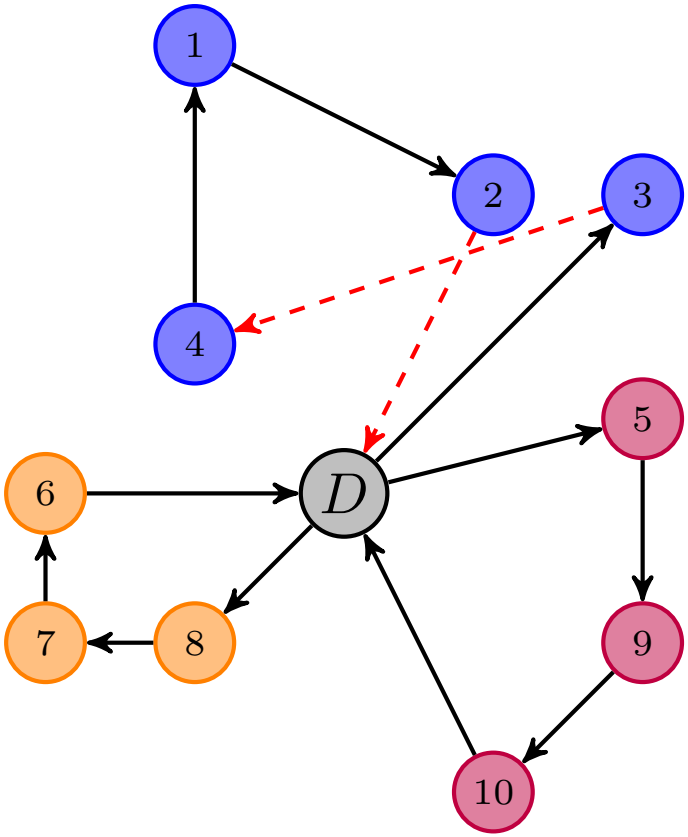
- Exchange (intra-route and inter-route)

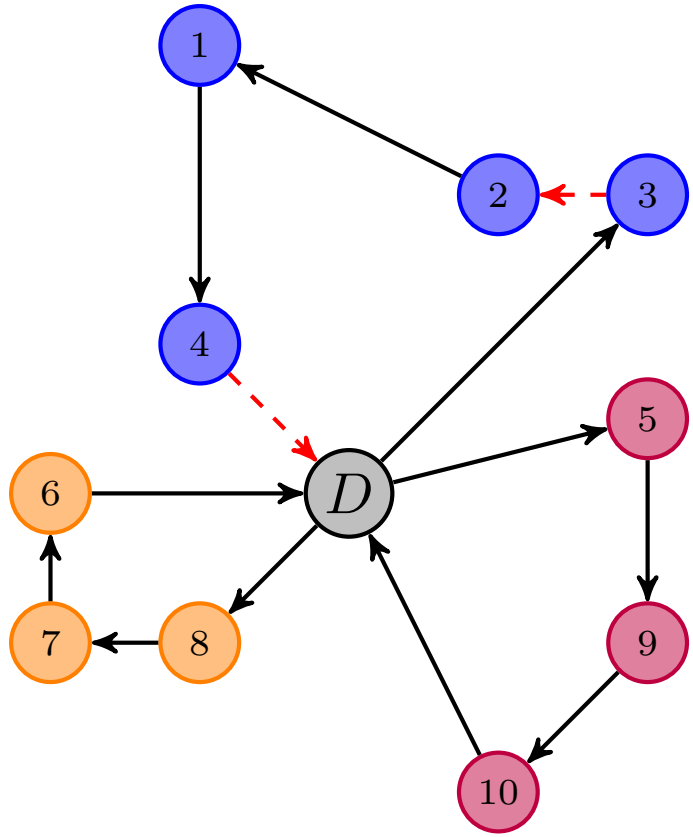




Differential cost of neighbouring solutions

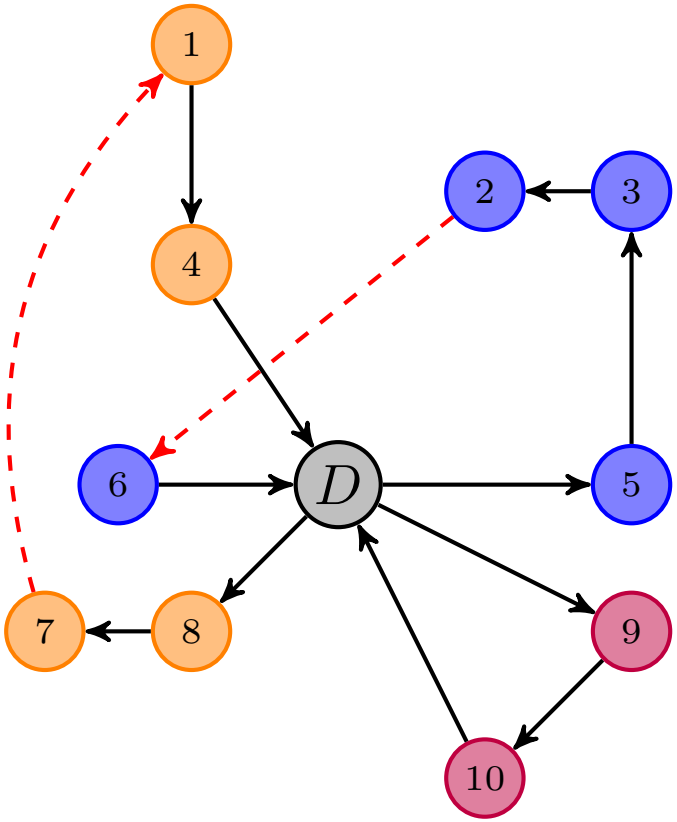
- Two-opt (intra-route)

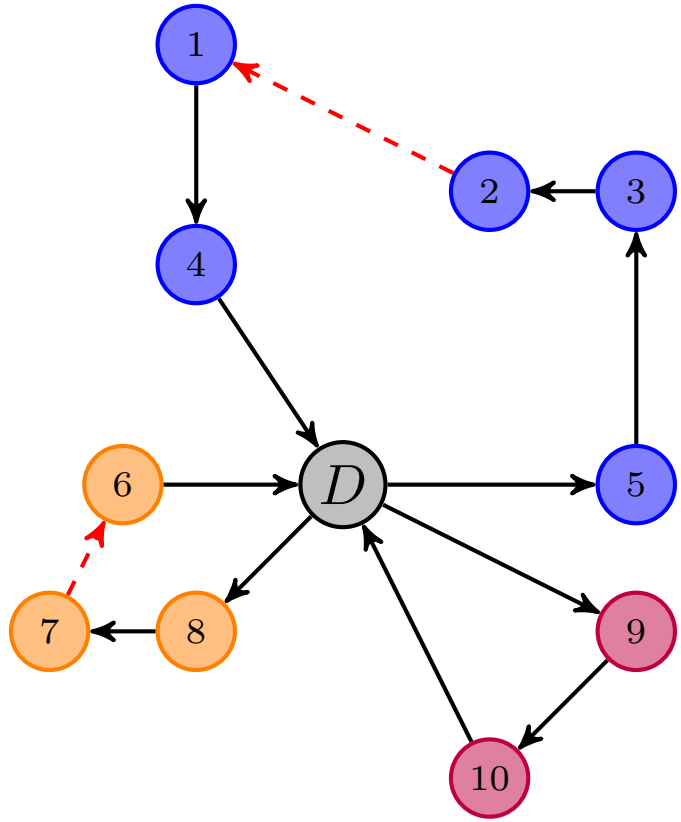




Differential cost of neighbouring solutions

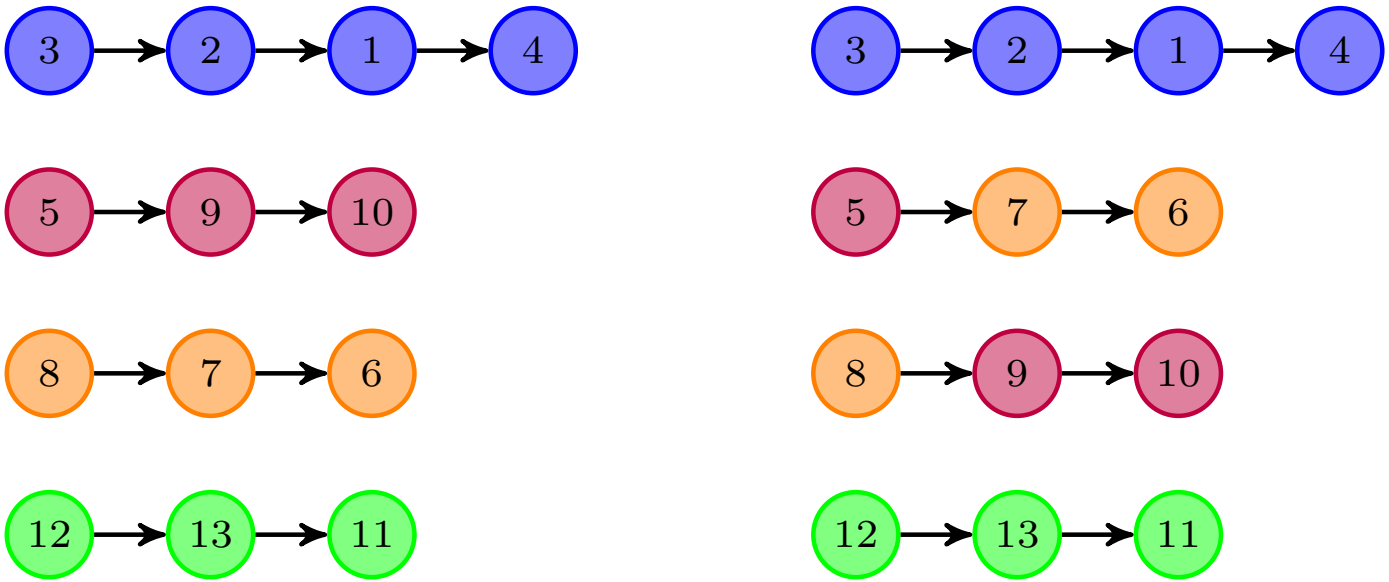
- Crossover (inter-route)



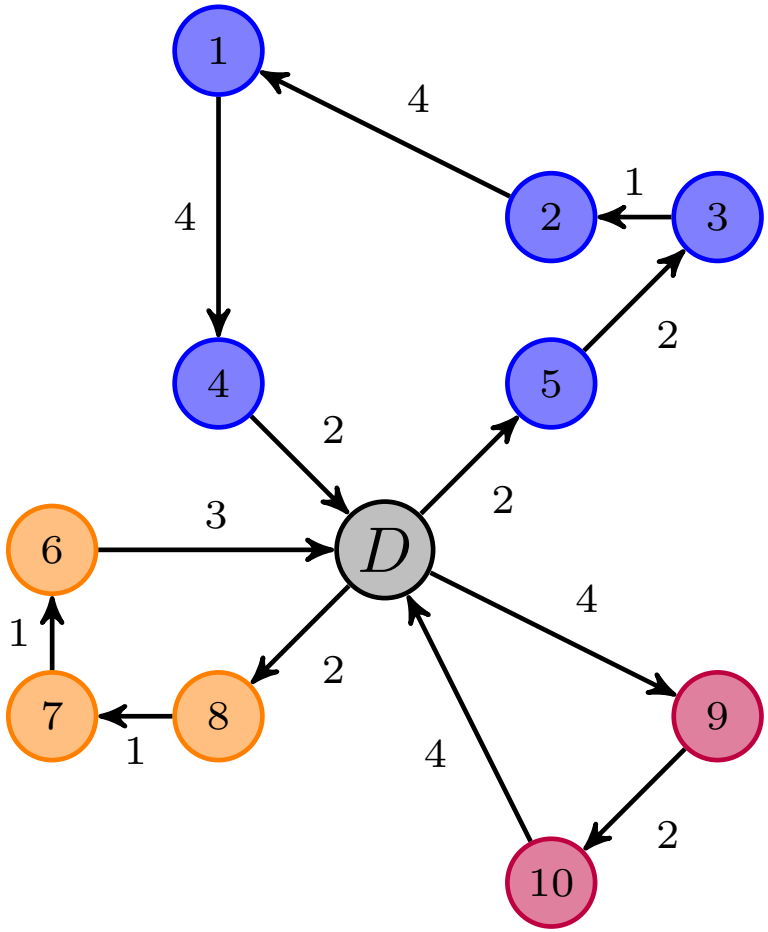


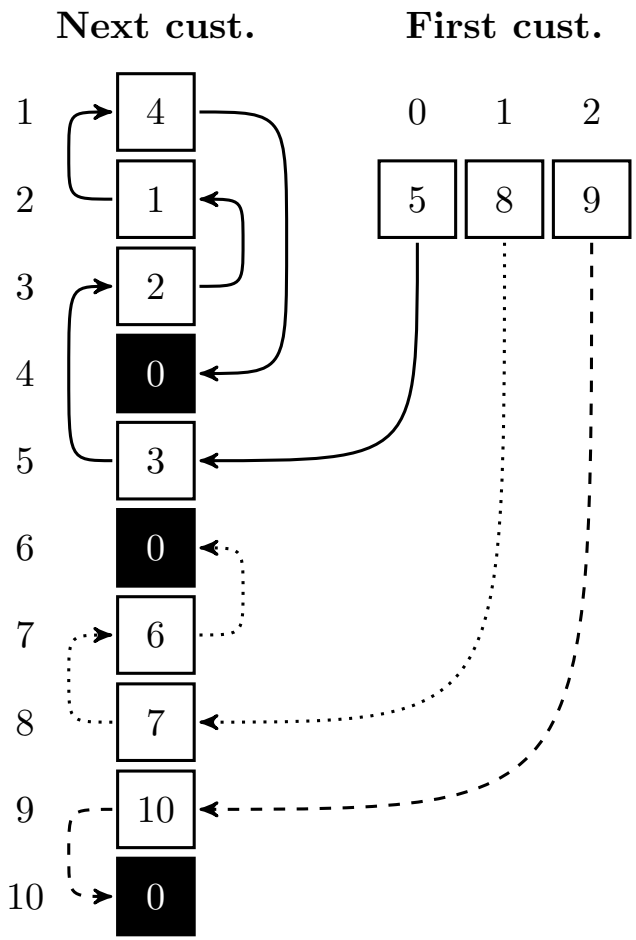
Update the set of neighbouring solutions

- Only recalculate the differential cost of the neighbouring solutions that involve the modified routes
  - Time complexity reduced from k^2 to $2(k-2)+2k$
- Reduction of average execution time by 50%

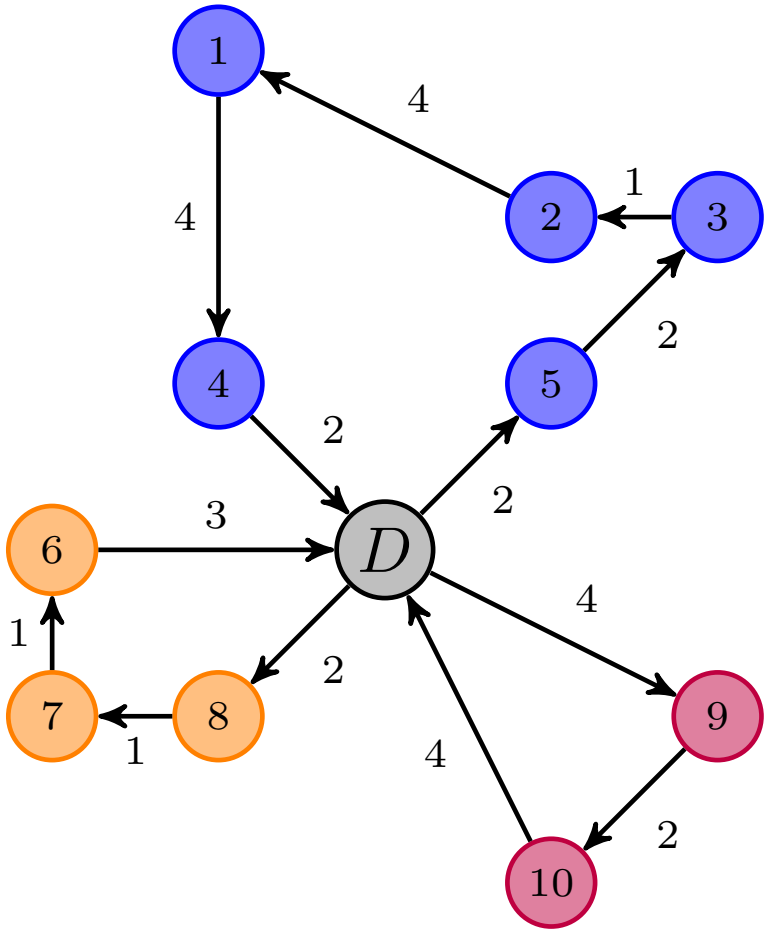


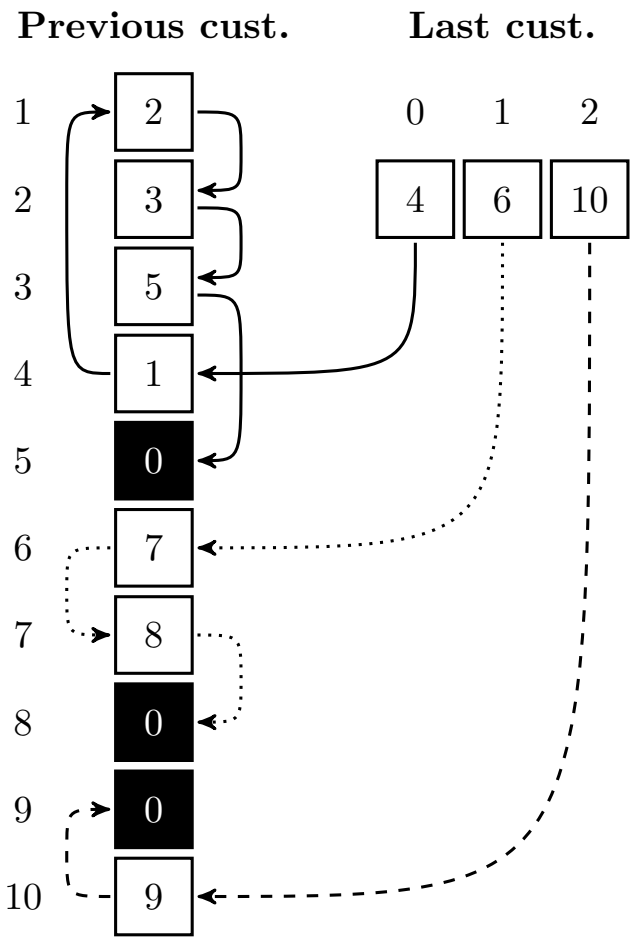
Data Structure





Data Structure





Final remarks

About considering infeasible solutions:

- Can be helpful to find valuable information about the global optimum solution in fewer iterations
- Useful when is difficult to generate a feasible initial solution

Last recommendation:

- Evaluate the trade-off between efficiency and usability of the code