Metaheuristics & network design

Talk given at the Network Design Workshop of the Ninth INFORMS Telecommunications Conference
University of Maryland, College Park, MD ~ March 29, 2008

Mauricio G. C. Resende
AT&T Labs Research
Florham Park, New Jersey
mgcr@att.com

Summary

- GRASP & path-relinking
  - GRASP
  - Path-relinking
  - GRASP with path-relinking
  - GRASP with path-relinking for the prize-collecting Steiner problem in graphs
- Genetic algorithms
  - Genetic algorithm (GA)
  - GA with random-keys
  - Weight setting for OSPF routing
  - Survivable network design with OSPF routing

Combinatorial Optimization

Combinatorial optimization: process of finding the best, or optimal, solution for problems with a discrete set of feasible solutions.

Network design: is an important application of combinatorial optimization.

Combinatorial Optimization

- Given:
  - discrete set of solutions X
  - objective function $f(x)$: $x \in X \rightarrow R$
- Objective:
  - find $x \in X : f(x) \leq f(y), \forall y \in X$

Heuristics for Combinatorial Optimization

Aim of heuristic methods for combinatorial optimization is to quickly produce good-quality solutions, without necessarily providing any guarantee of solution quality.

Metaheuristics

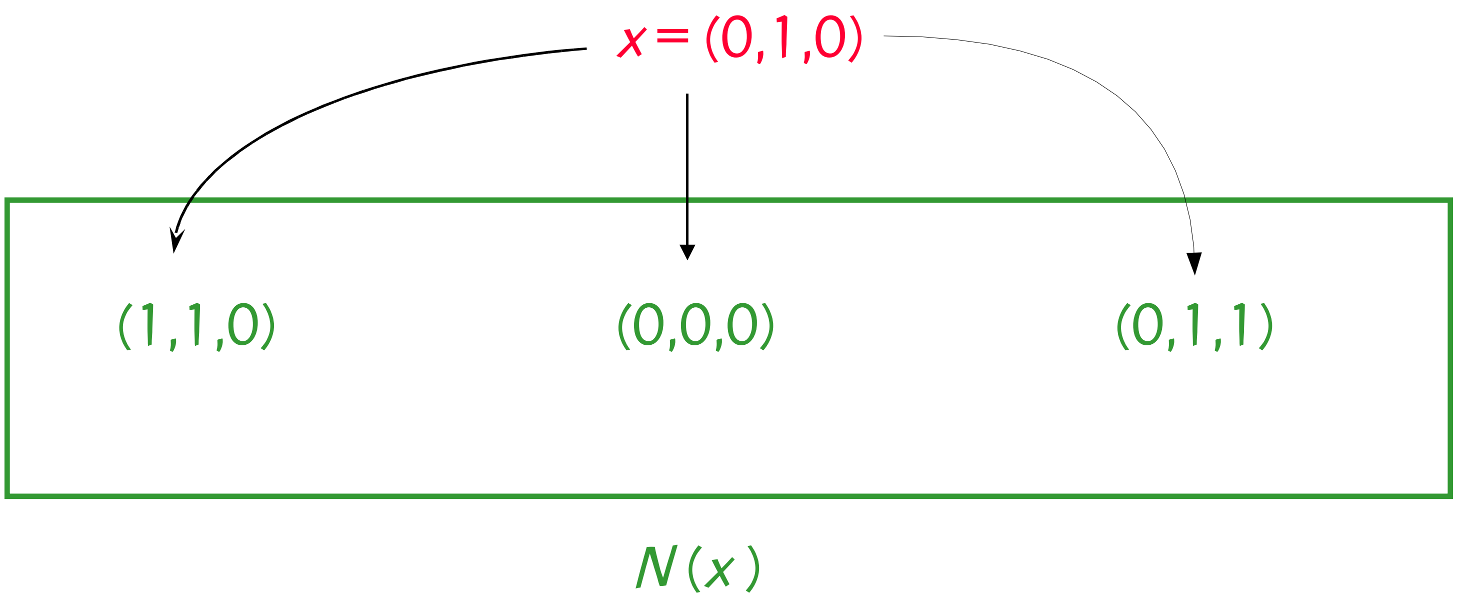
- Metaheuristics are high level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone.
- Examples: simulated annealing, tabu search, scatter search, ant colony optimization, variable neighborhood search, pilot method, GRASP, and genetic algorithms.

Local Search

- To define local search, one needs to specify a local neighborhood structure.
- Given a solution x , the elements of the neighborhood N(x) of x are those solutions y that can be obtained by applying an elementary modification (often called a move) to x.

Local Search Neighborhoods

Consider $x = (0,1,0)$ and the 1-flip neighborhood of a 0/1 array.



Local Search

Given an initial solution x_0, a neighborhood $N(x)$, and function $f(x)$ to be minimized:

$x = x_0$;

check for better solution in neighborhood of *x*

while ($\exists\, y \in N(x) \mid f(y) < f(x)$) {

$x = y$; move to better solution *y*

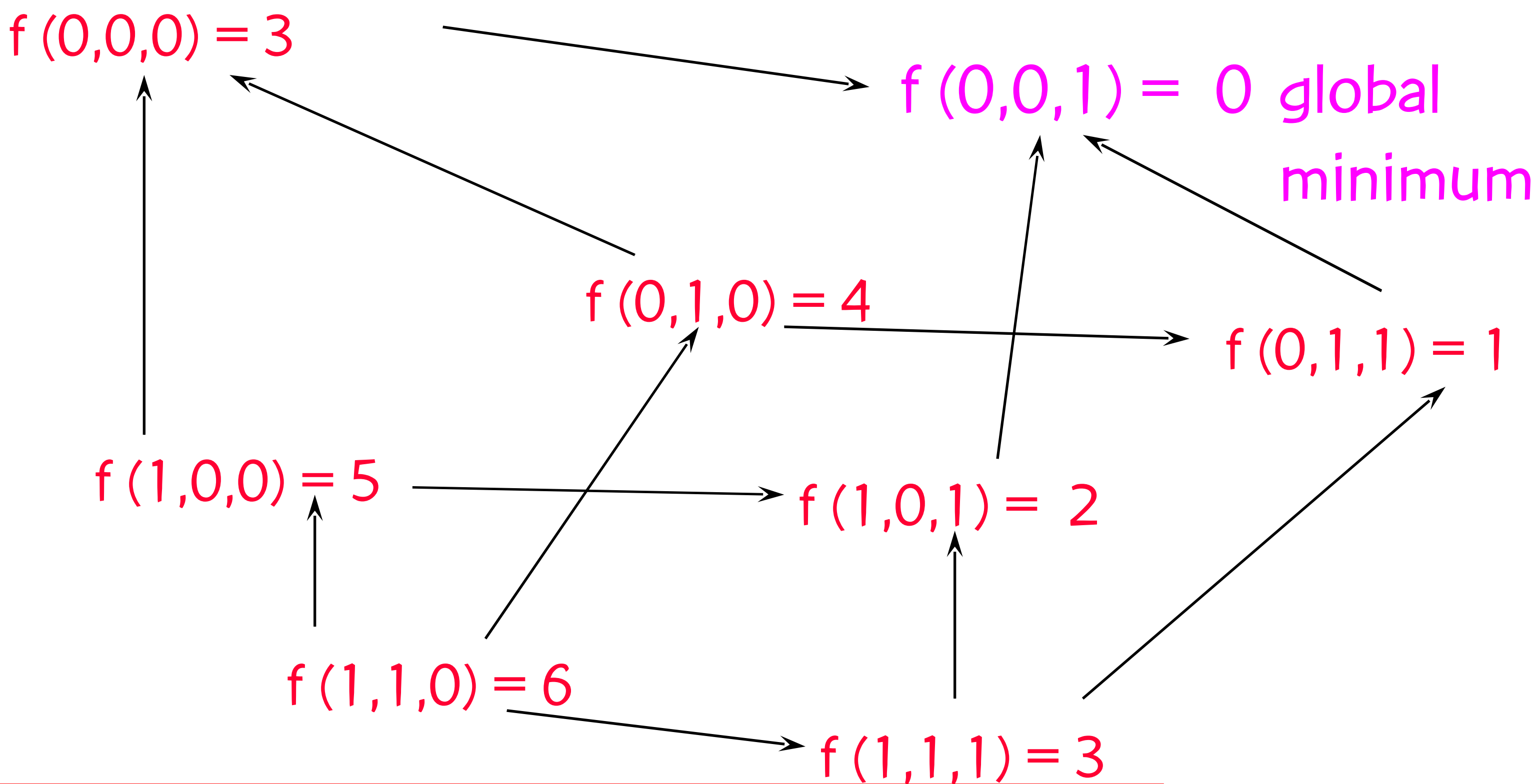
}

Time complexity of local search can be exponential.

At the end, x is a local minimum of $f(x)$.

Local Search

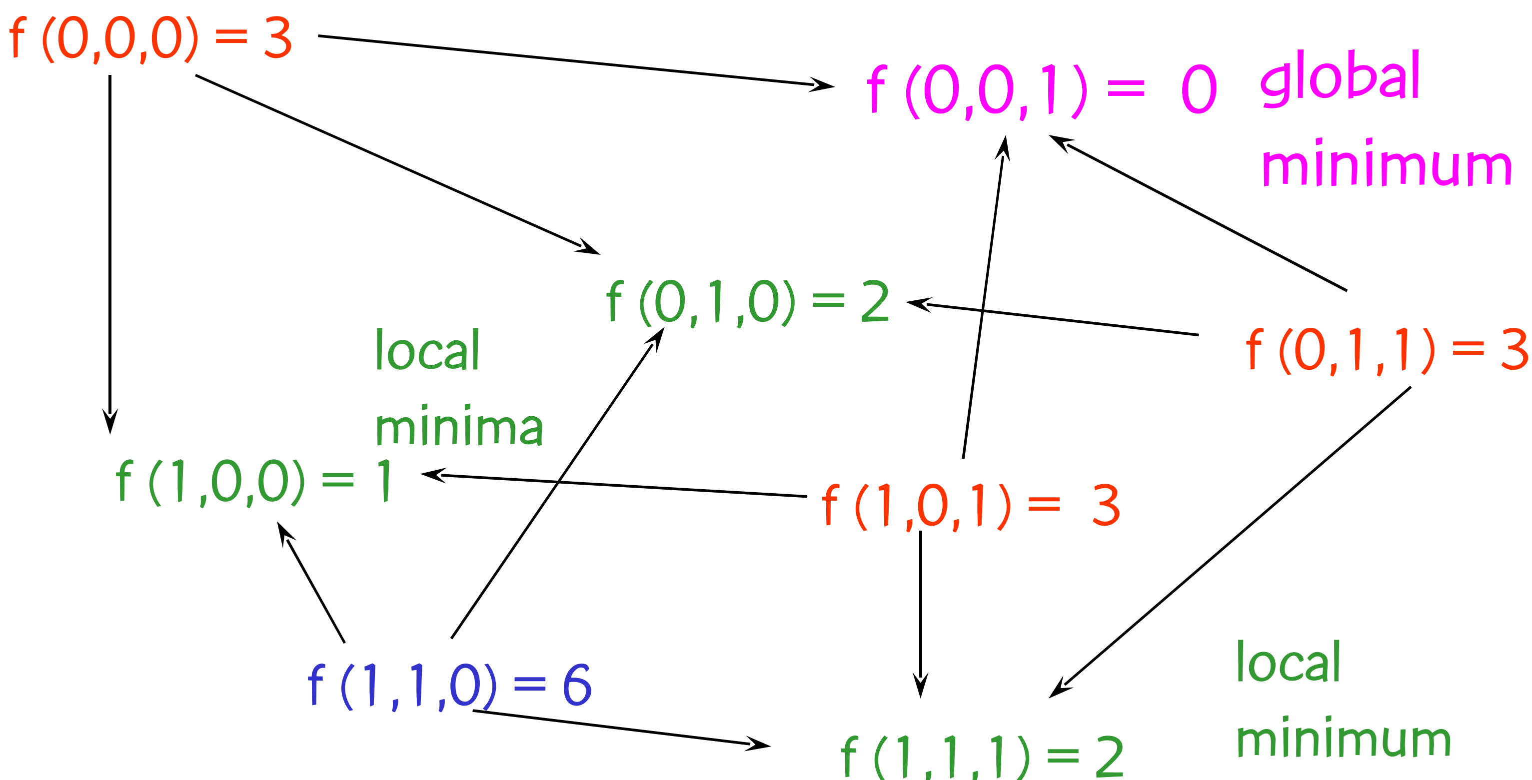
(ideal situation)



With any starting solution Local Search finds the global optimum.

Local Search

(more realistic situation)



But some starting solutions lead Local Search to a local minimum.

Local Search

Effectiveness of local search depends on several factors:

- neighborhood structure

some freedom to choose

Local Search

Effectiveness of local search depends on several factors:

- neighborhood structure
- function to be minimized

some freedom to choose

usually pre-determined

Local Search

Effectiveness of local search depends on several factors:

– neighborhood structure

– function to be minimized

– starting solution

some freedom to choose

usually pre-determined

usually easier to control

The greedy algorithm

- Constructs a solution, one element at a time:

repeat until done

  - Defines candidate elements.
  - Applies a greedy function to each candidate element.
  - Ranks elements according to greedy function value.
  - Add best ranked element to solution

The greedy algorithm

- Constructs a solution, one element at a time:

repeat until done

  - Defines candidate elements.
  - Applies a greedy function to each candidate element.
  - Ranks elements according to greedy function value.
  - Add best ranked element to solution.

Greedy solutions are not necessarily locally optimal.

The greedy algorithm

- Constructs a solution, one element at a time:
  - Defines candidate elements.
  - Applies a greedy function to each candidate element.
  - Ranks elements according to greedy function value.
  - Add best ranked element to solution.

repeat until done

Greedy solutions are not necessarily locally optimal.

Applying local search to greedy solutions usually leads to a local optimum that is not globally optimum.

Multi-start greedy method

$c^* = \infty$

repeat

$x = greedy()$

if $f(x) < c^*$ then

$x^* = x$

$c^* = f(x^*)$

endif

Multi-start greedy method

$c^* = \infty$

repeat

$x = \text{greedy}()$

if $f(x) < c^*$ then

  $x^* = x$

  $c^* = f(x^*)$

endif

multi-start with greedy does poorly because greedy lacks randomness

Random multi-start

$c^* = \infty$

repeat

$x = \text{random\_construction()}$

if $f(x) < c^*$ then

$x^* = x$

$c^* = f(x^*)$

endif

Example: Probability of finding opt with K samplings on a 0–1 vector of size N

K \ N:	10	15	20	25	30
10	.010	.000	.000	.000	.000
100	.093	.003	.000	.000	.000
1000	.624	.030	.000	.000	.000
10000	1.000	.263	.009	.000	.000
100000	1.000	.953	.091	.003	.000

Semi-greedy heuristic

Hart and Shogan (1987)

- A semi-greedy heuristic adds randomization to the greedy algorithm.
- repeat until solution is constructed

repeat until done

  - For each candidate element
    - apply a greedy function to element
  - Rank all elements according to their greedy function values
  - Place well-ranked elements in a restricted candidate list (RCL)
  - Select an element from the RCL at random & add it to the solution

Hart-Shogan Algorithm

$c^* = \infty$

repeat

$x = \text{semi\_greedy\_construction}()$

if $f(x) < c^*$ then

$x^* = x$

$c^* = f(x^*)$

endif

Hart-Shogan Algorithm

$c^* = \infty$

repeat

x = semi_greedy_construction()

if $f(x) < c^*$ then

$x^* = x$

$c^* = f(x^*)$

endif

semi-greedy solutions are not necessarily locally optimum

GRASP

Greedy Randomized Adaptive Search Procedure

GRASP

Feo & Resende (1989, 1995); Resende & Ribeiro (2003)

$c^* = \infty$

repeat

$x = \text{semi\_greedy\_construction}()$

$x = \text{local\_search}(x)$

if $f(x) < c^*$ then

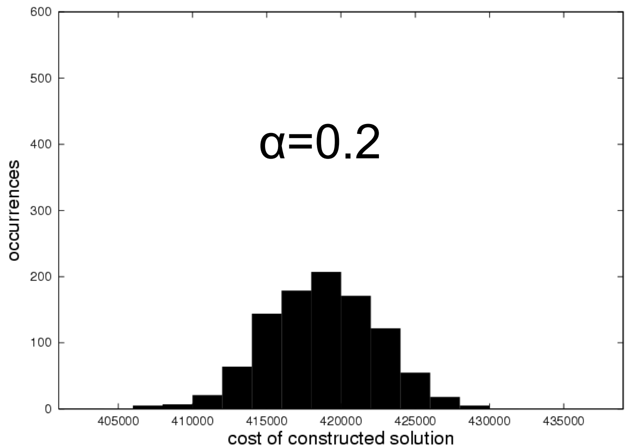
$\quad x^* = x$

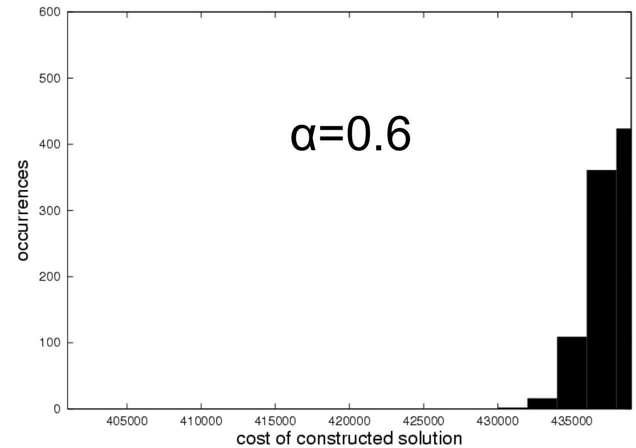
$\quad c^* = f(x^*)$

endif

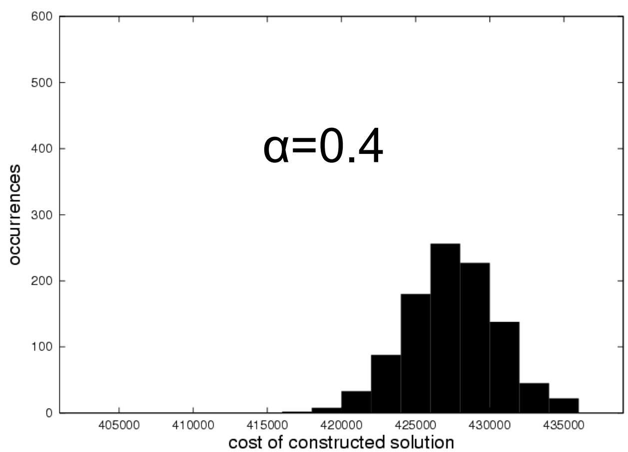
Semi-greediness is more general in GRASP

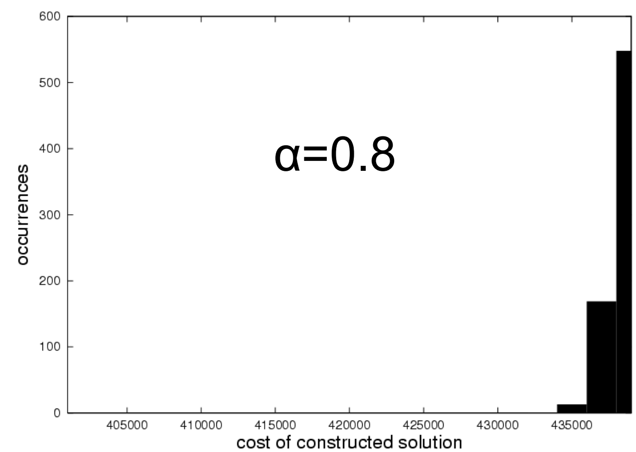
Illustrative results: RCL parameter





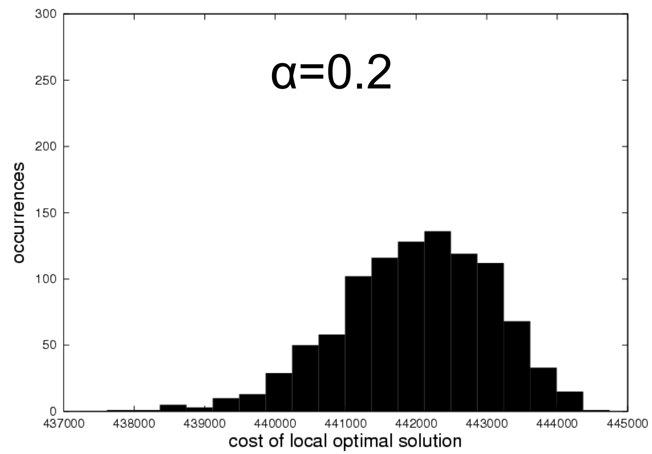
Semi-greedy algorithm

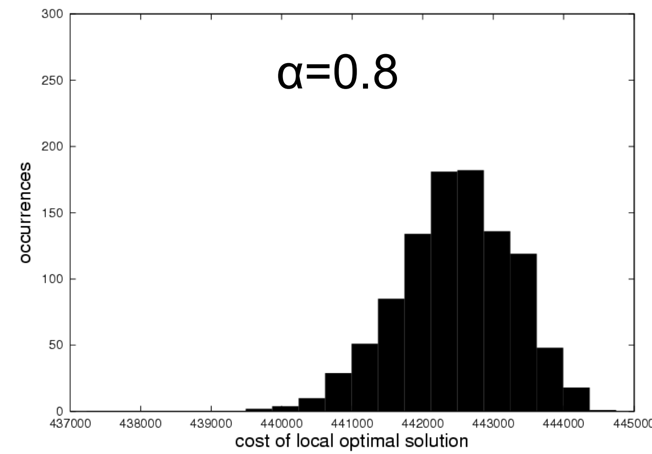




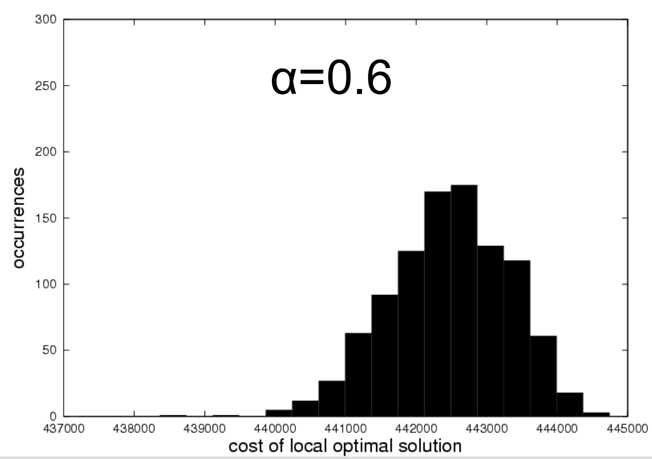
weighted MAX-SAT instance, 1000 iterations

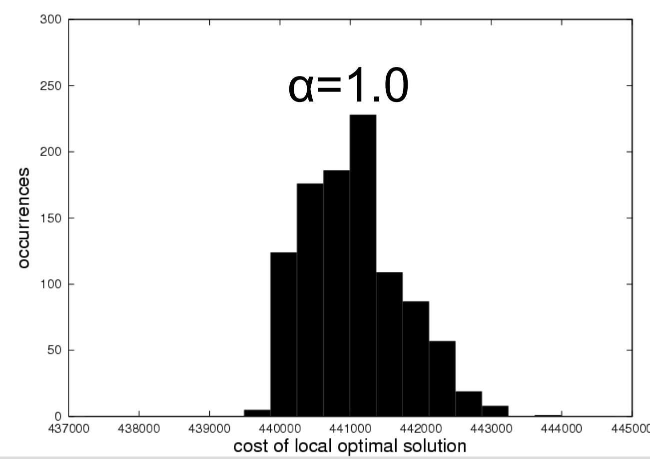
Illustrative results: RCL parameter





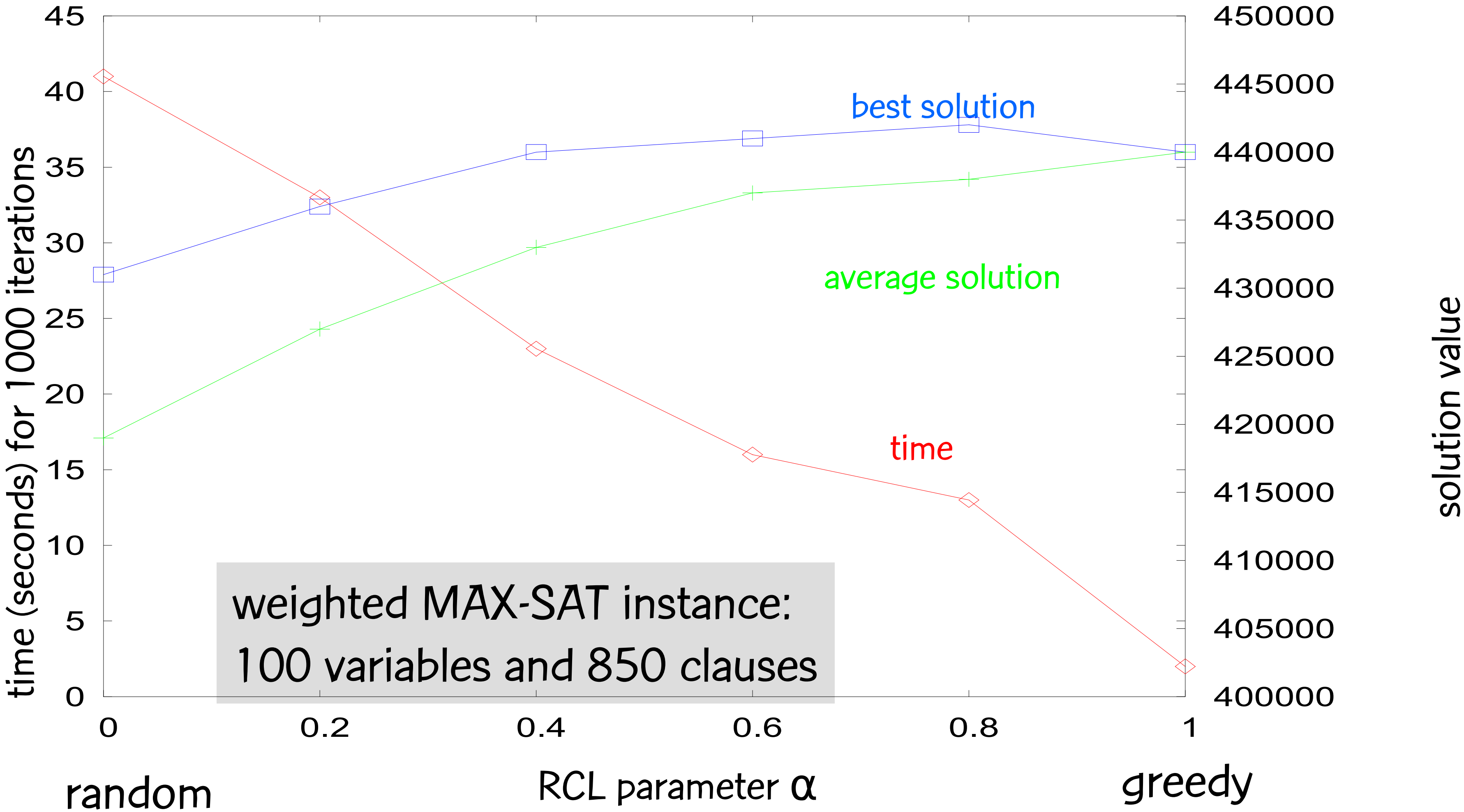
GRASP: Semi-greedy + local search





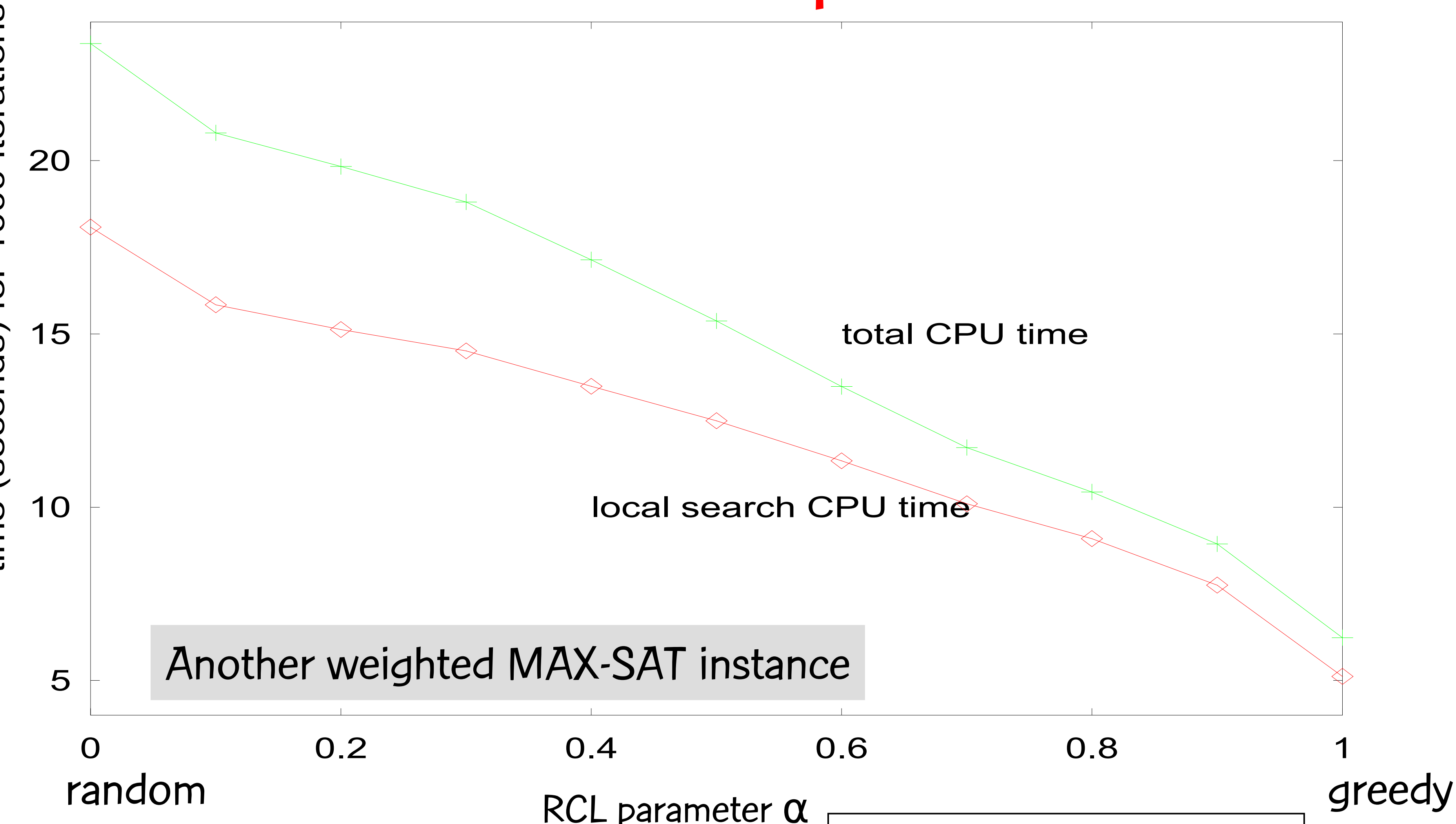
weighted MAX-SAT instance, 1000 GRASP iterations

Illustrative results: RCL parameter



SGI Challenge 196 MHz

Illustrative results: RCL parameter

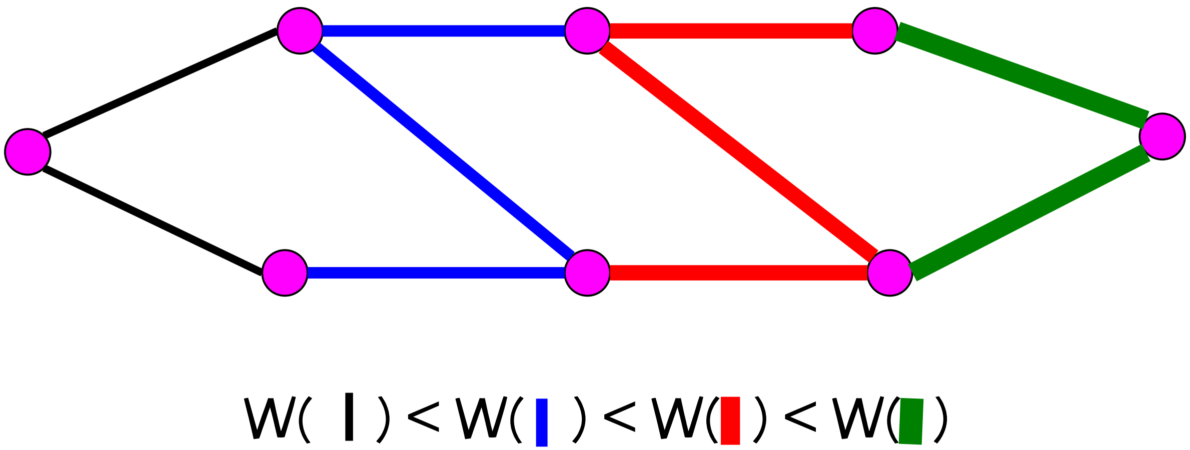


SGI Challenge 196 MHz

Construction with cost perturbation

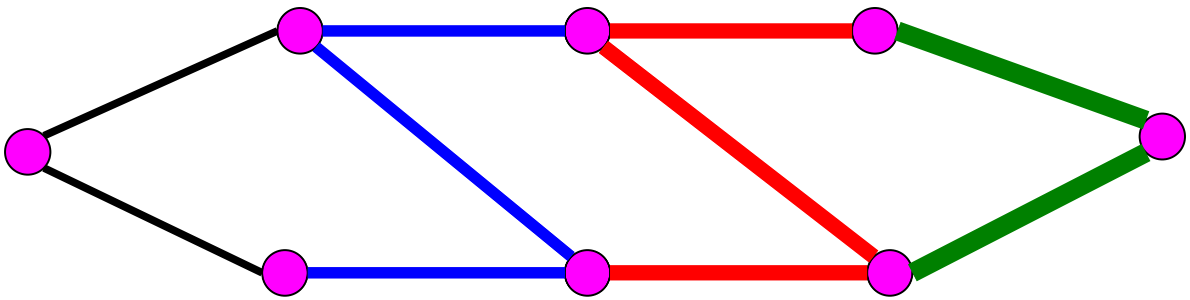
- Introduces noise into original costs: similar to Noisy Method of Charon and Hudry (1993, 2002)
- Randomly perturb original costs and apply some heuristic.
- Adds flexibility to algorithm design:
    - May be more effective than greedy randomized construction in circumstances where the construction algorithm is not very sensitive to randomization (Ribeiro, Uchoa, & Werneck, 2002).
    - Also useful when no greedy algorithm is available (Canuto, R., & Ribeiro, 2001).

Construction with cost perturbation

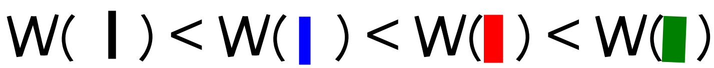


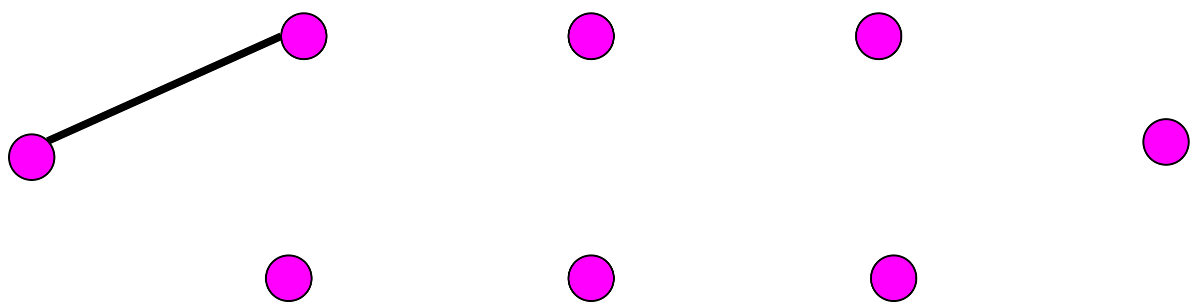
Perturb with costs increasing from top to bottom.

Construction with cost perturbation

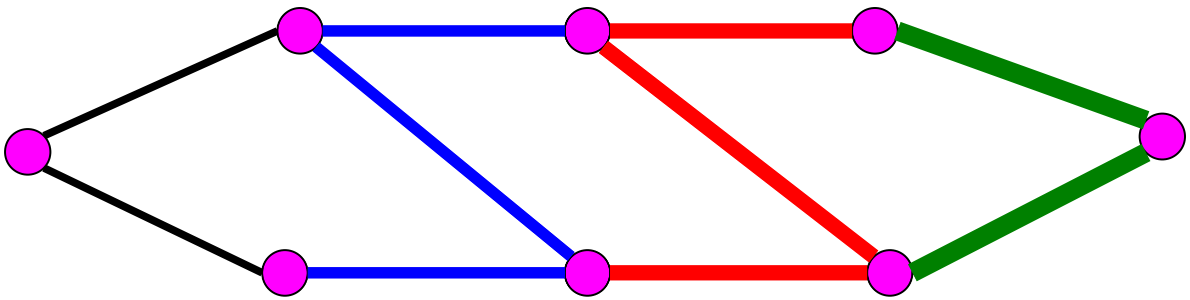


Perturb with costs increasing from top to bottom.



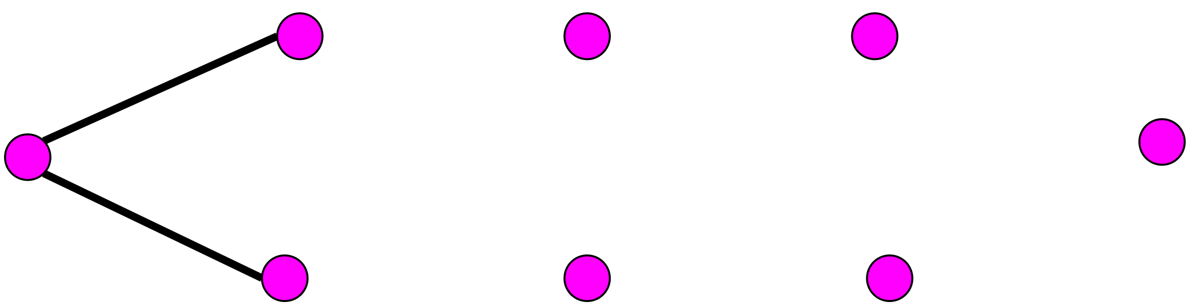


Construction with cost perturbation

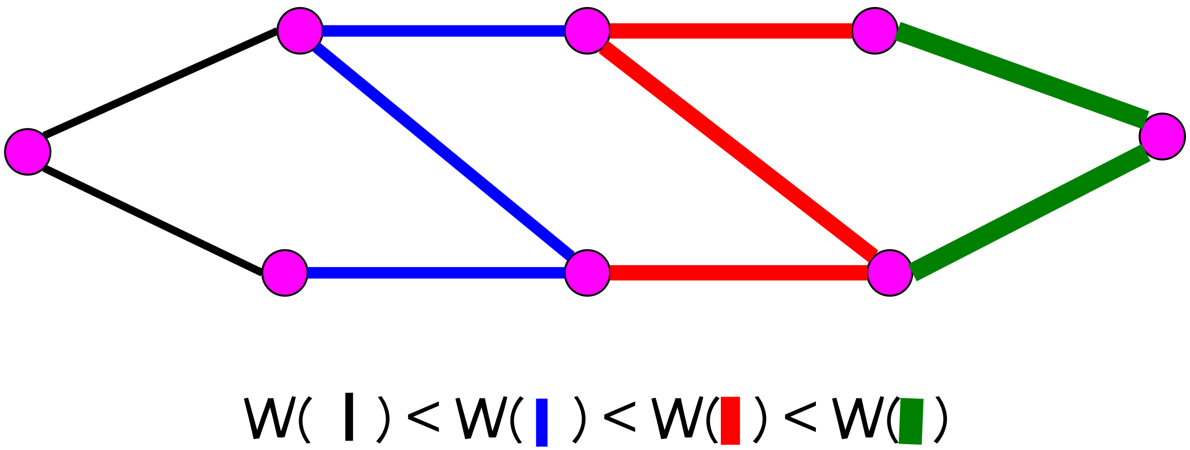


Perturb with costs increasing from top to bottom.

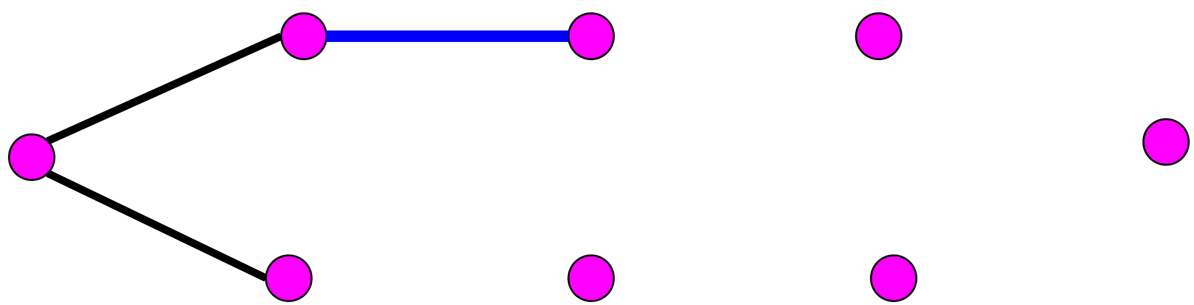
W( ) < W( ) < W( ) < W( )



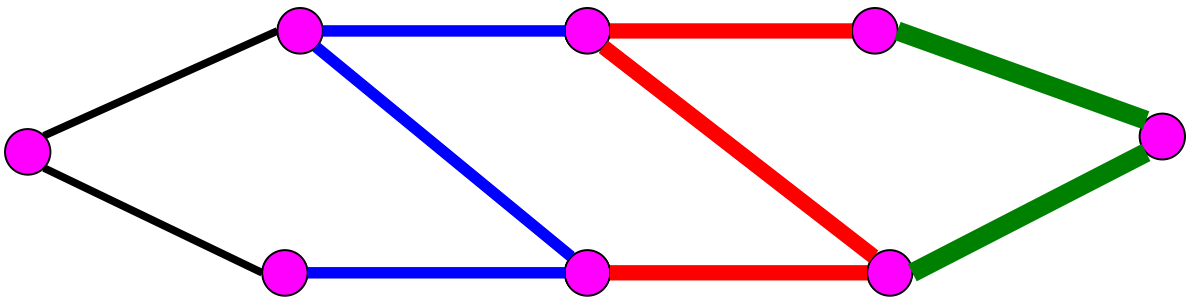
Construction with cost perturbation



Perturb with costs increasing from top to bottom.

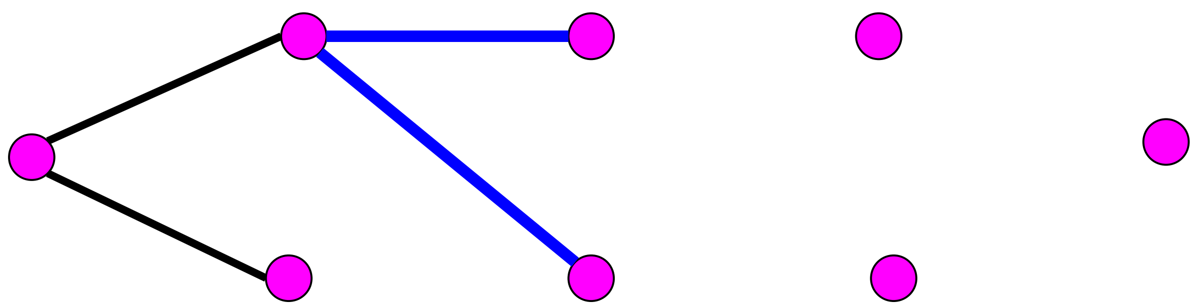


Construction with cost perturbation

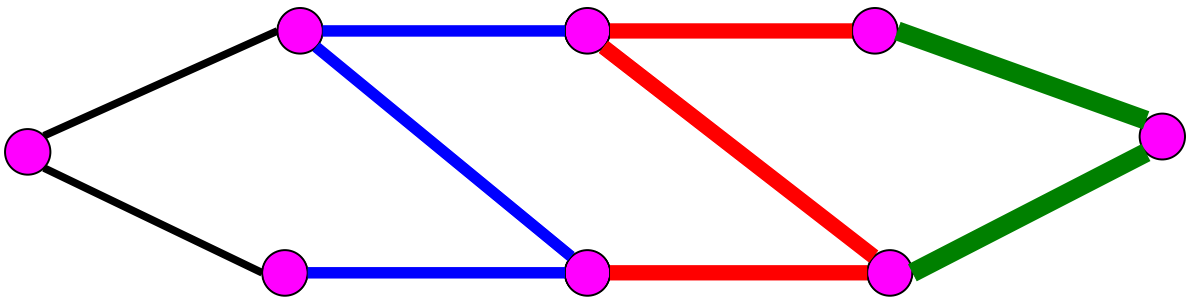


Perturb with costs increasing from top to bottom.

W( ) < W( ) < W( ) < W( )

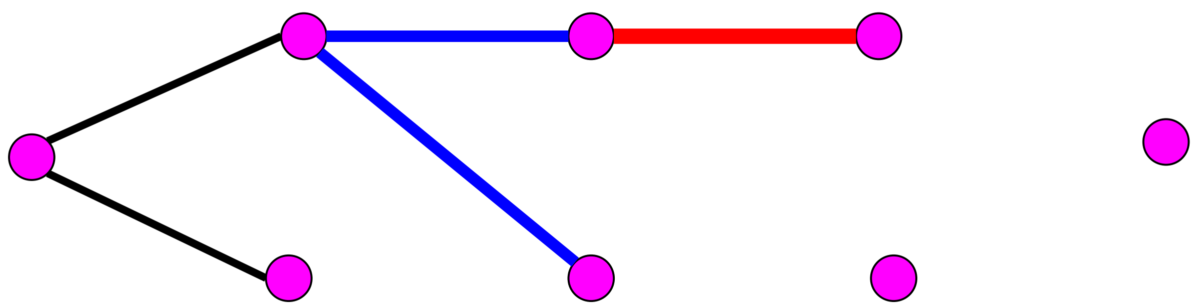


Construction with cost perturbation

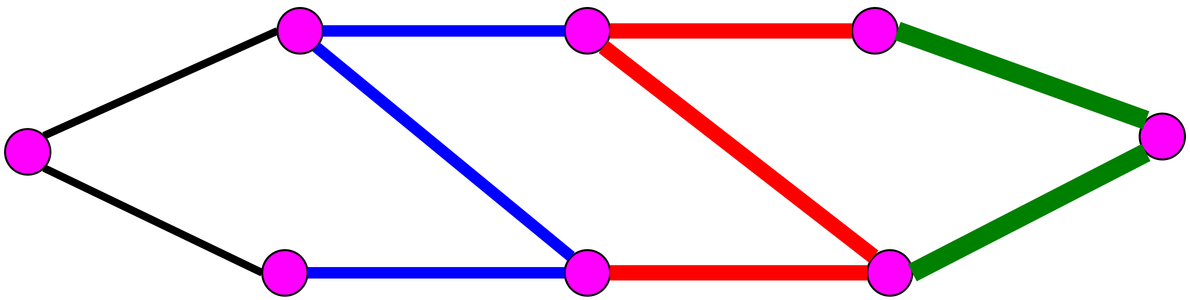


Perturb with costs increasing from top to bottom.

W( ) < W( ) < W( ) < W( )

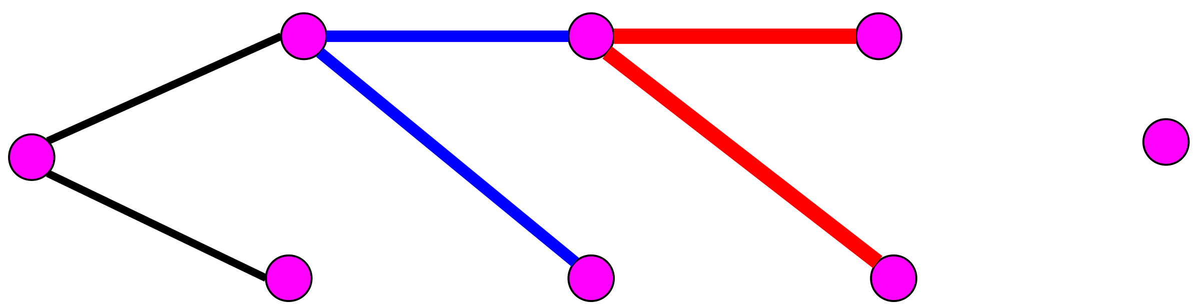


Construction with cost perturbation

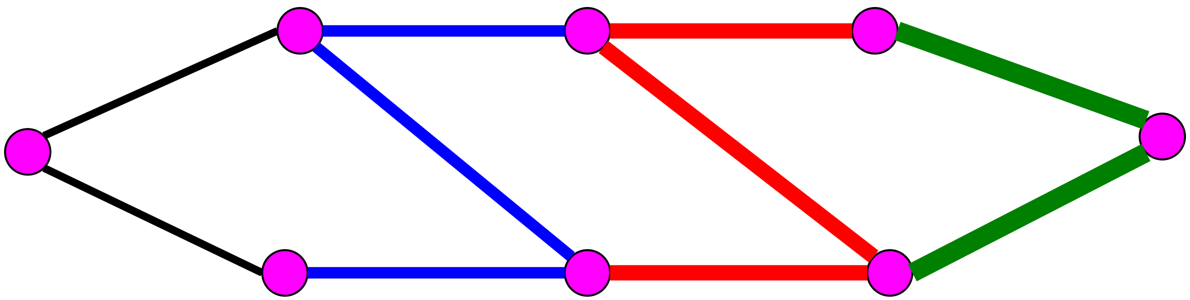


Perturb with costs increasing from top to bottom.

W( ) < W( ) < W( ) < W( )

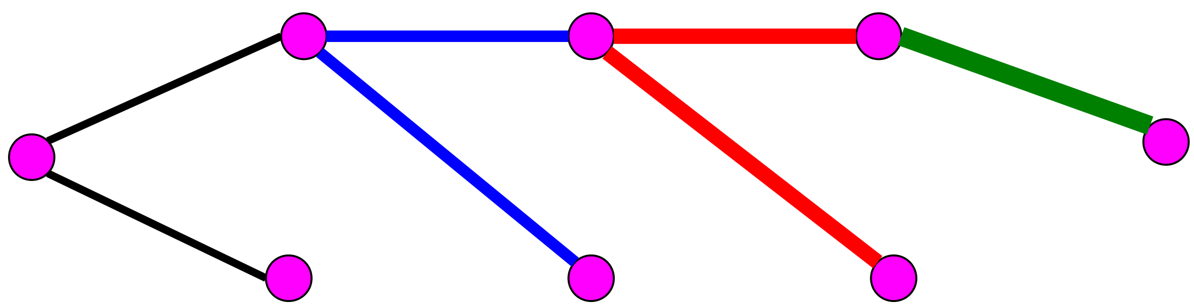


Construction with cost perturbation

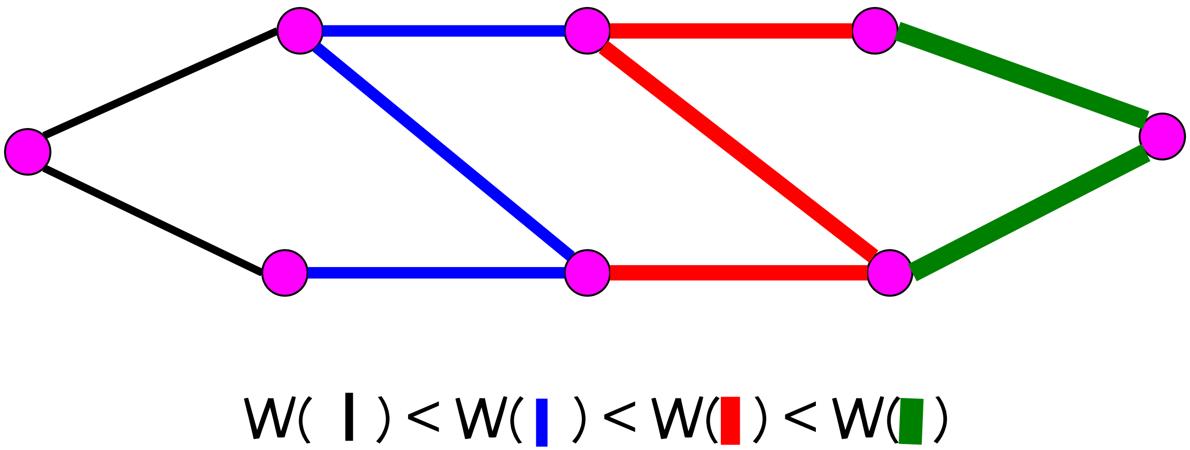


Perturb with costs increasing from top to bottom.

W( ) < W( ) < W( ) < W( )

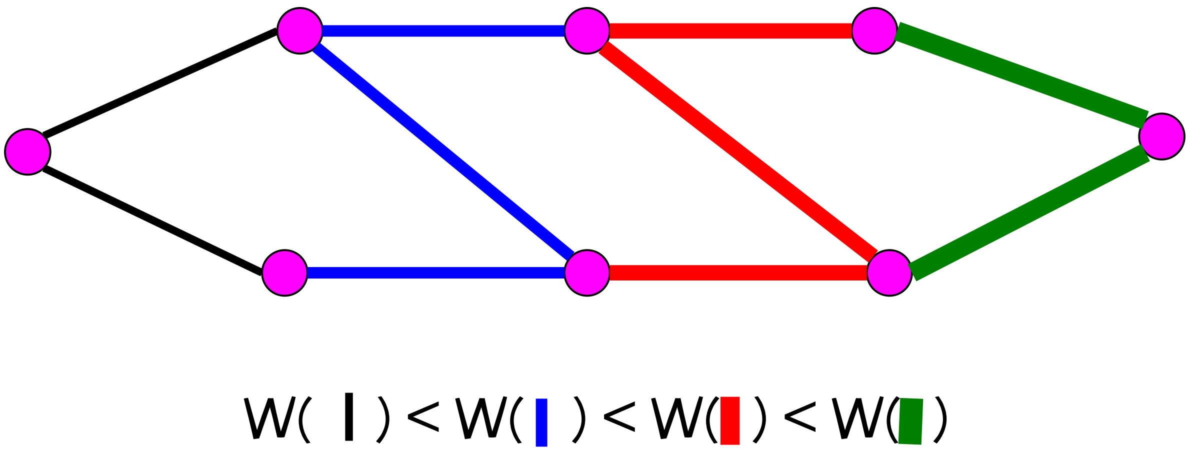


Construction with cost perturbation

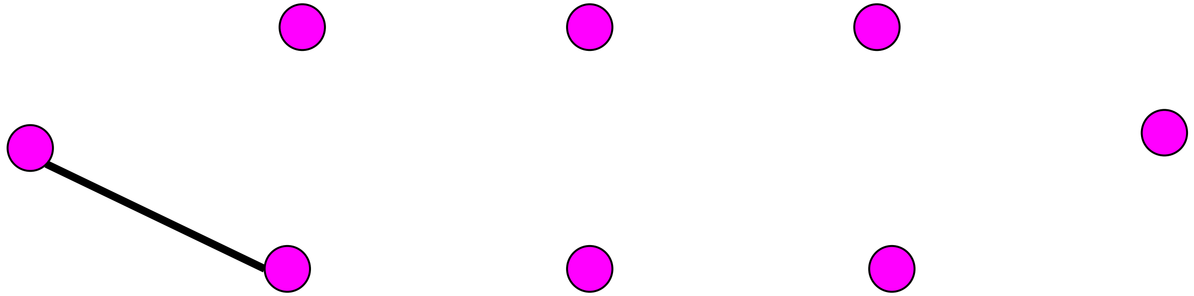


Perturb with costs increasing from bottom to top.

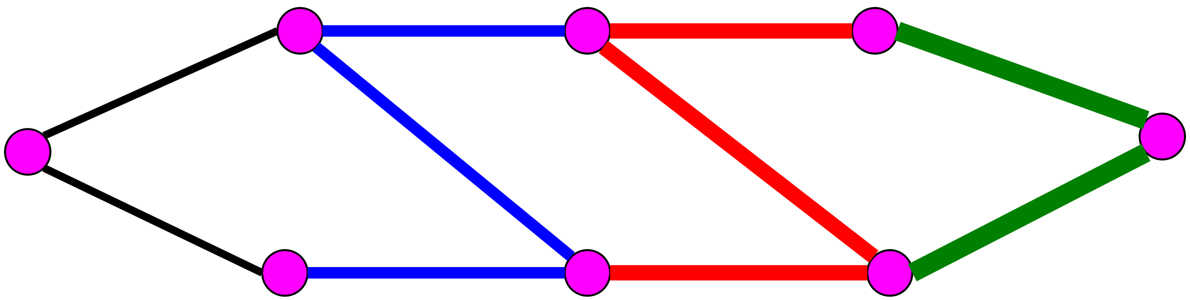
Construction with cost perturbation



Perturb with costs increasing from bottom to top.

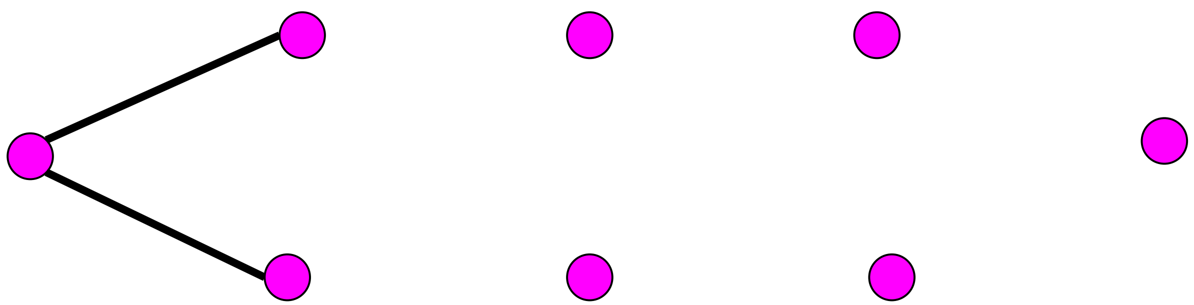


Construction with cost perturbation

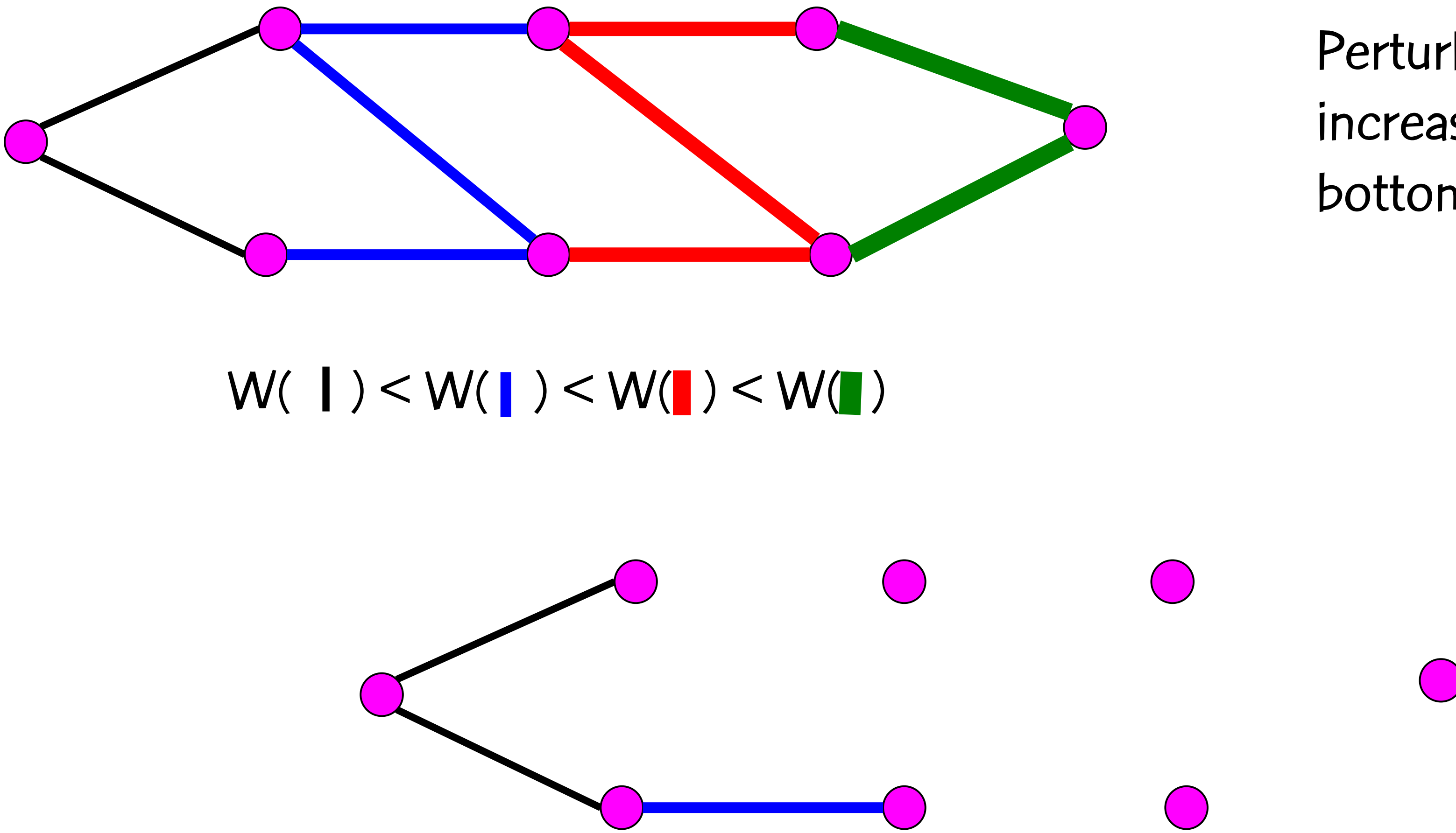


Perturb with costs increasing from bottom to top.

$W(\,|\,) < W(\,|\,) < W(\,|\,) < W(\,|\,)$

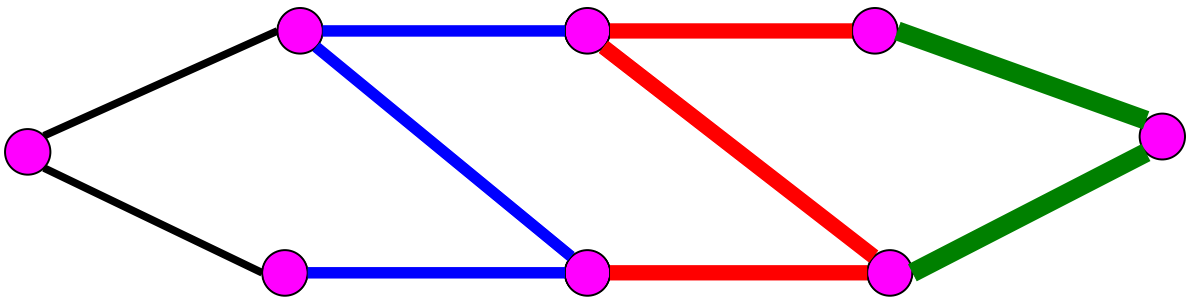


Construction with cost perturbation



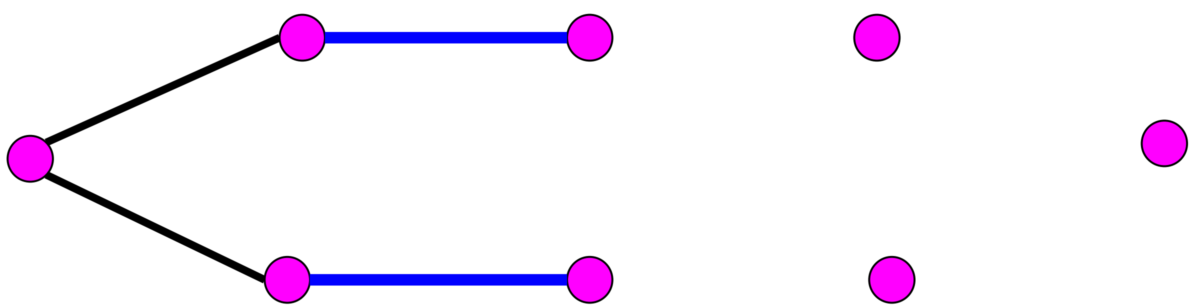
Perturb with costs increasing from bottom to top.

Construction with cost perturbation

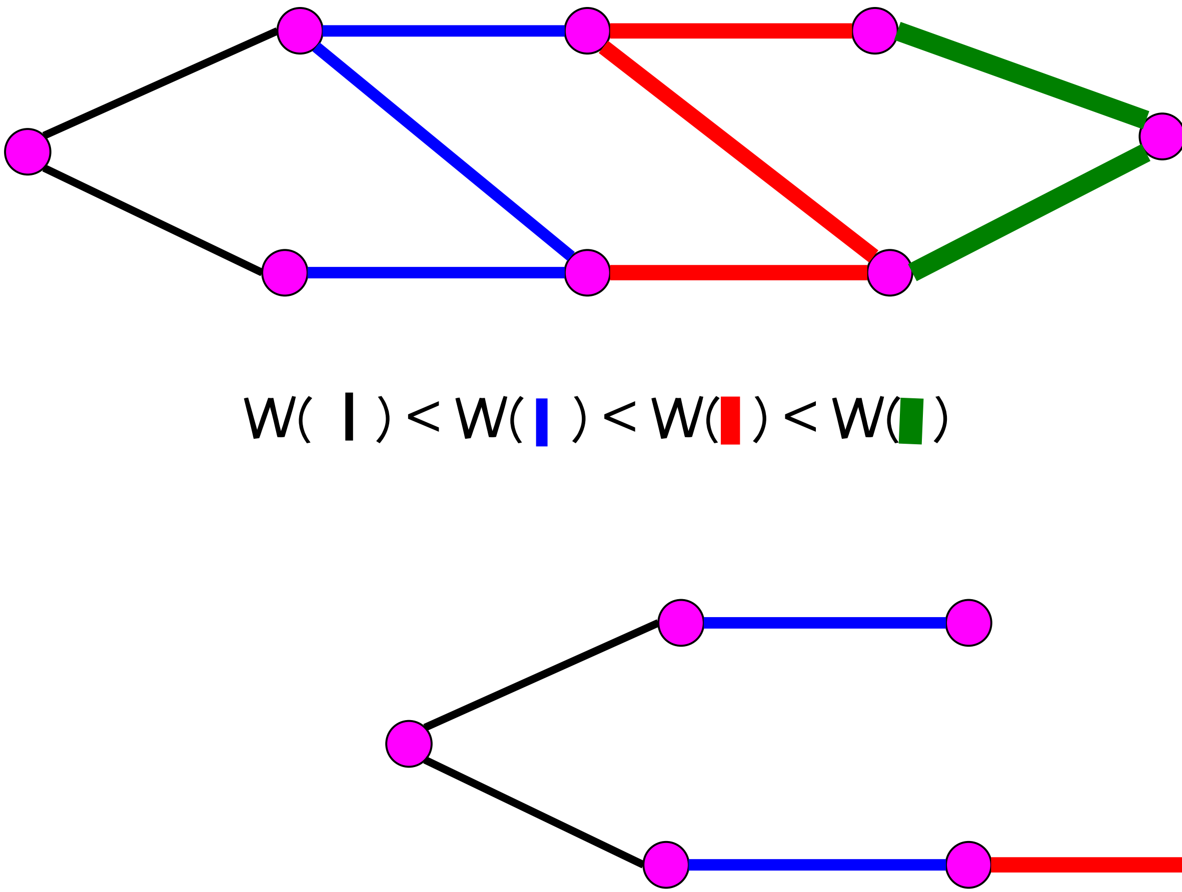


Perturb with costs increasing from bottom to top.

W( ) < W( ) < W( ) < W( )

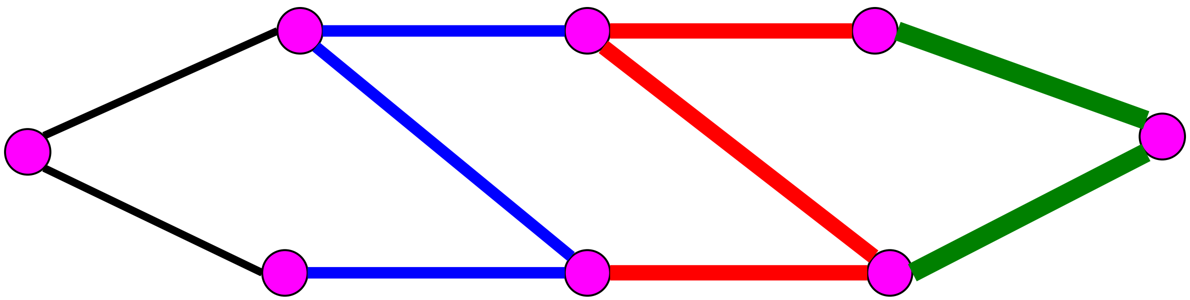


Construction with cost perturbation



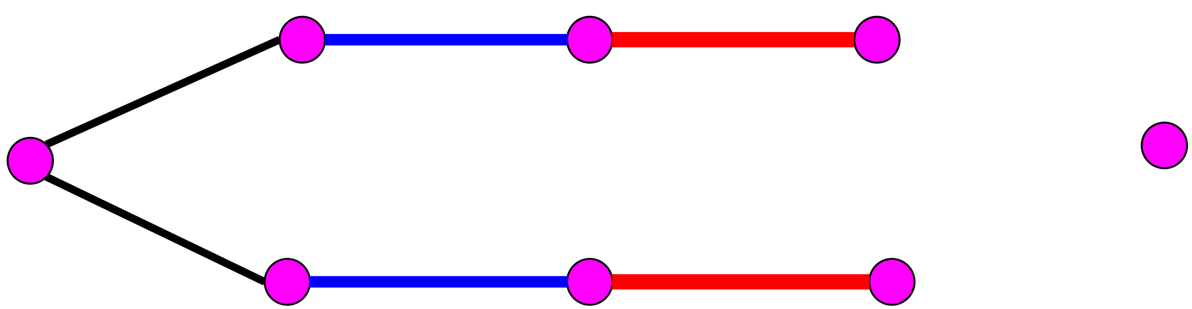
Perturb with costs increasing from bottom to top.

Construction with cost perturbation

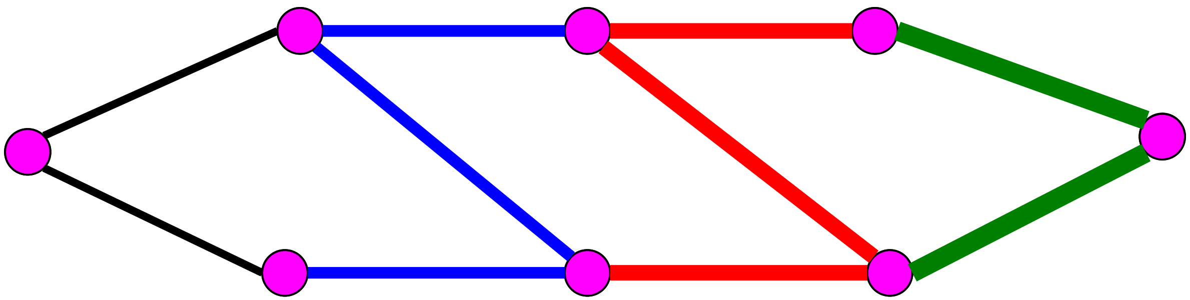


Perturb with costs increasing from bottom to top.

W( ) < W( ) < W( ) < W( )

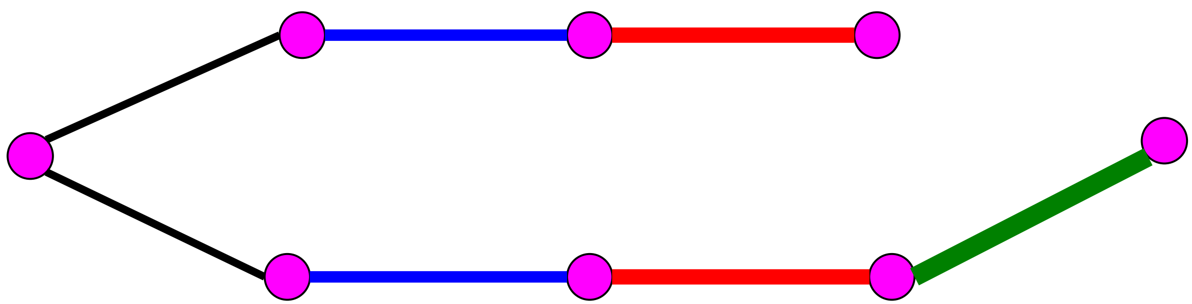


Construction with cost perturbation

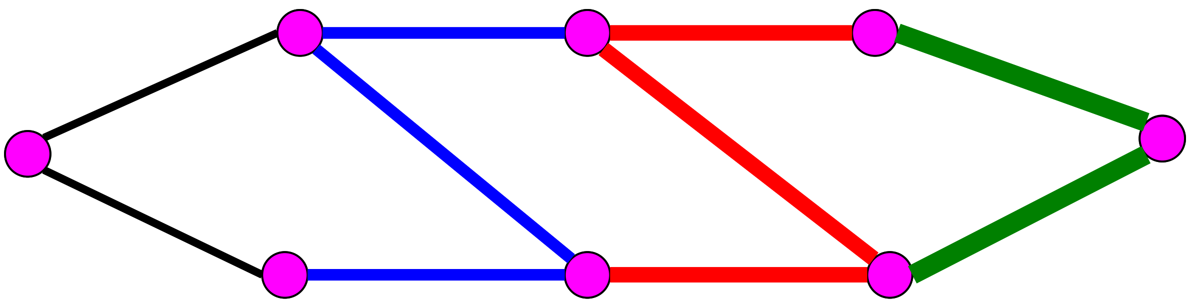


Perturb with costs increasing from bottom to top.

W( ) < W( ) < W( ) < W( )

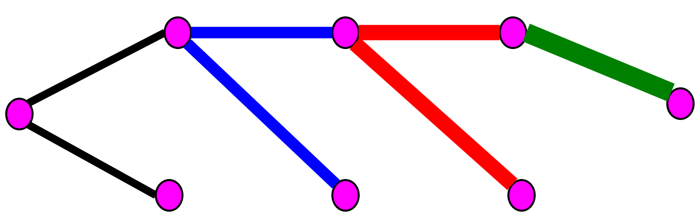


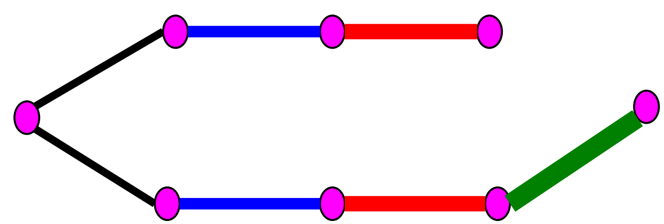
Construction with cost perturbation



Greedy heuristic generates two different spanning trees.

W( ) < W( ) < W( ) < W( )

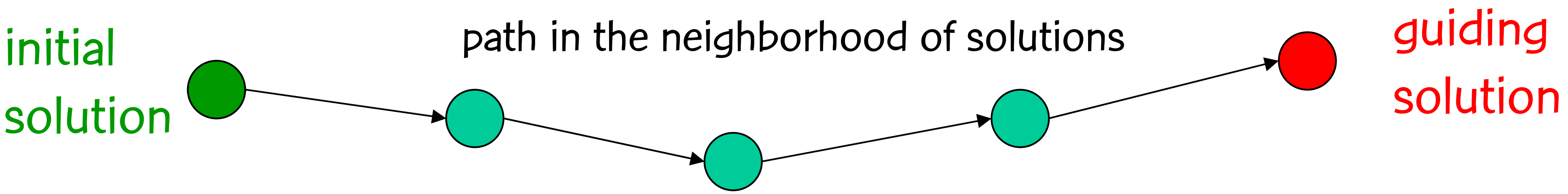




Path-relinking (PR)

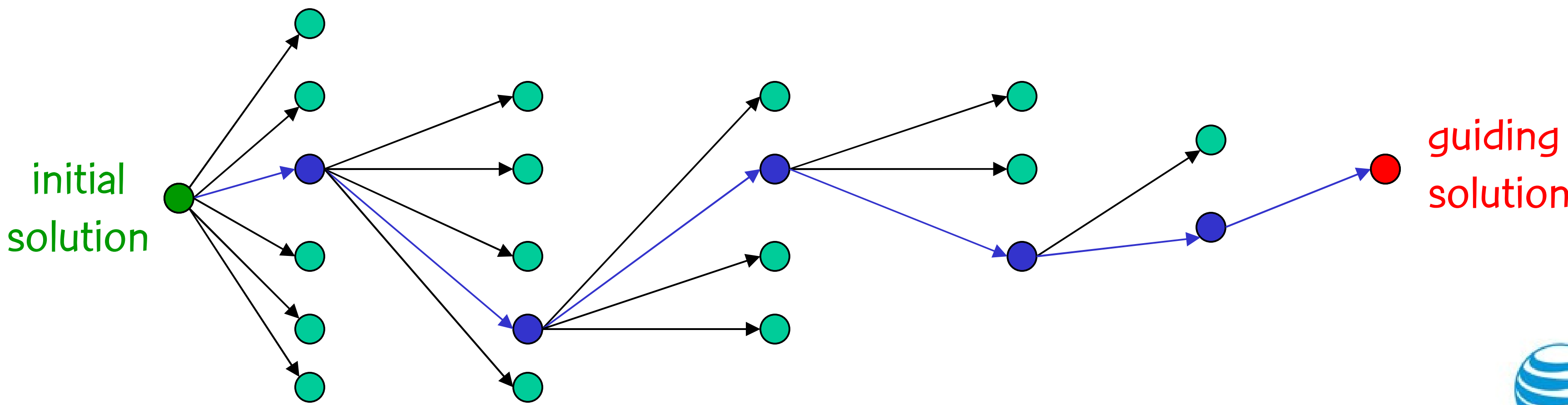
Path-relinking

- Intensification strategy exploring trajectories connecting high-quality (elite) solutions (Glover, 1996)



Path-relinking

- Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.
- At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



starting solution

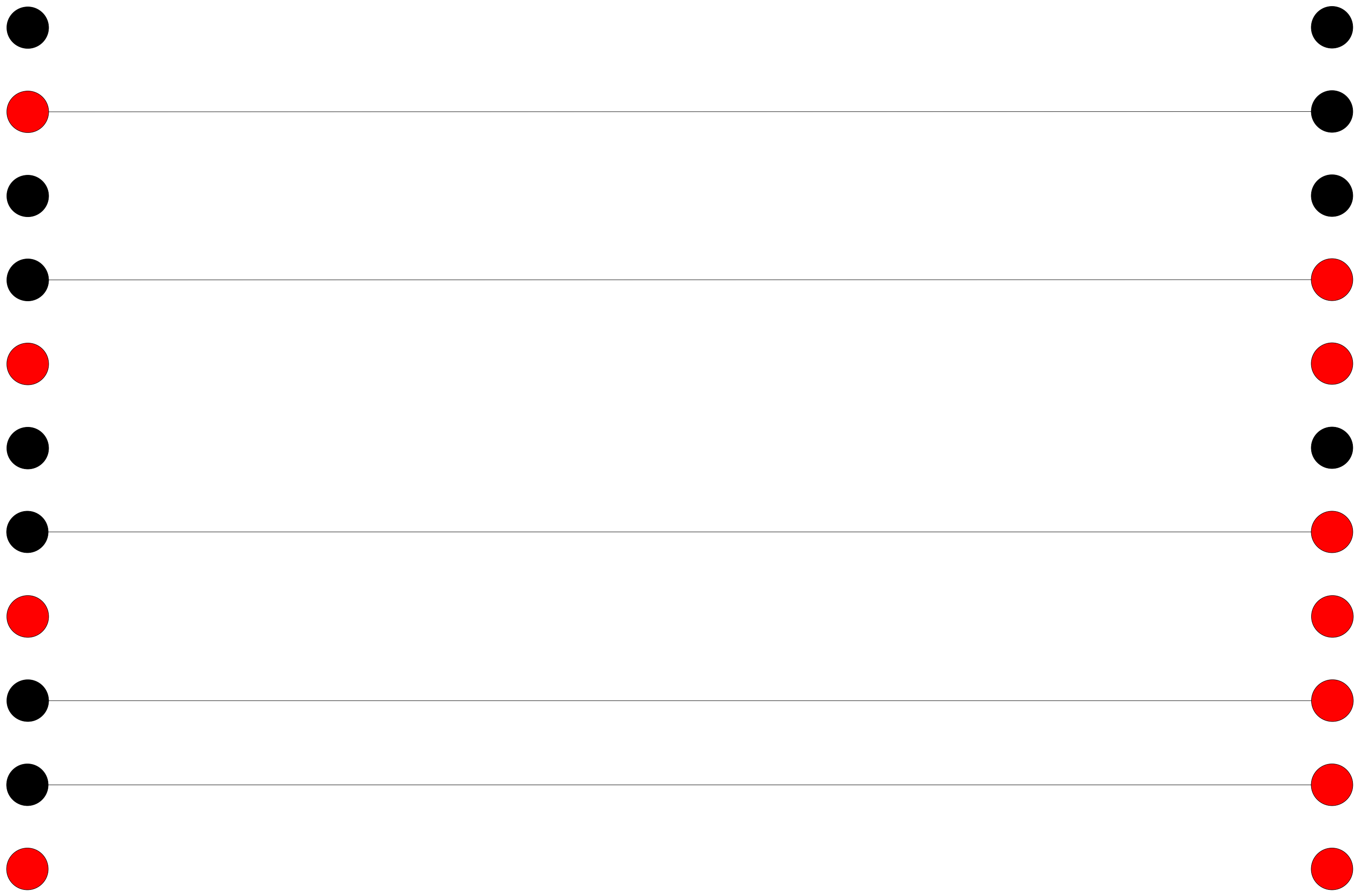
PR example

guiding solution

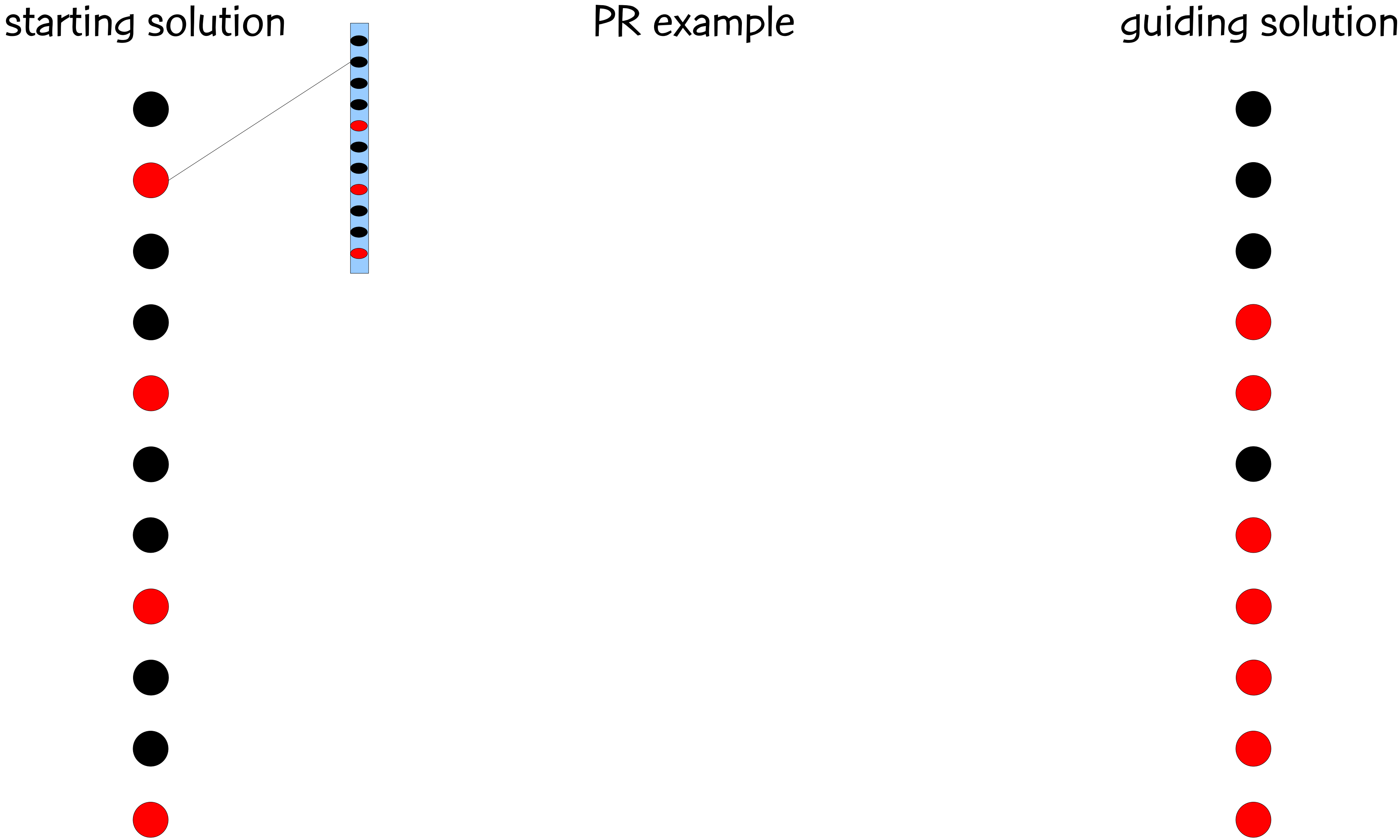
starting solution x

PR example

guiding solution y



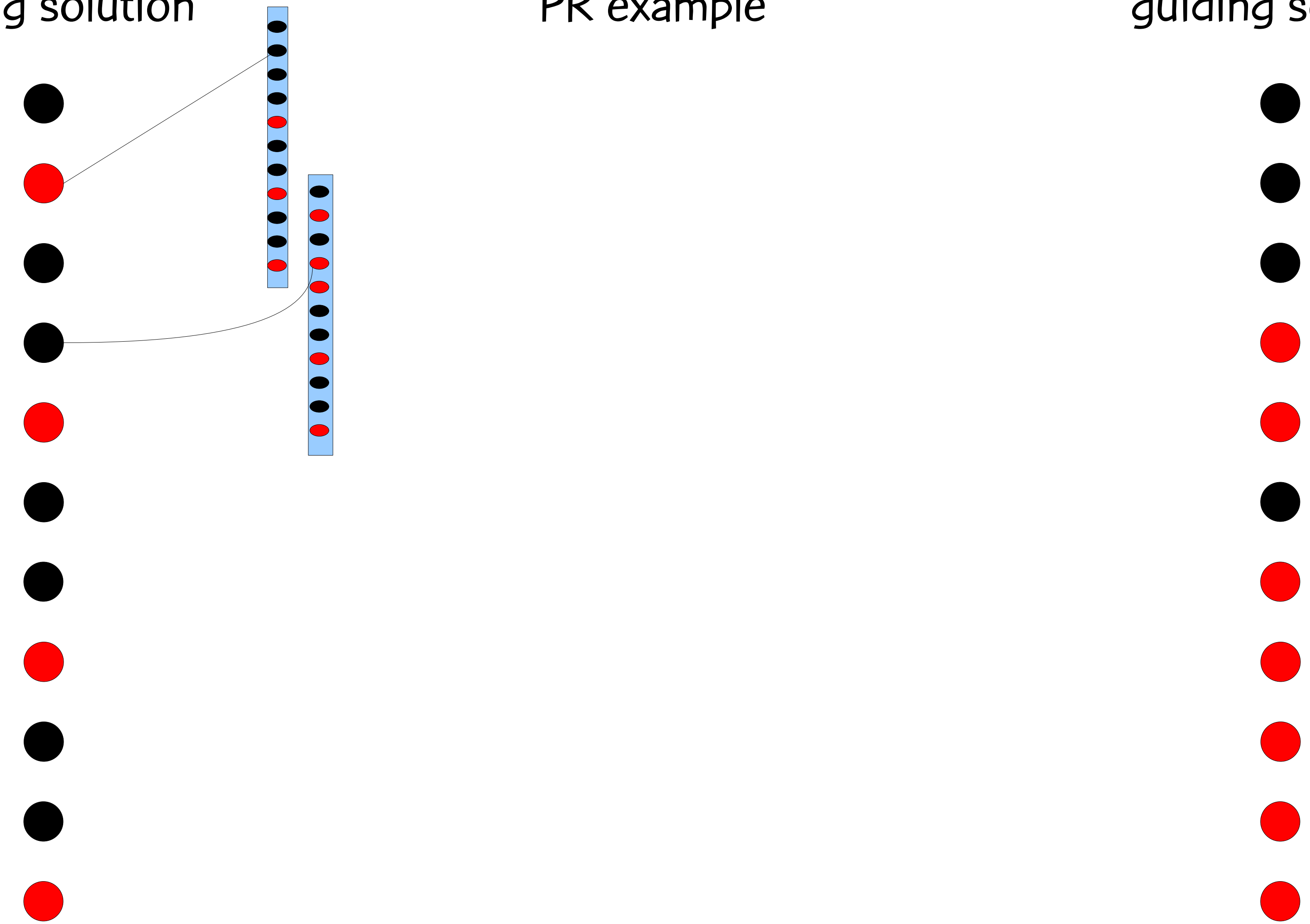
PR example



starting solution

PR example

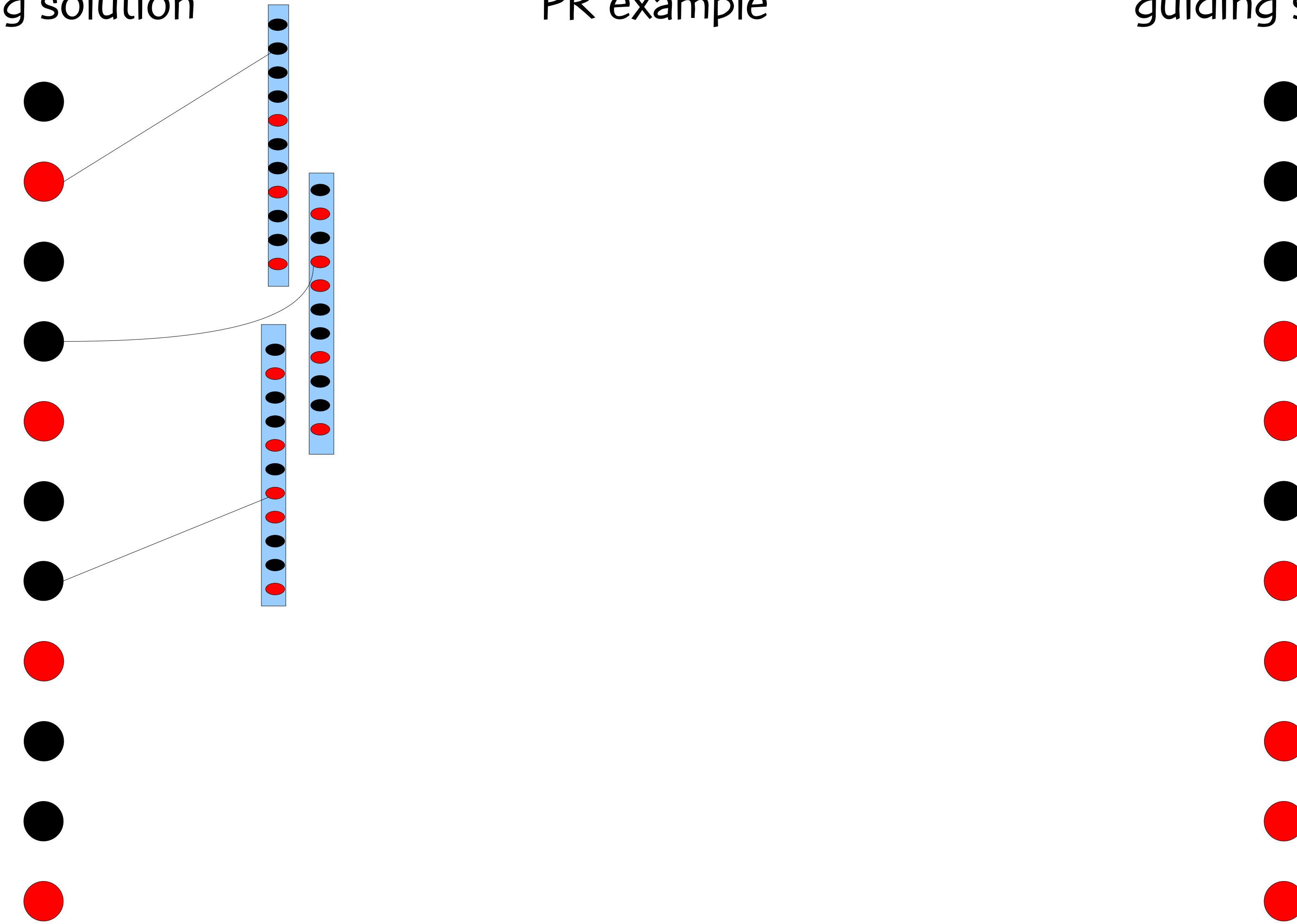
guiding solution



PR example

starting solution

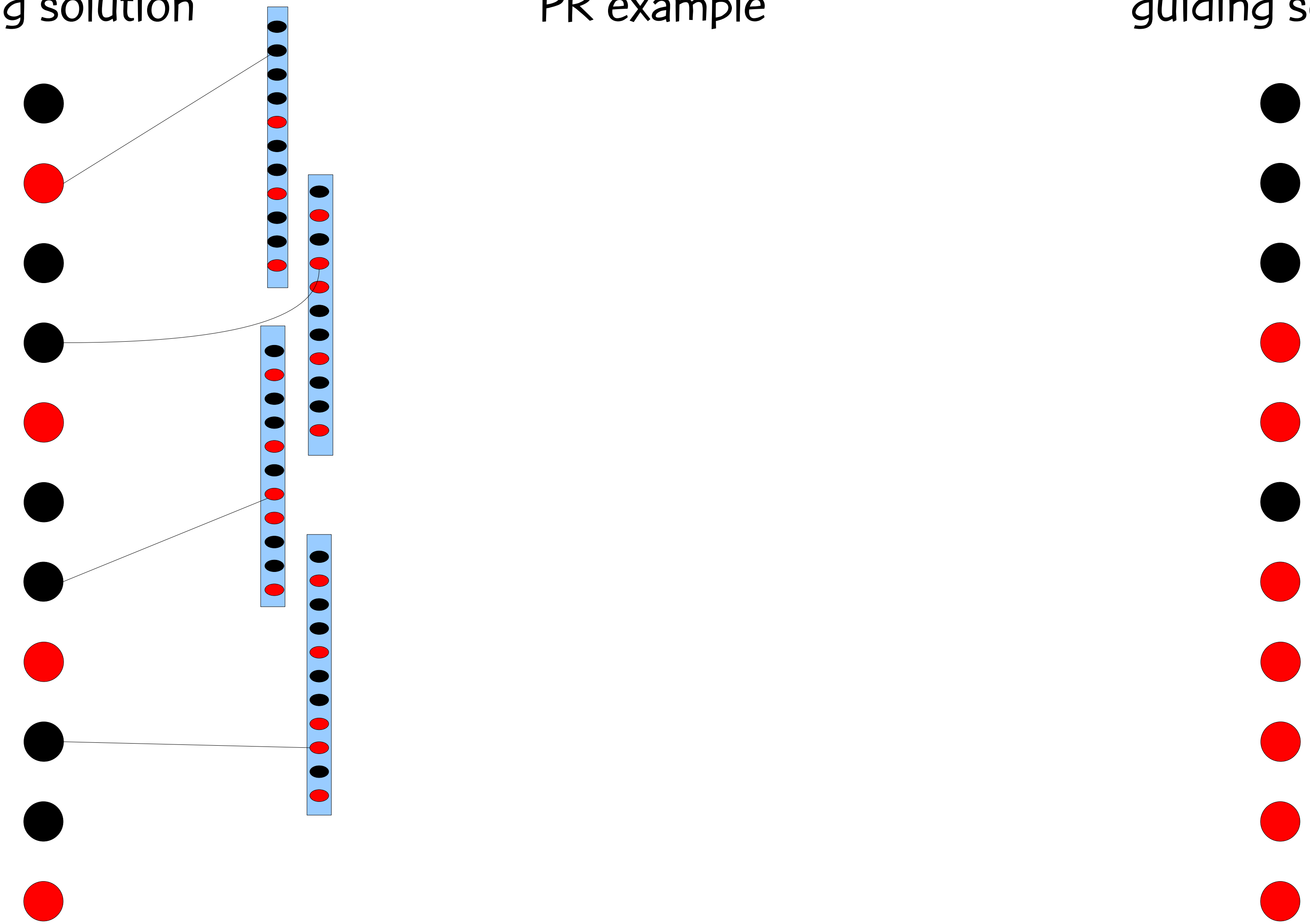
guiding solution



PR example

starting solution

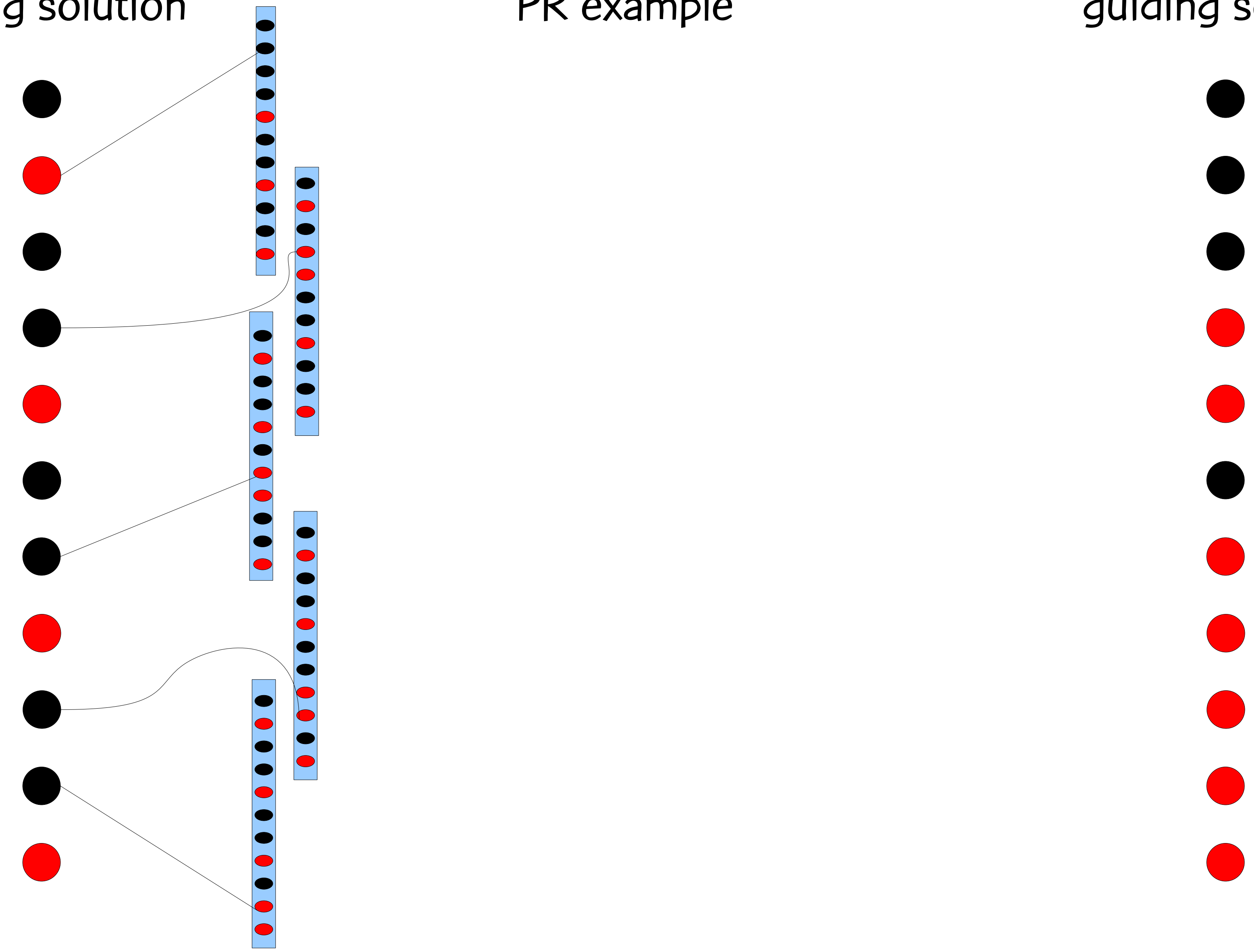
guiding solution



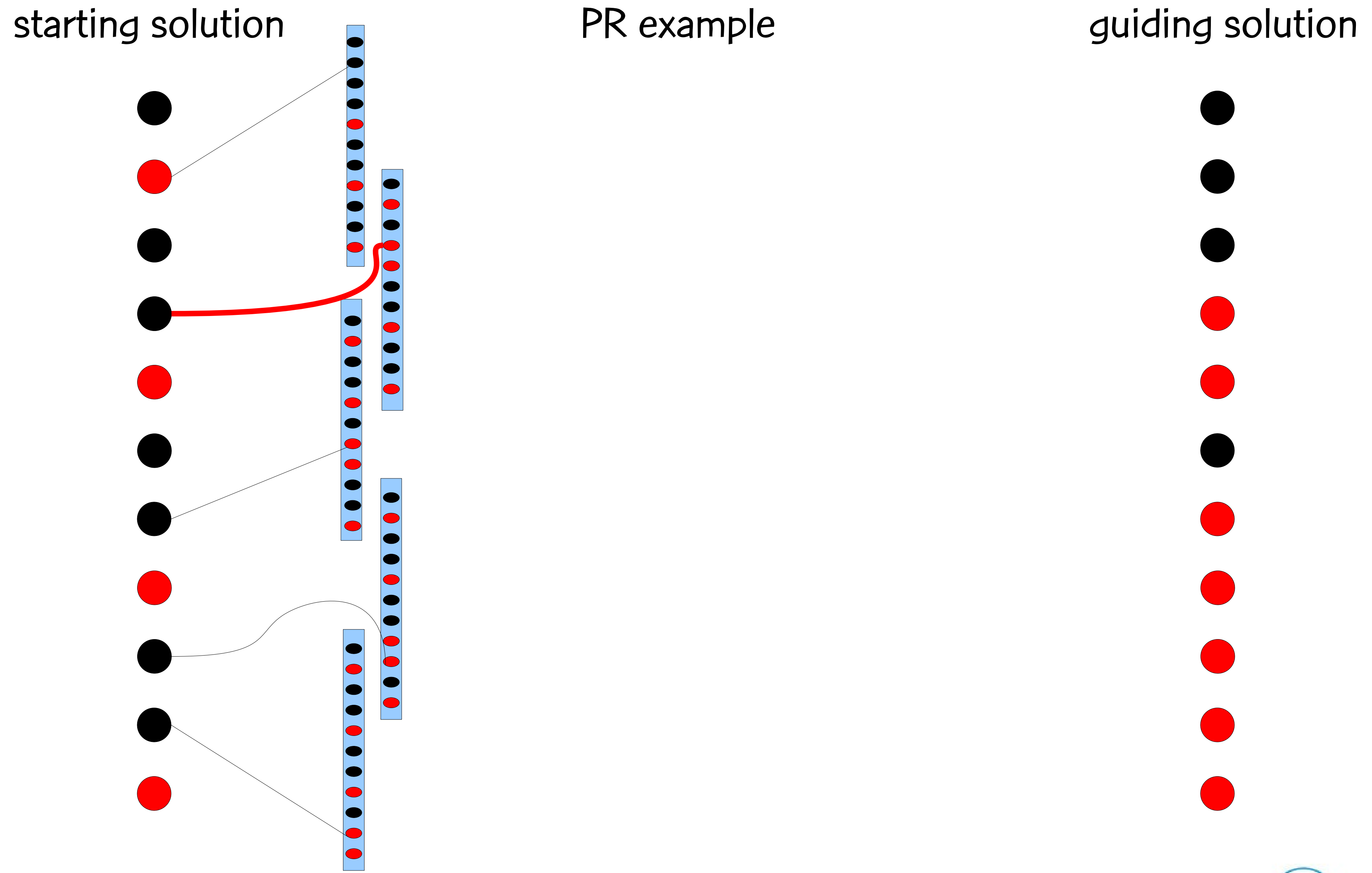
PR example

starting solution

guiding solution



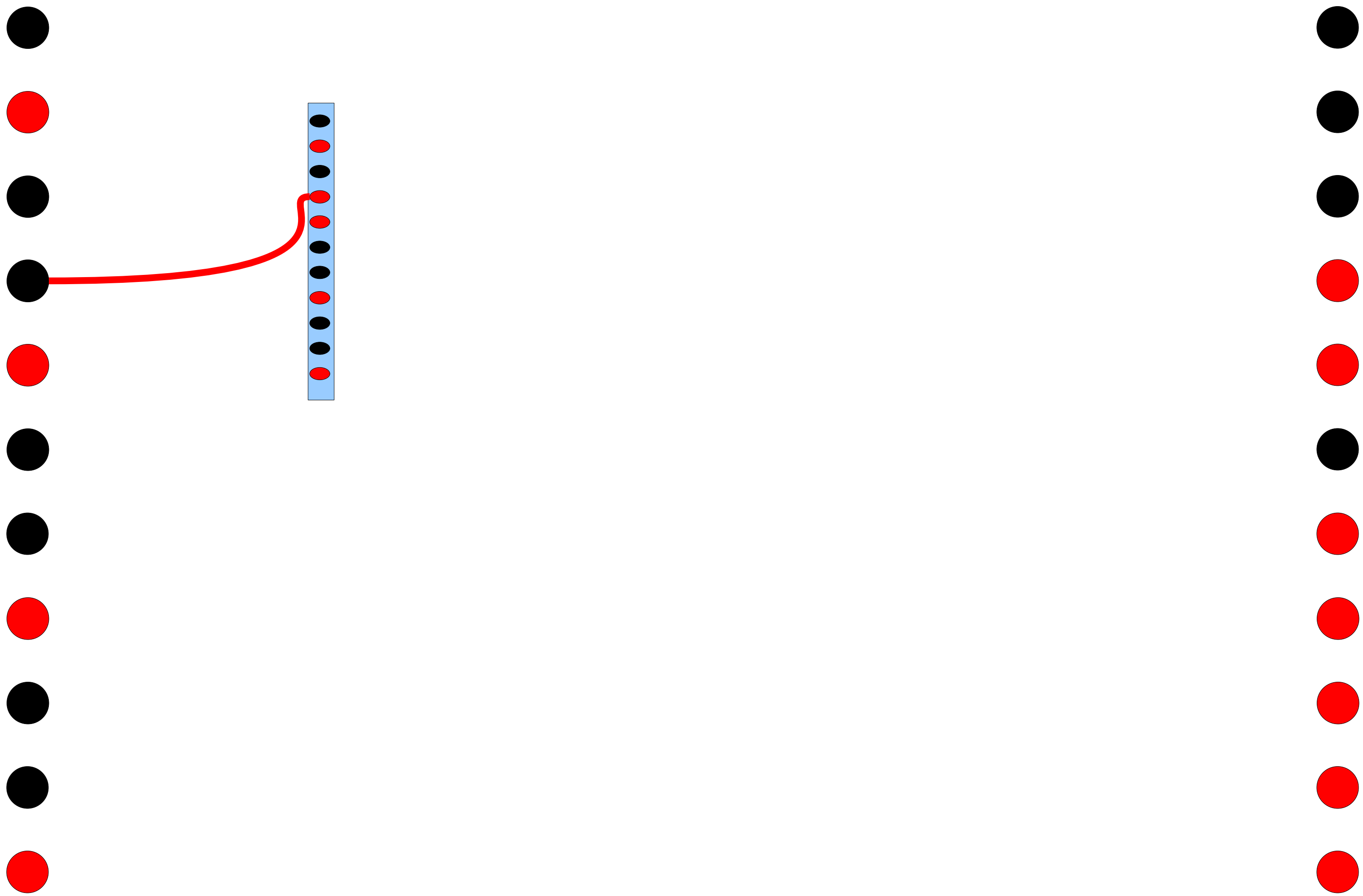
PR example



starting solution

PR example

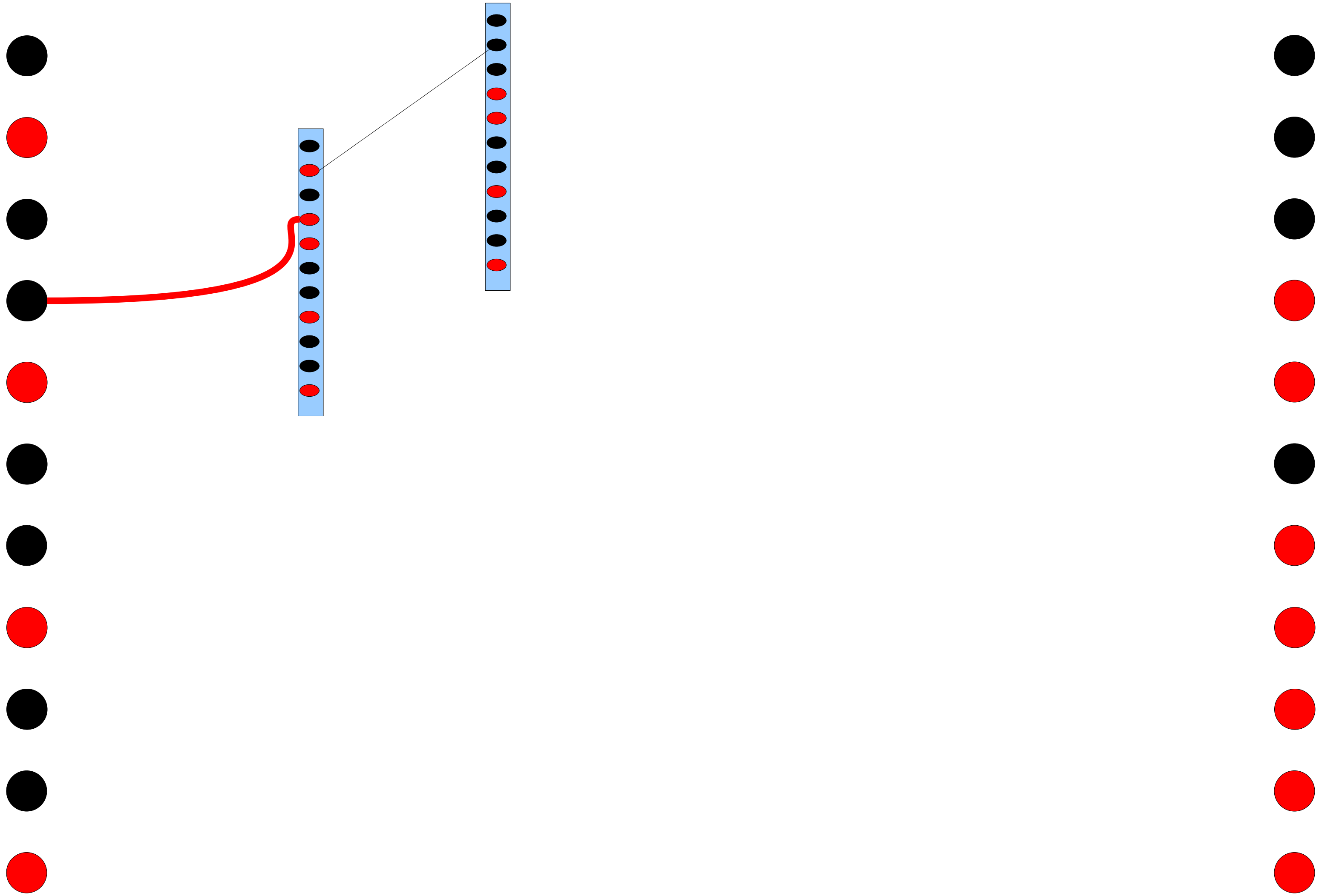
guiding solution



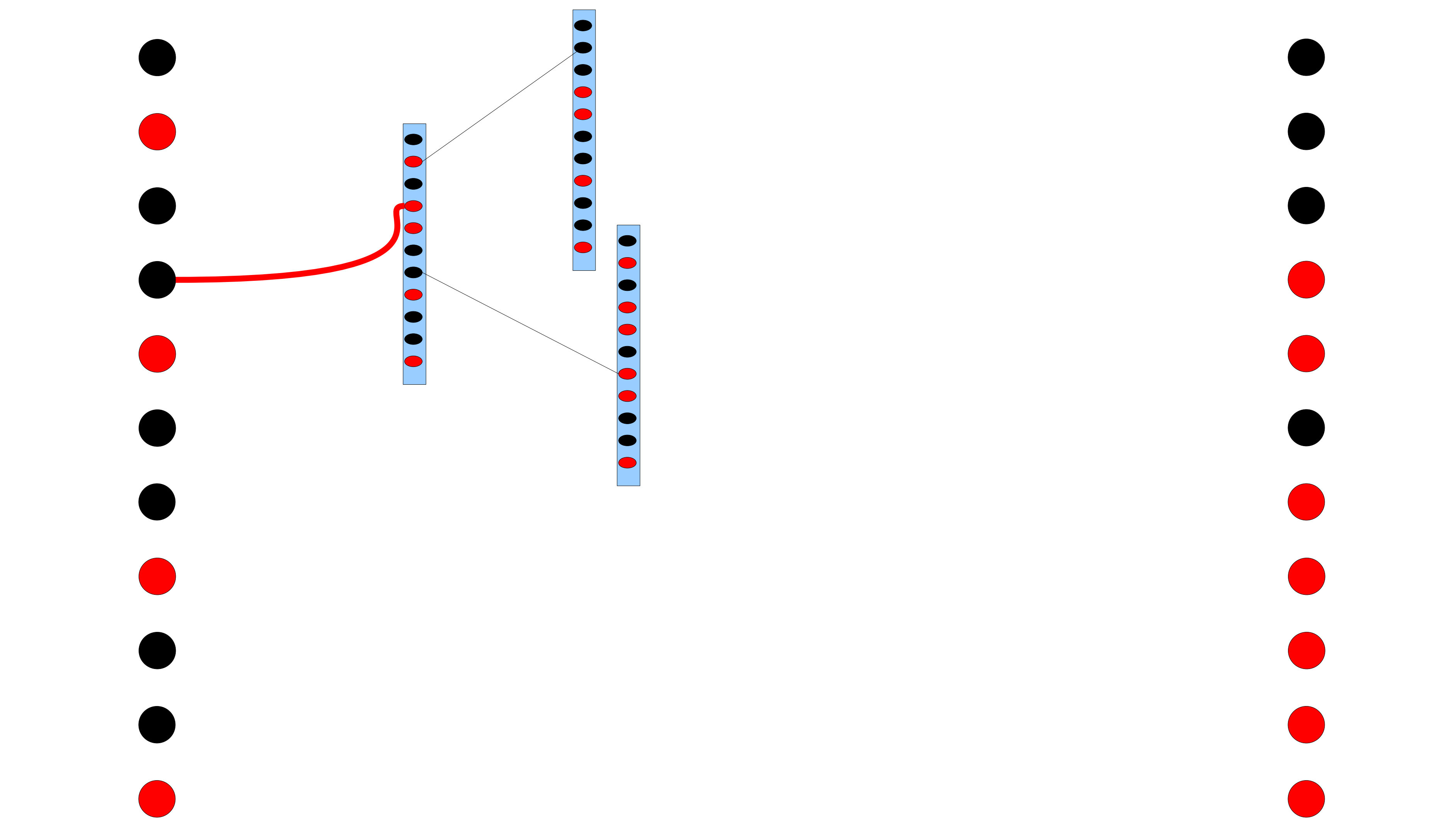
PR example

starting solution

guiding solution



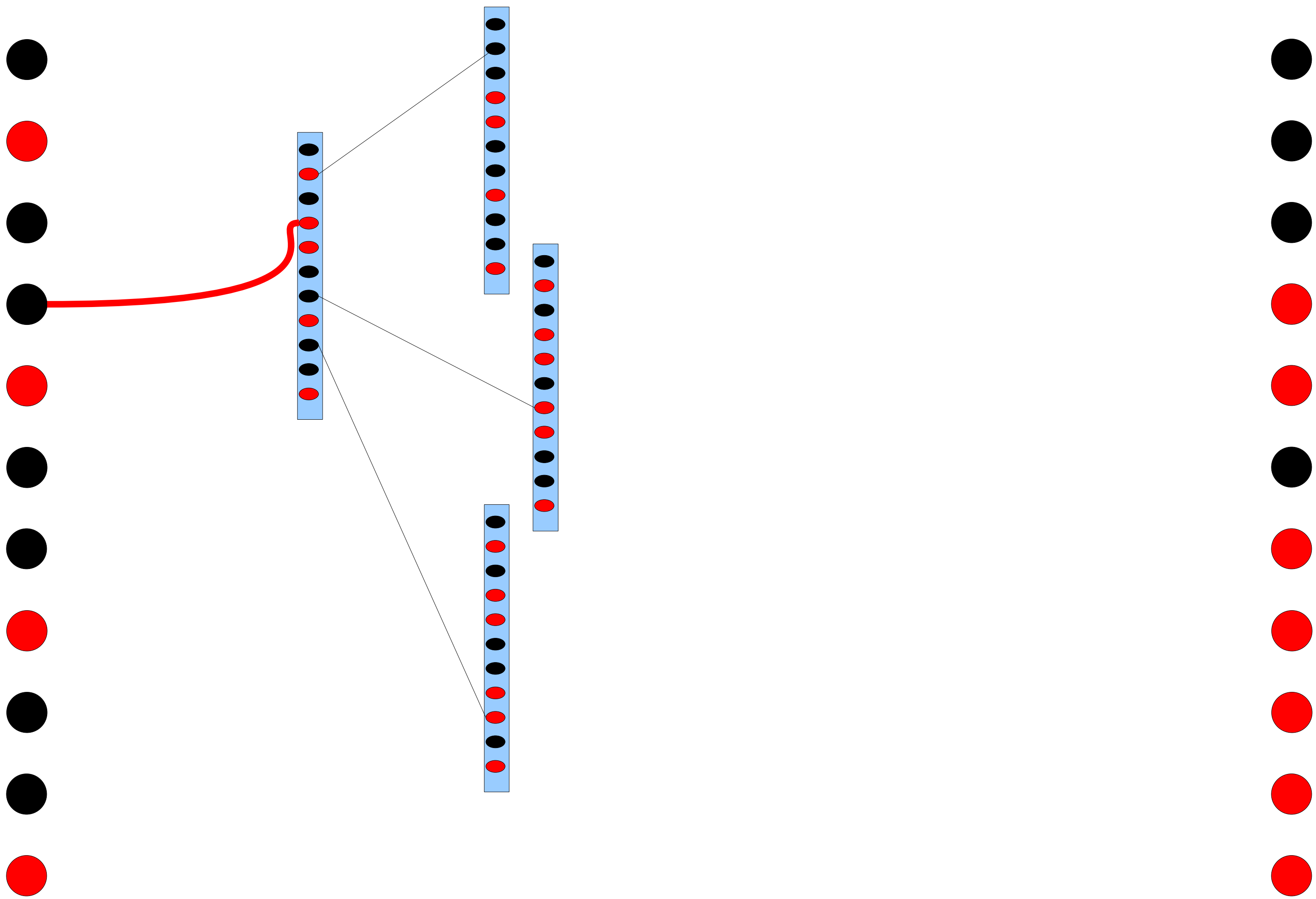
PR example



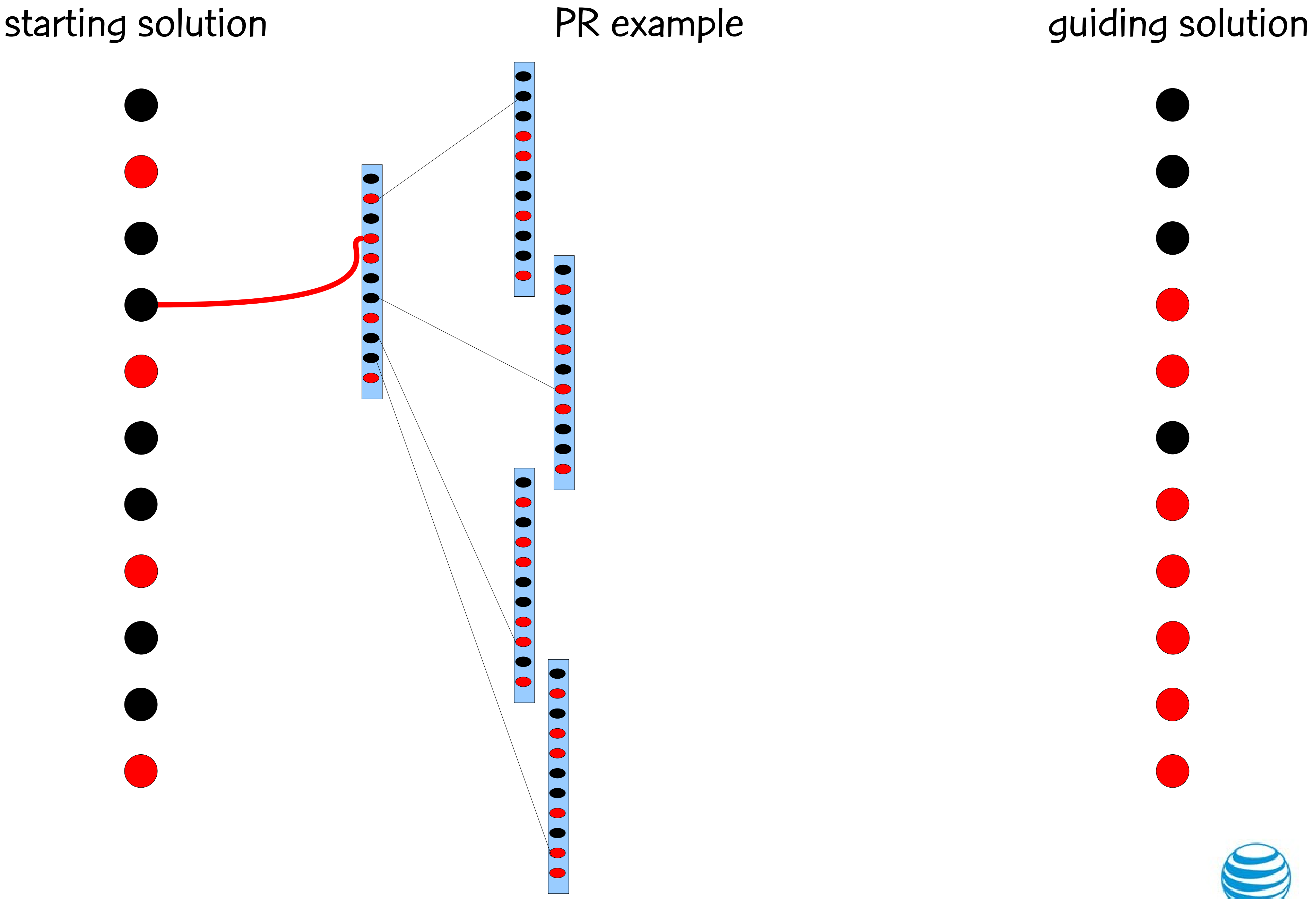
PR example

starting solution

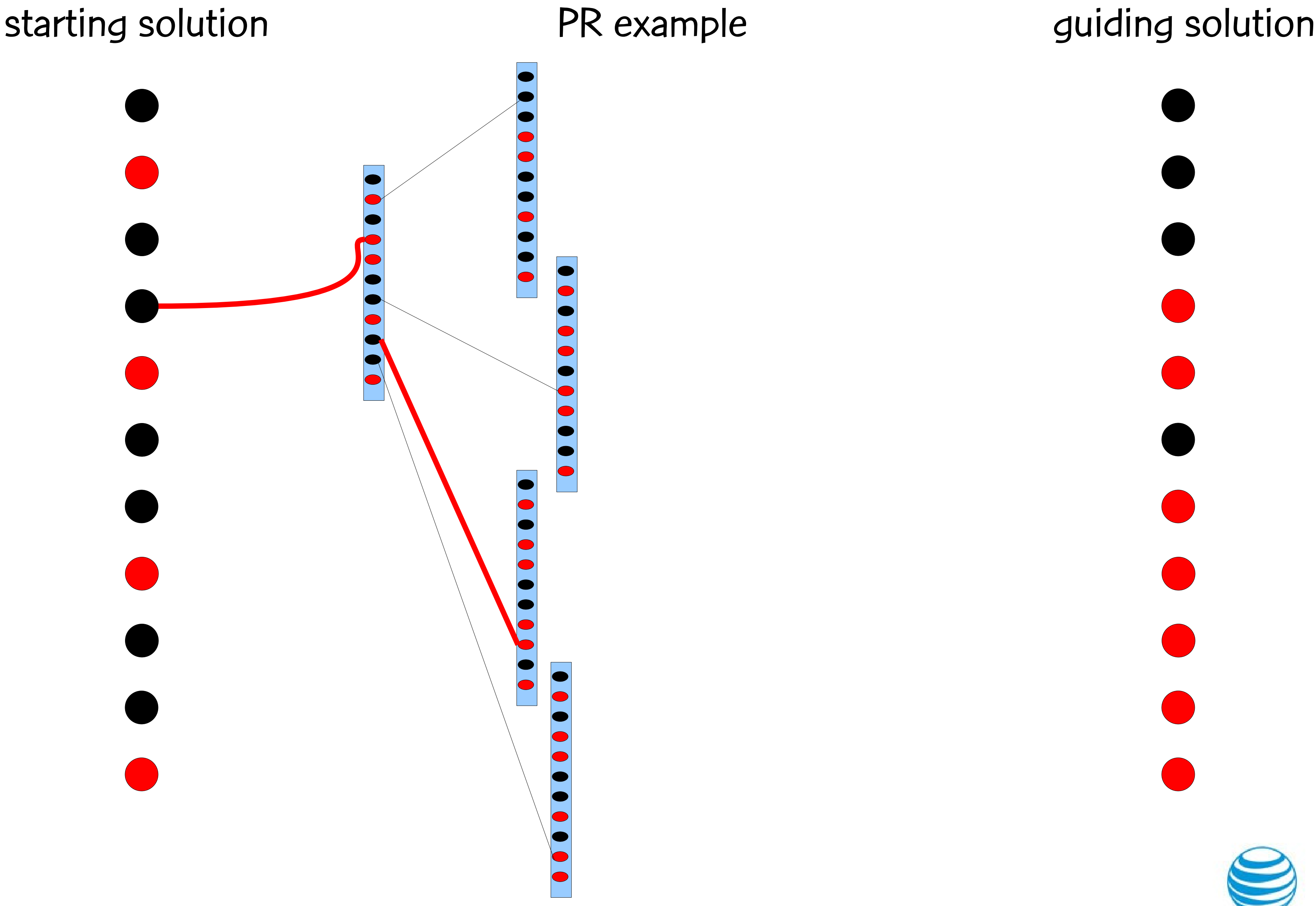
guiding solution



PR example



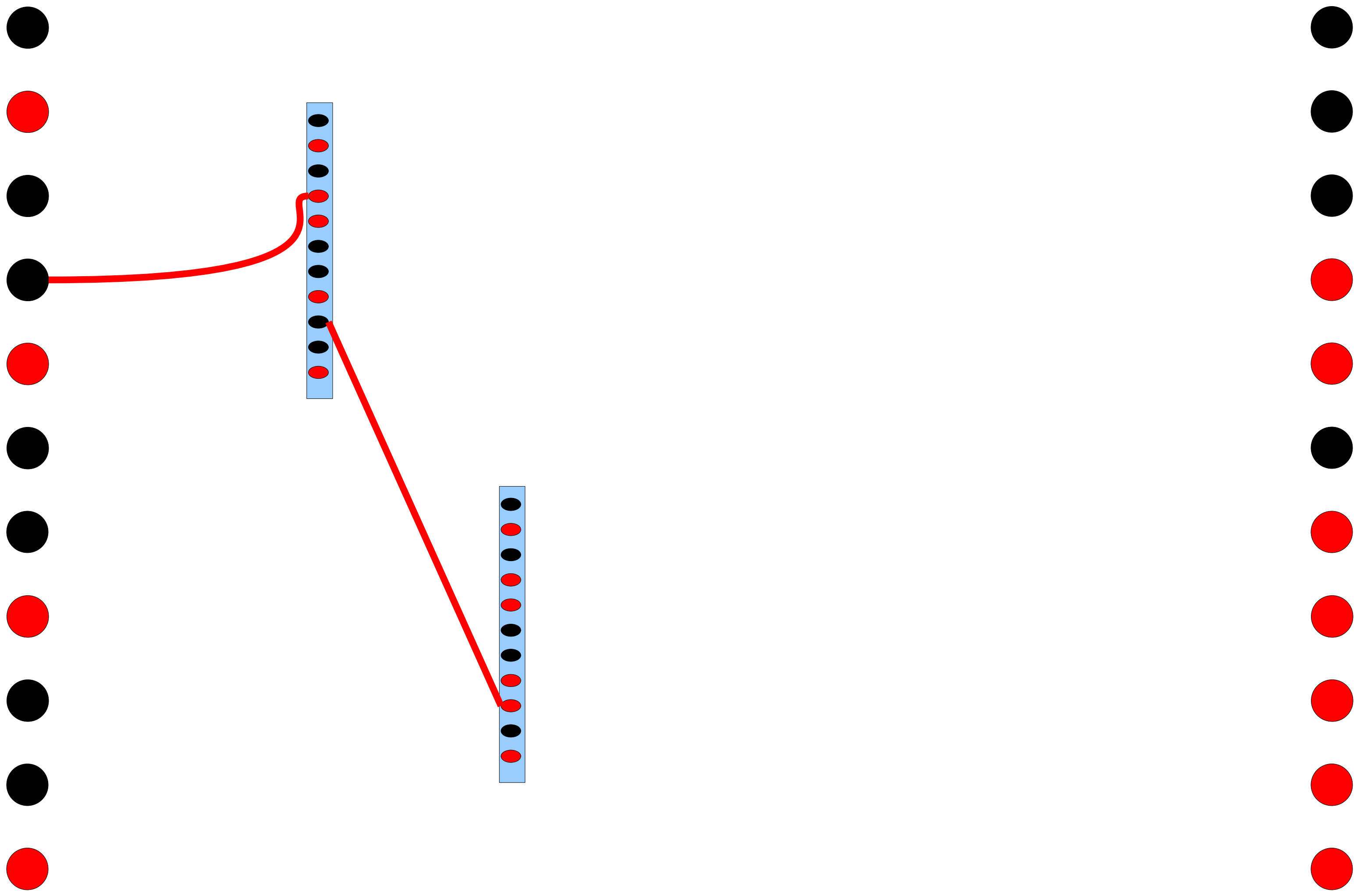
PR example



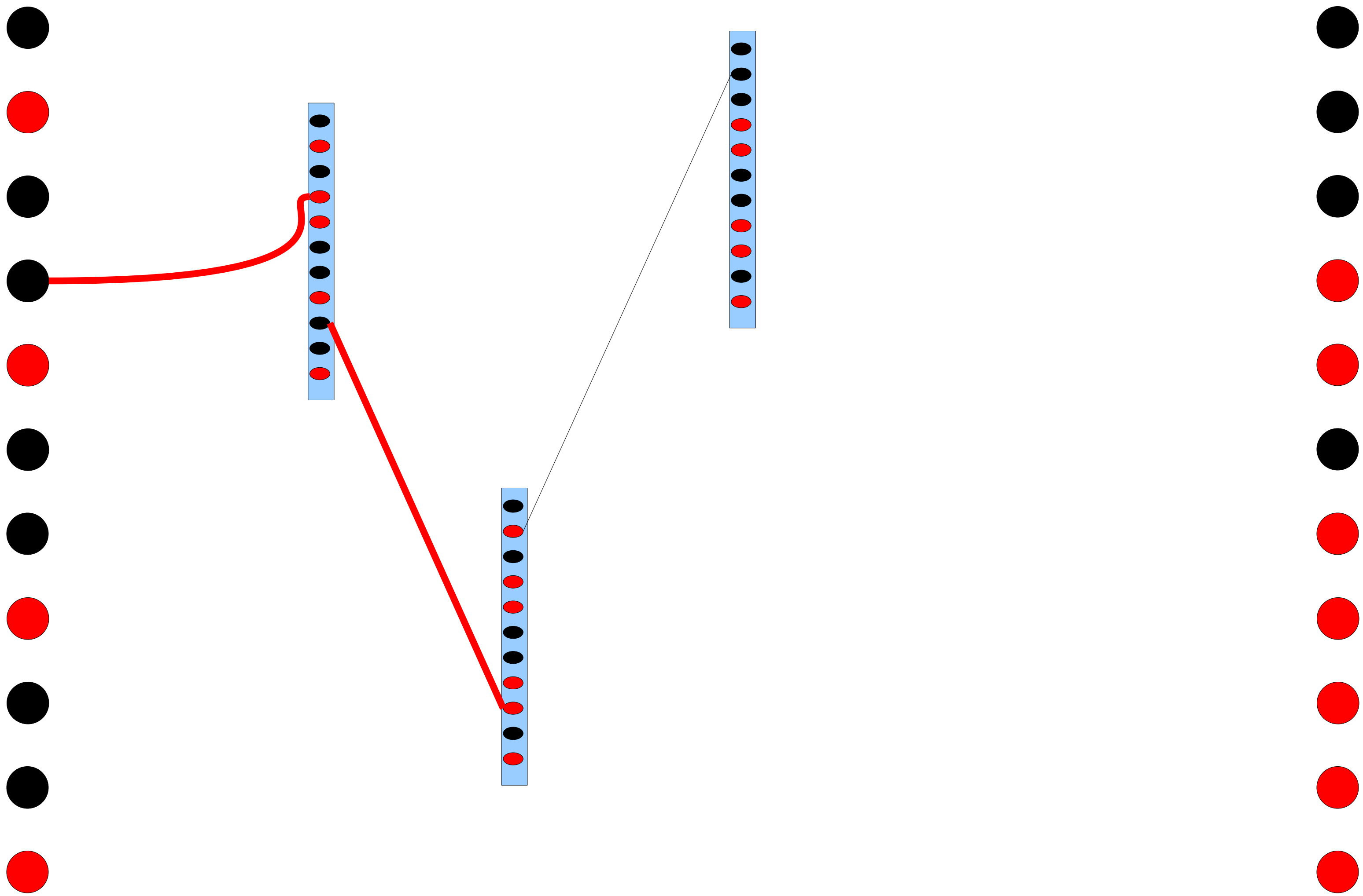
PR example

starting solution

guiding solution



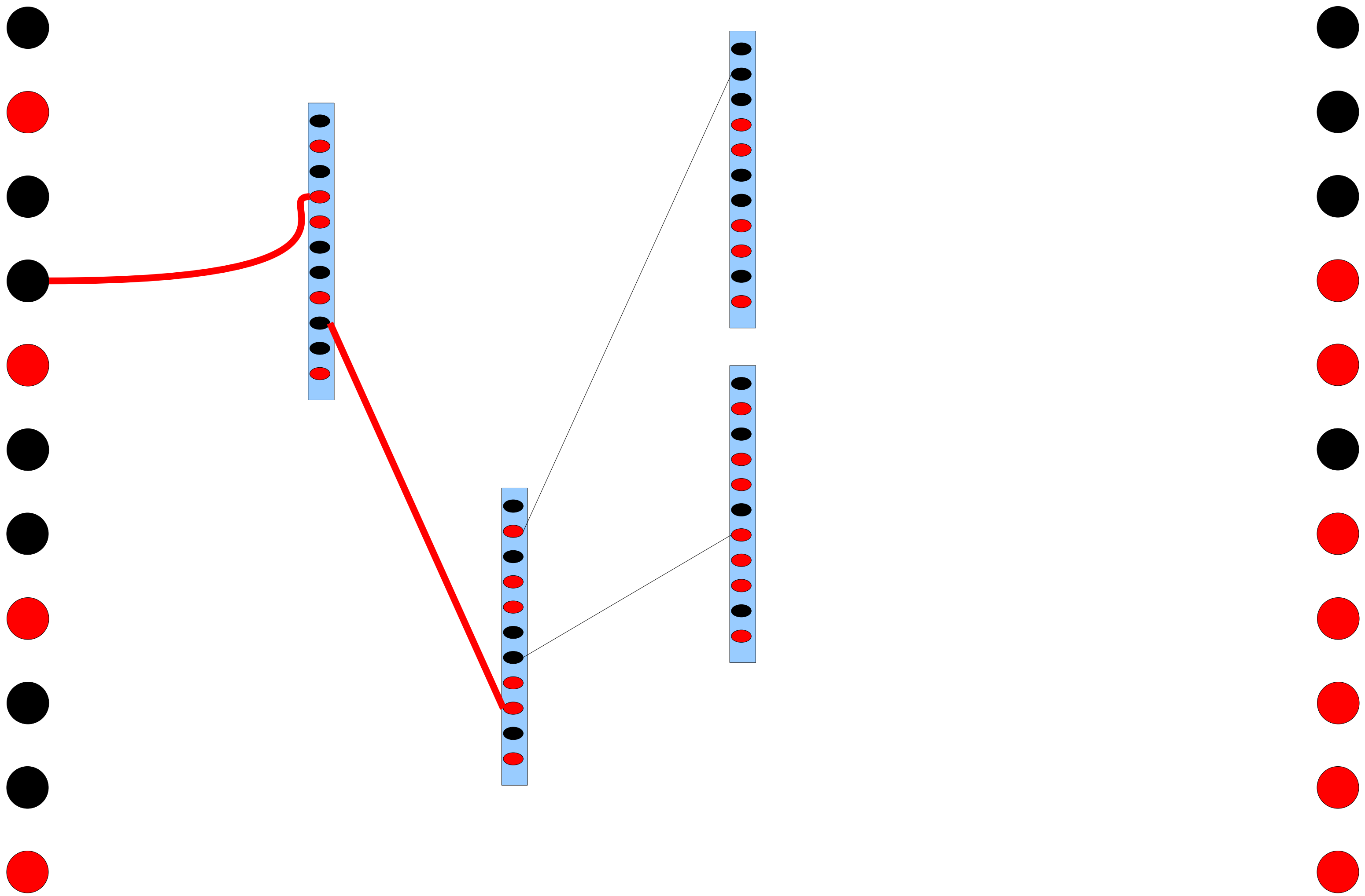
starting solution PR example guiding solution



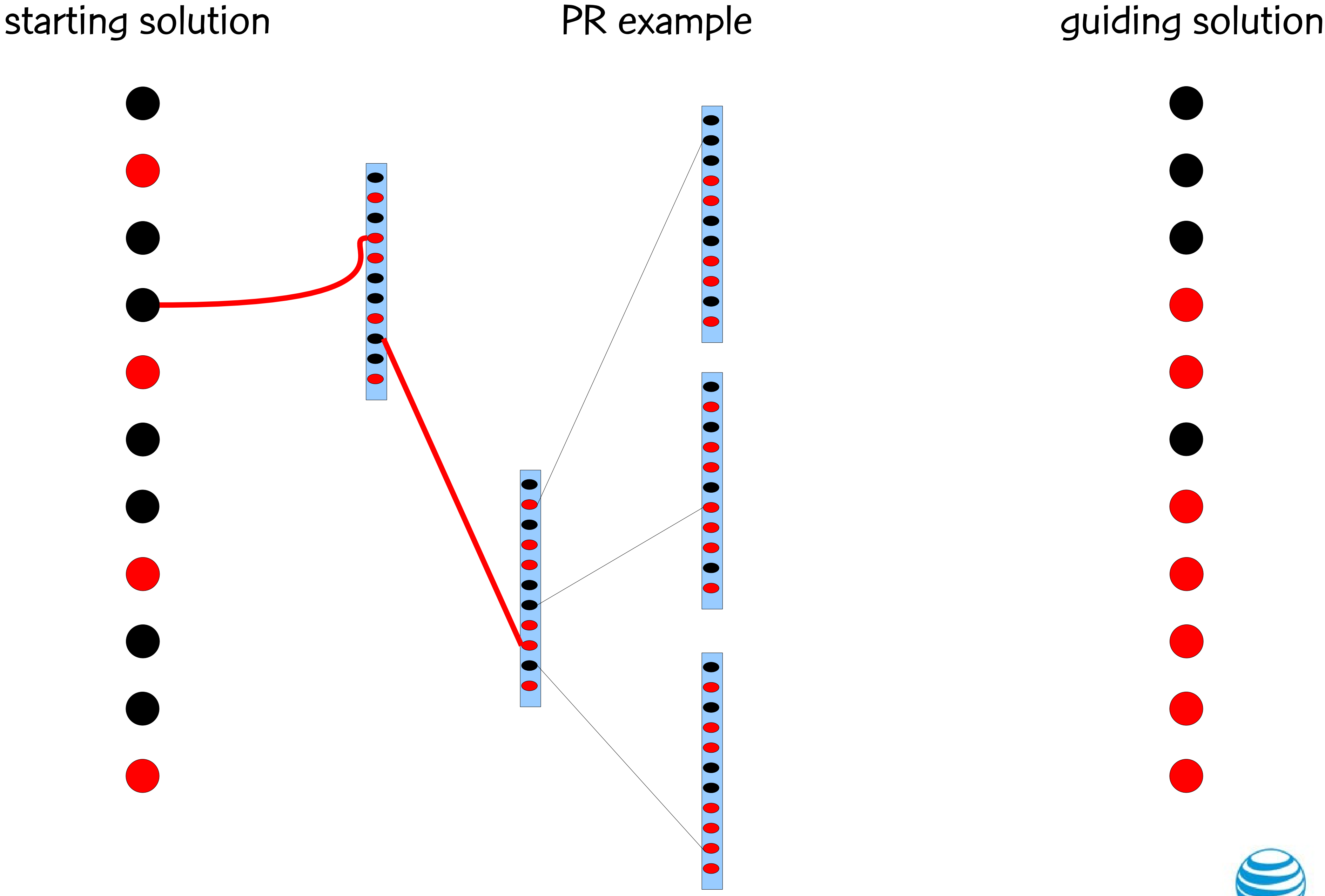
starting solution

PR example

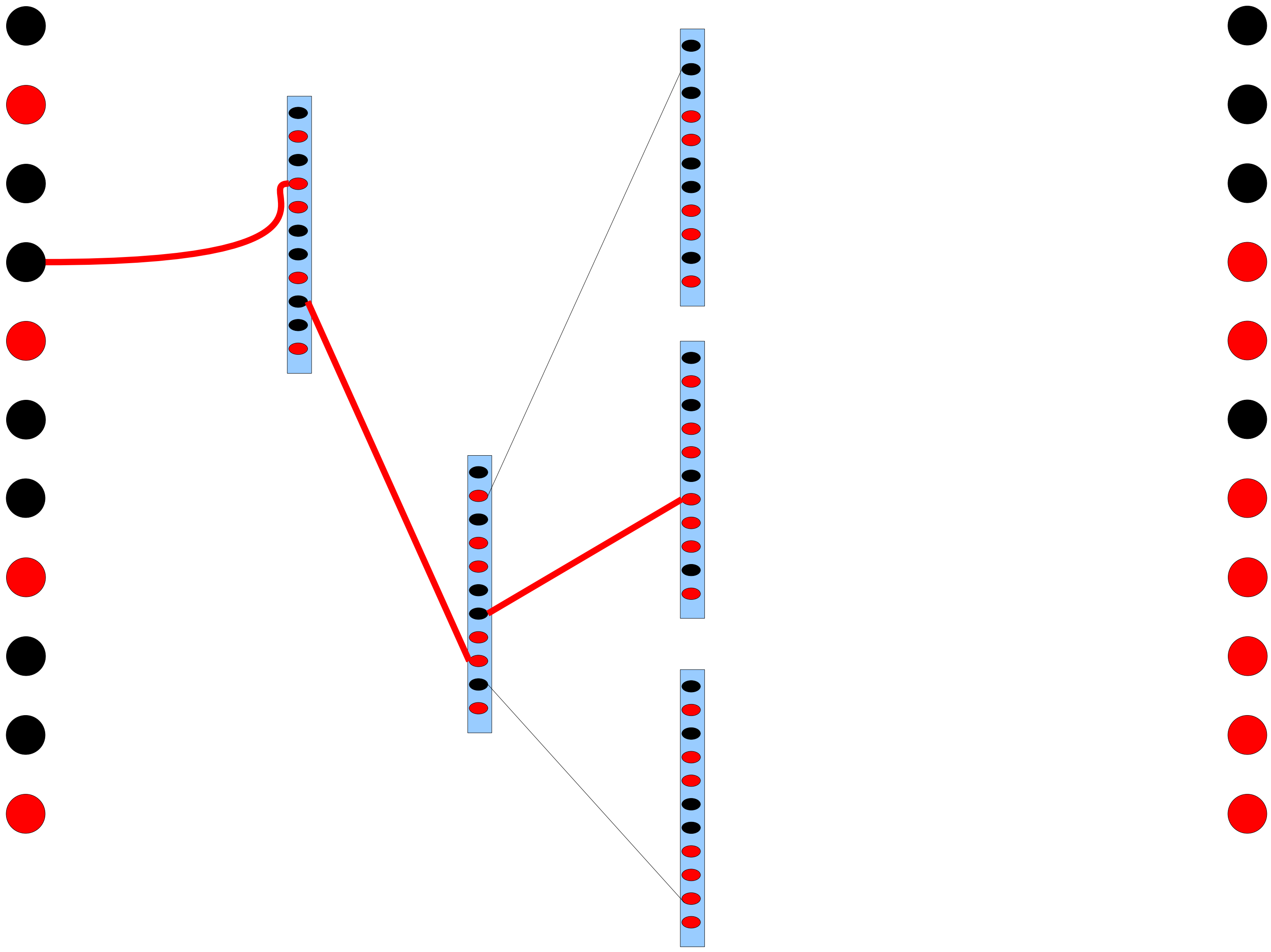
guiding solution



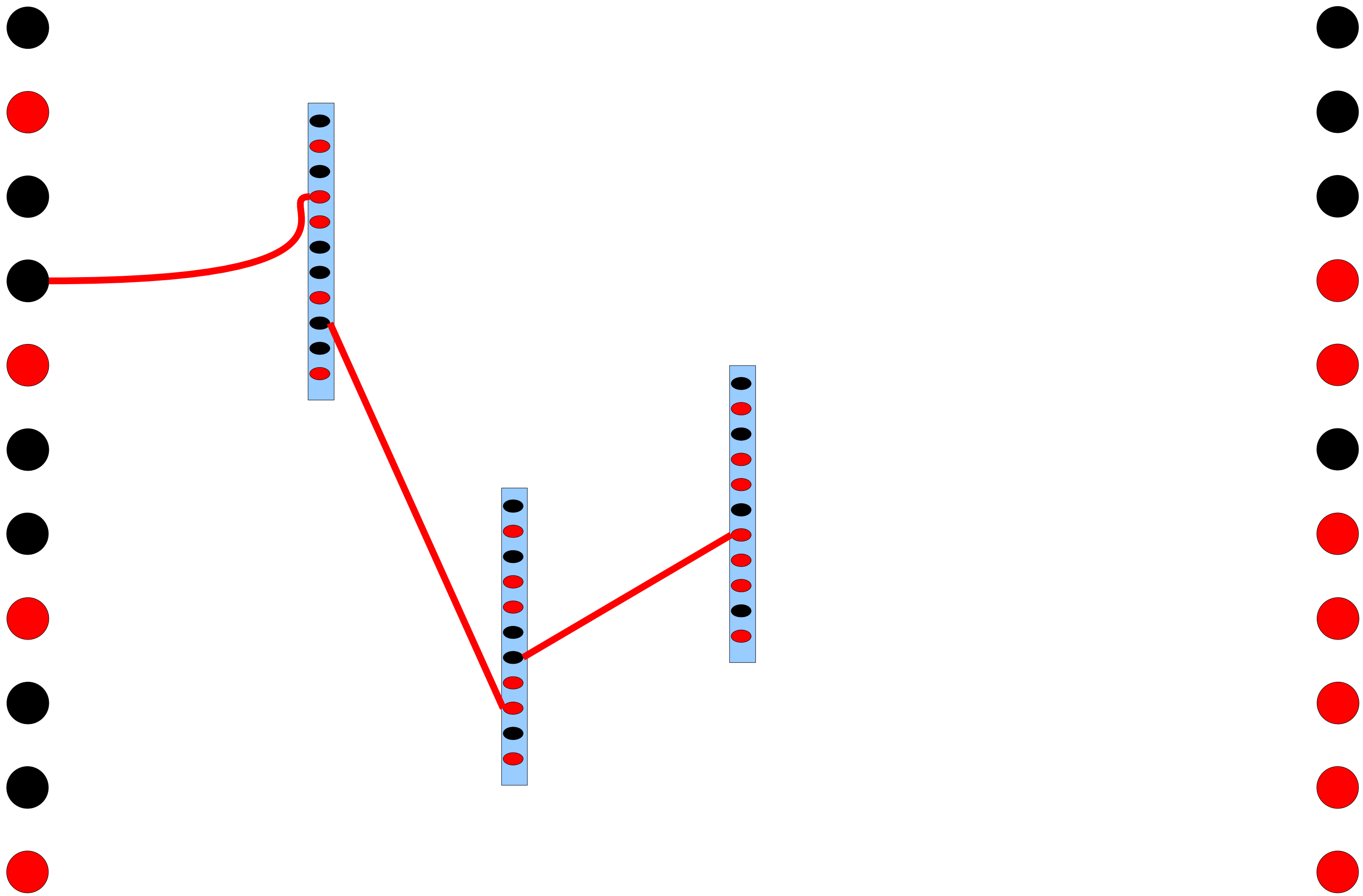
PR example



starting solution PR example guiding solution



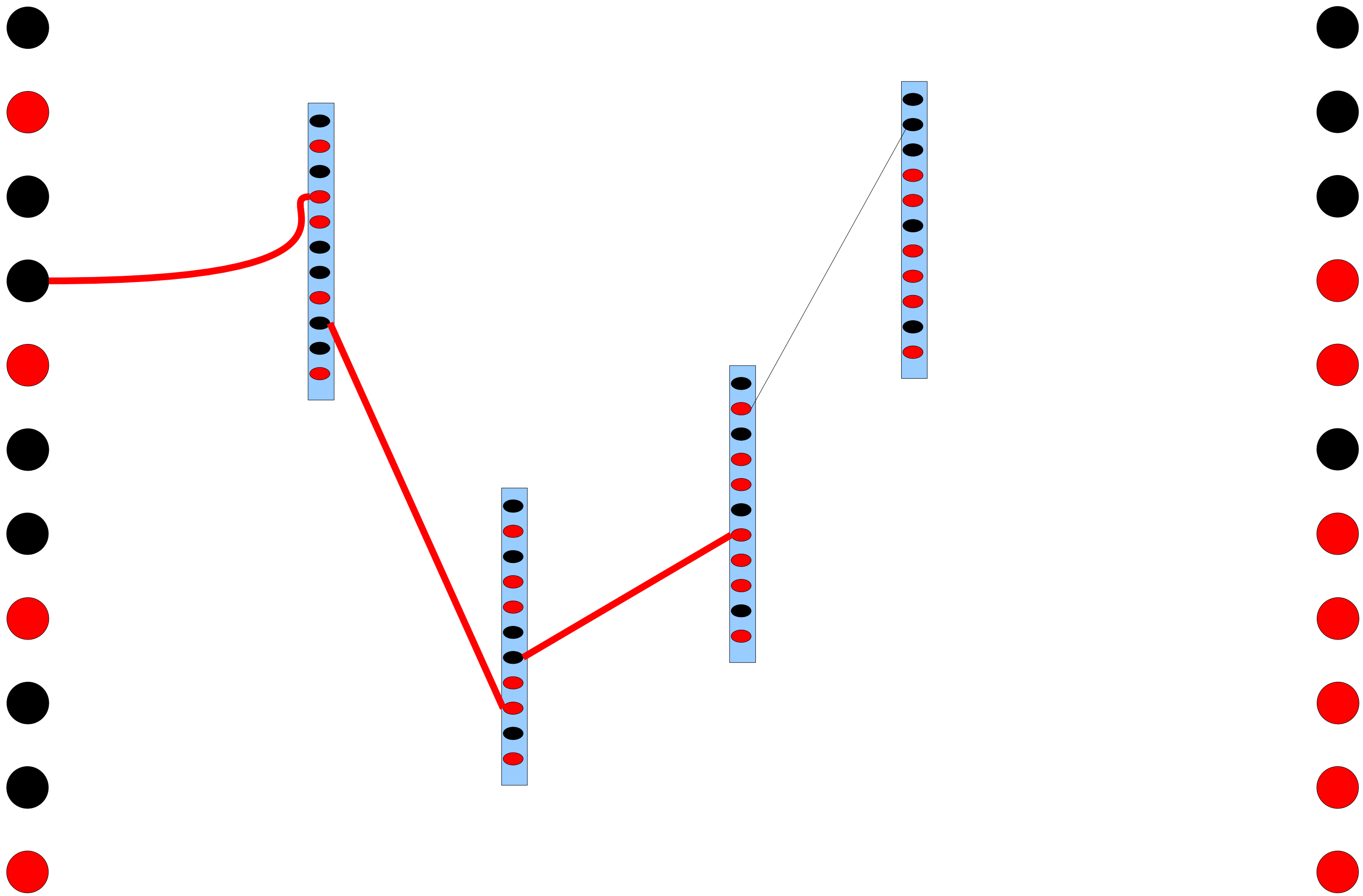
starting solution PR example guiding solution



PR example

starting solution

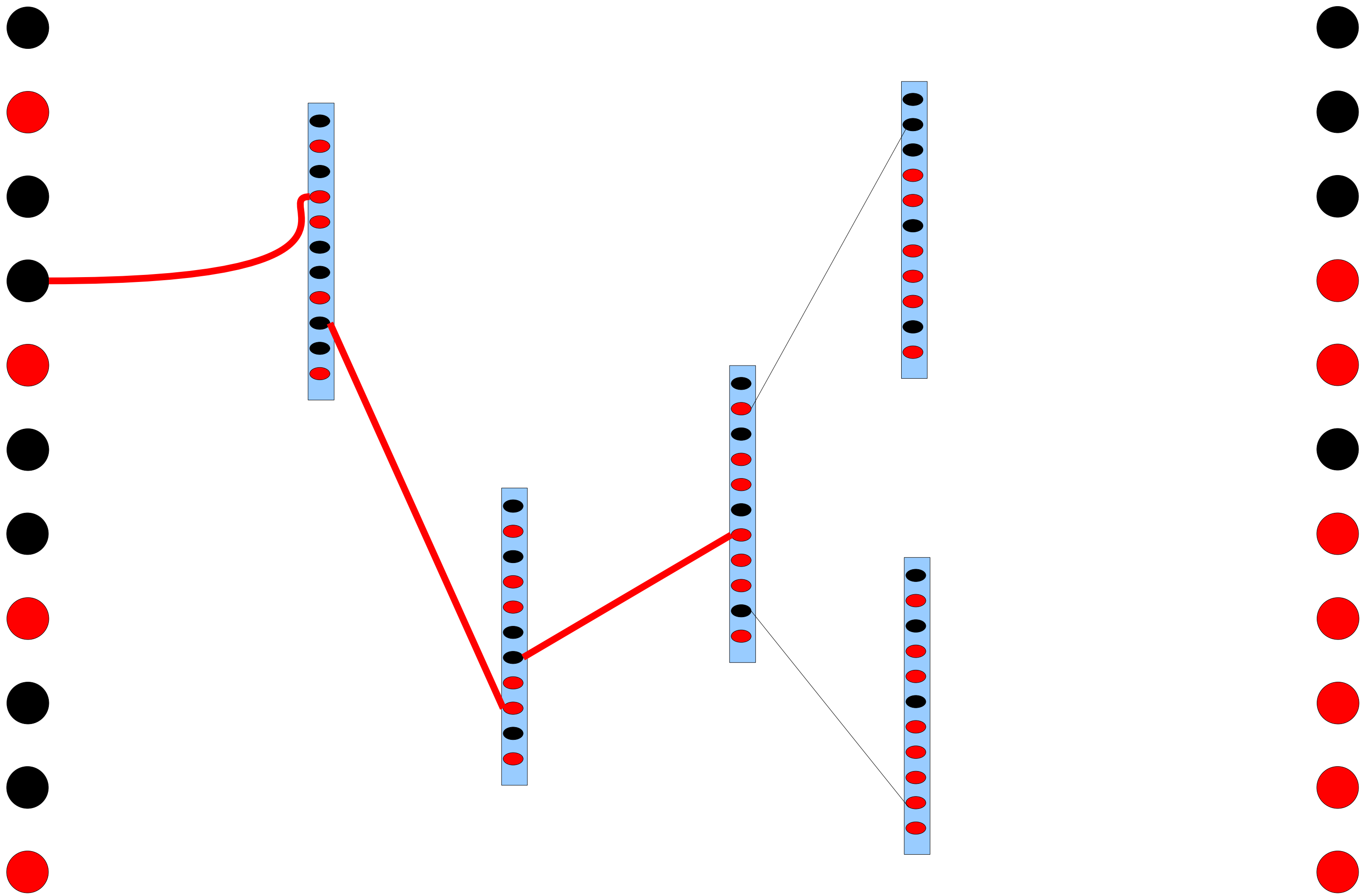
guiding solution



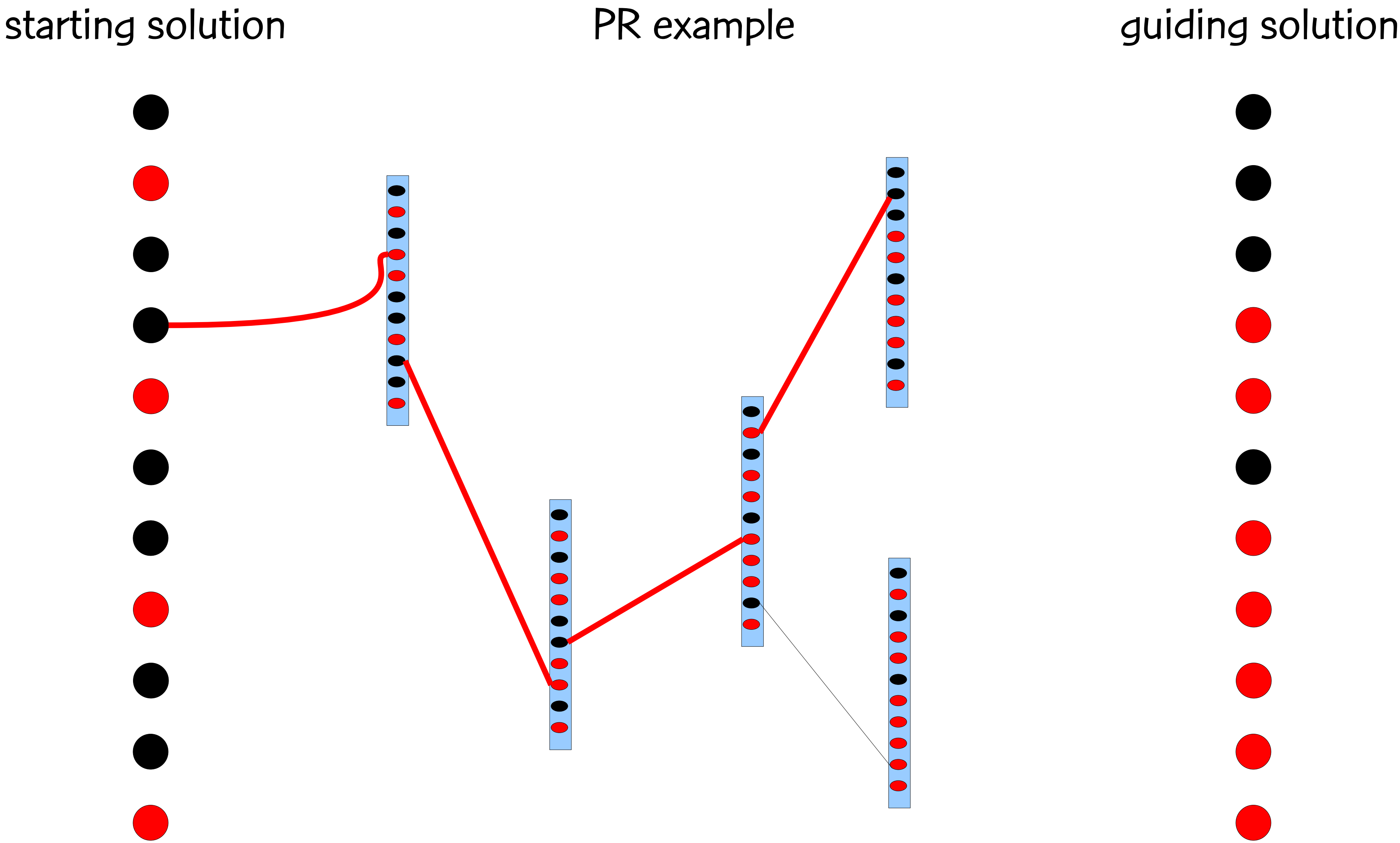
starting solution

PR example

guiding solution



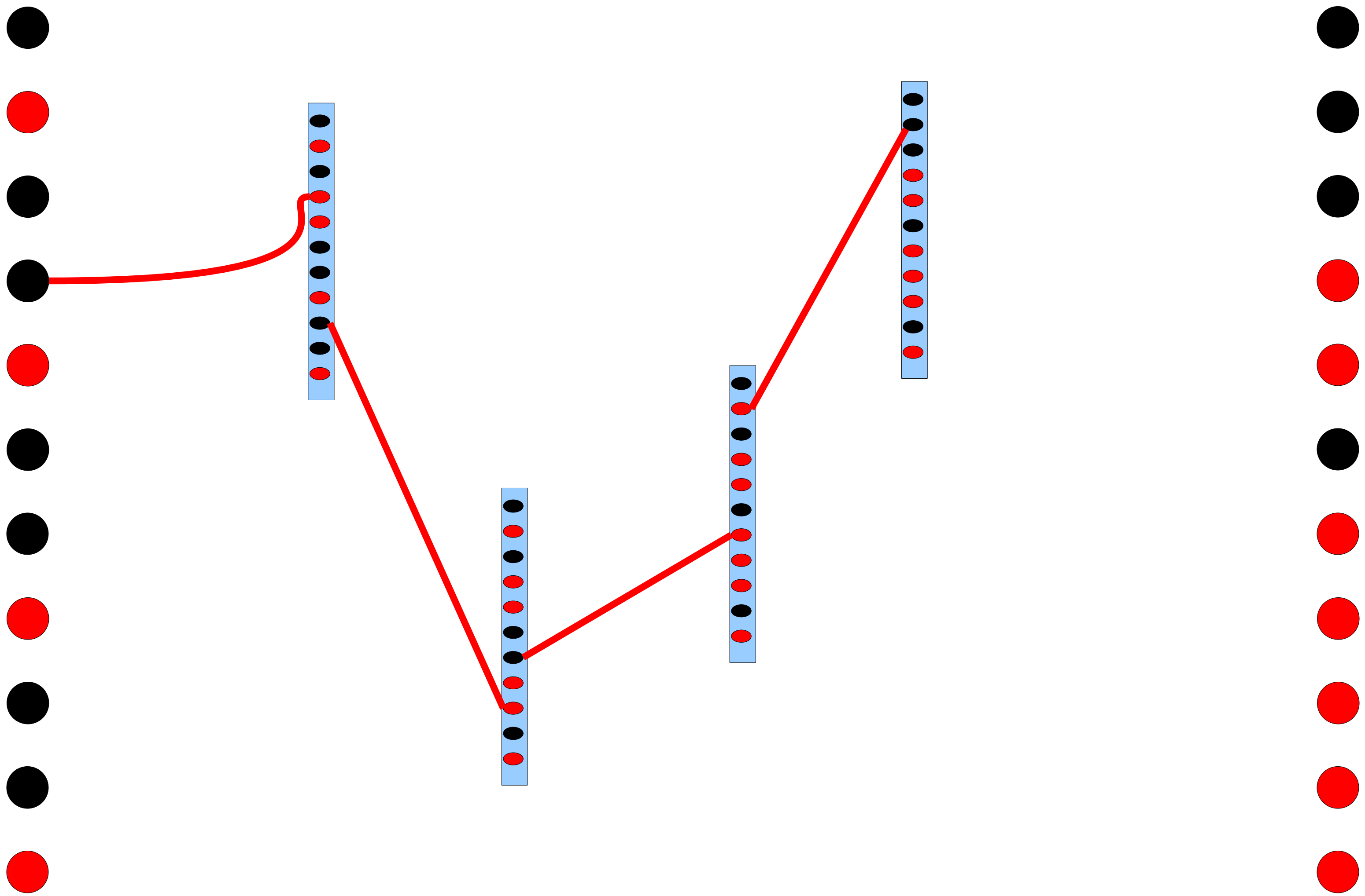
PR example



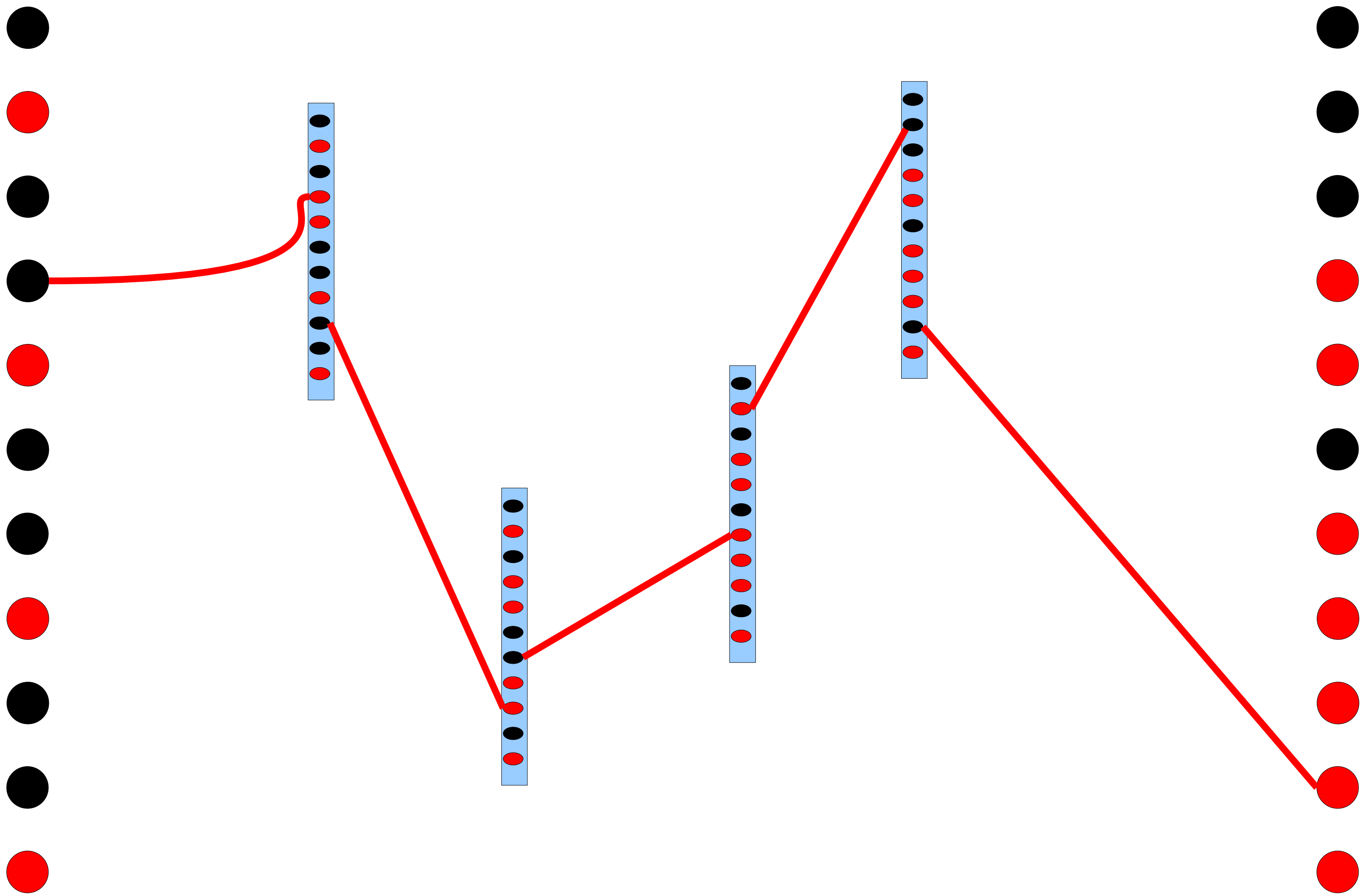
PR example

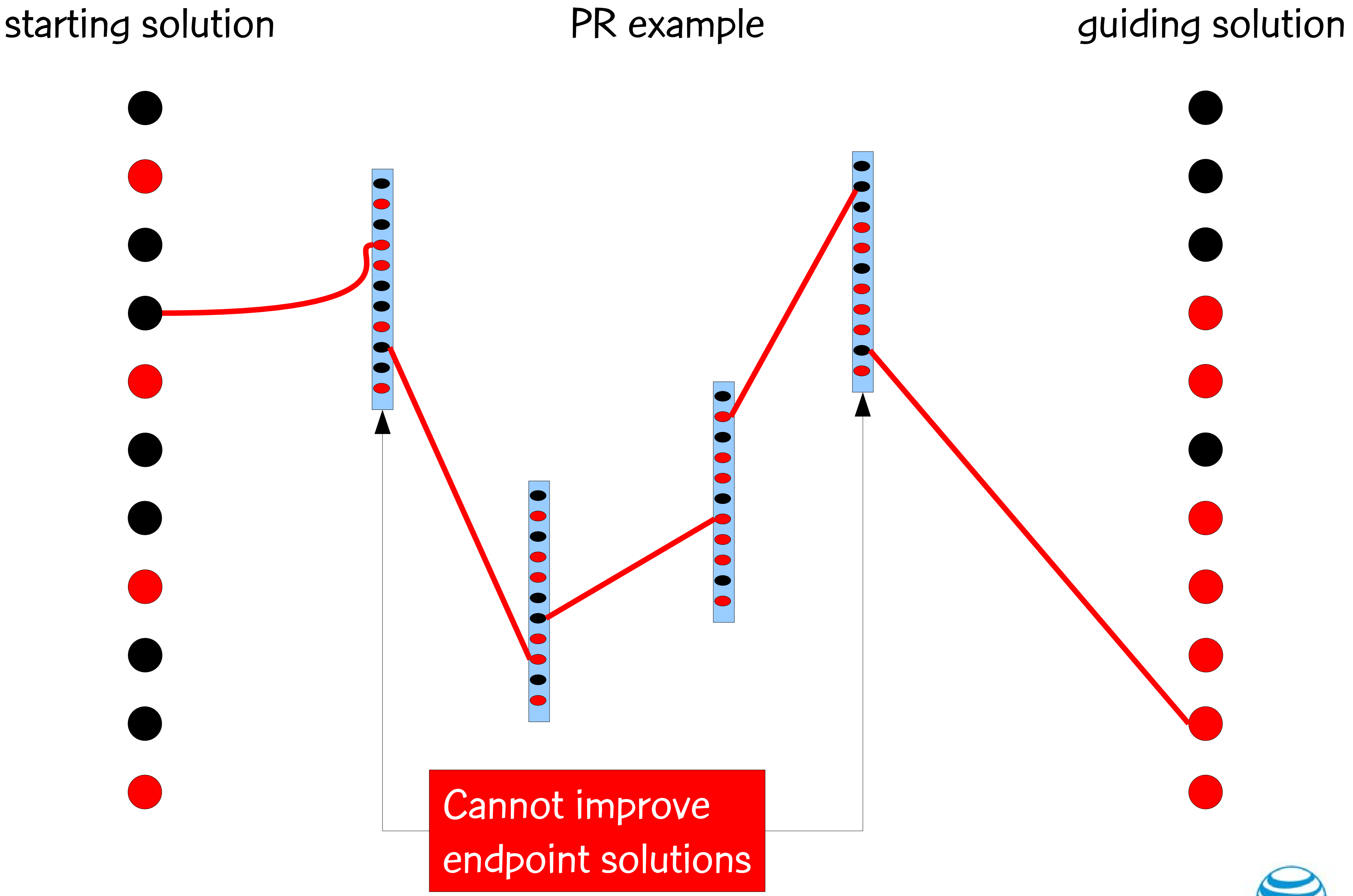
starting solution

guiding solution



starting solution — PR example — guiding solution

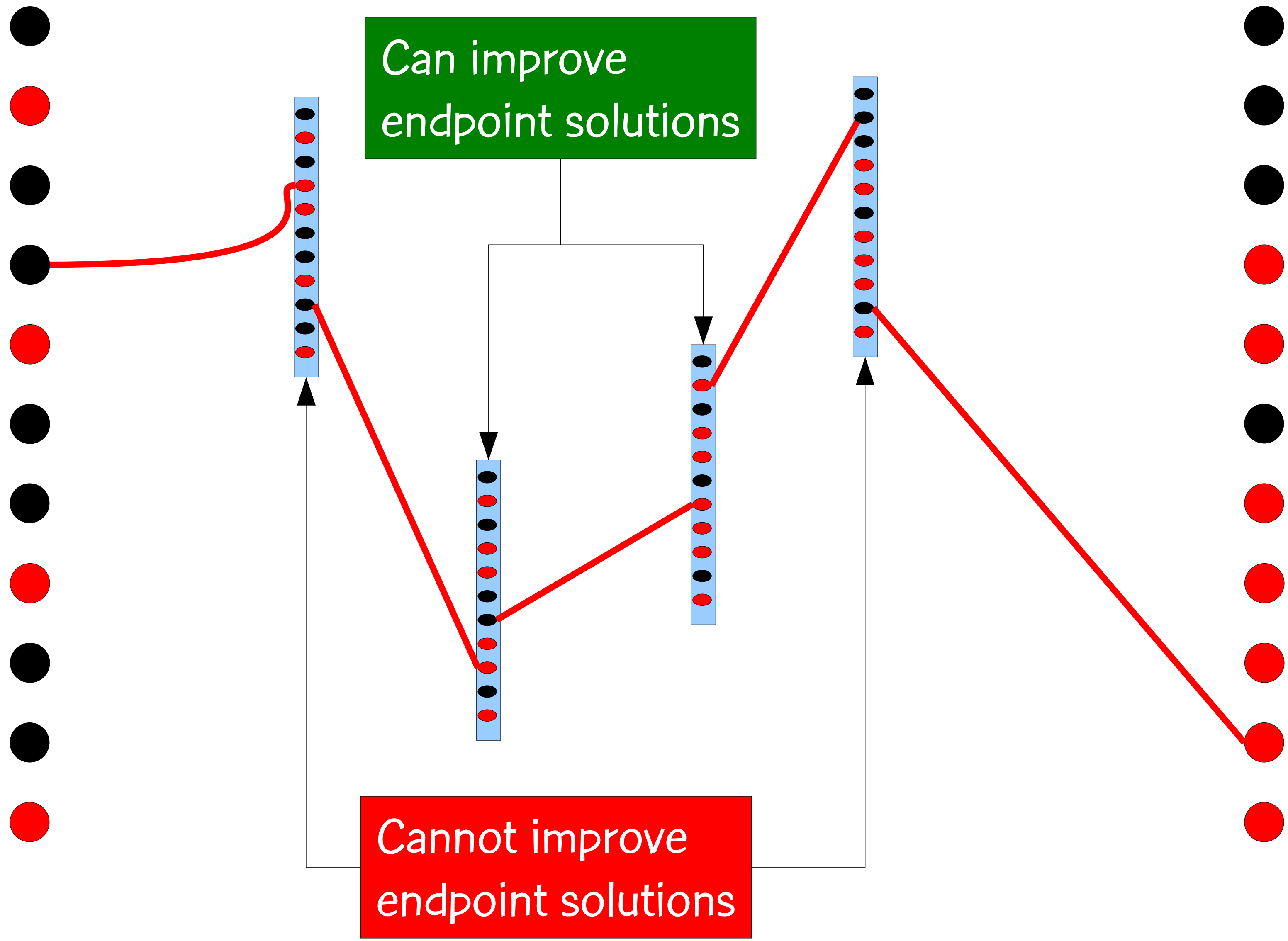



starting solution
PR example
guiding solution
Cannot improve
endpoint solutions

starting solution

PR example

guiding solution



GRASP with path-relinking

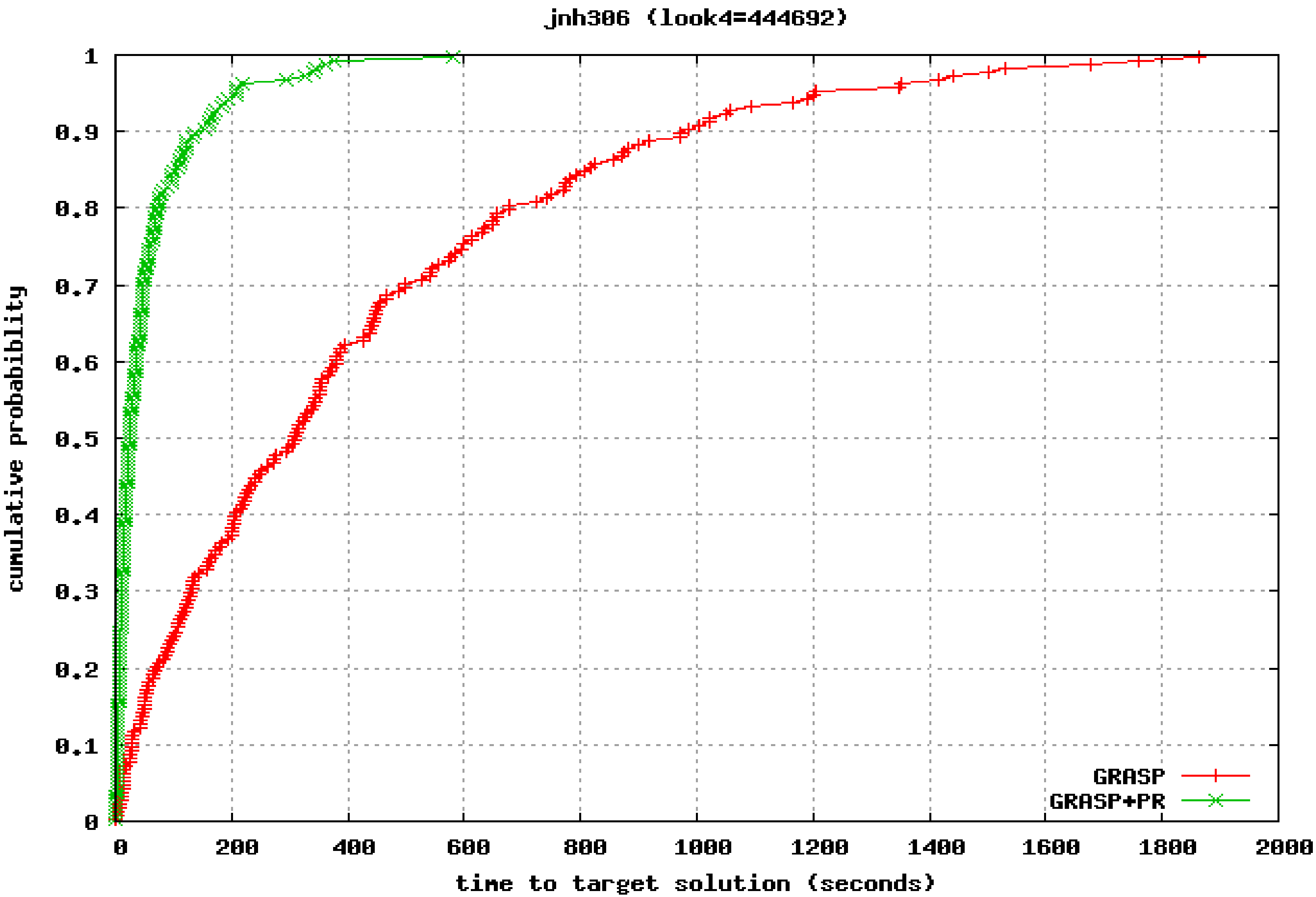
GRASP with path-relinking

- First proposed by Laguna and Martí (1999).
- Maintains a set of elite solutions found during GRASP iterations.
- After each GRASP iteration (construction and local search):
  - Use GRASP solution as initial solution.
  - Select an elite solution uniformly at random: guiding solution.
  - Perform path-relinking between these two solutions.

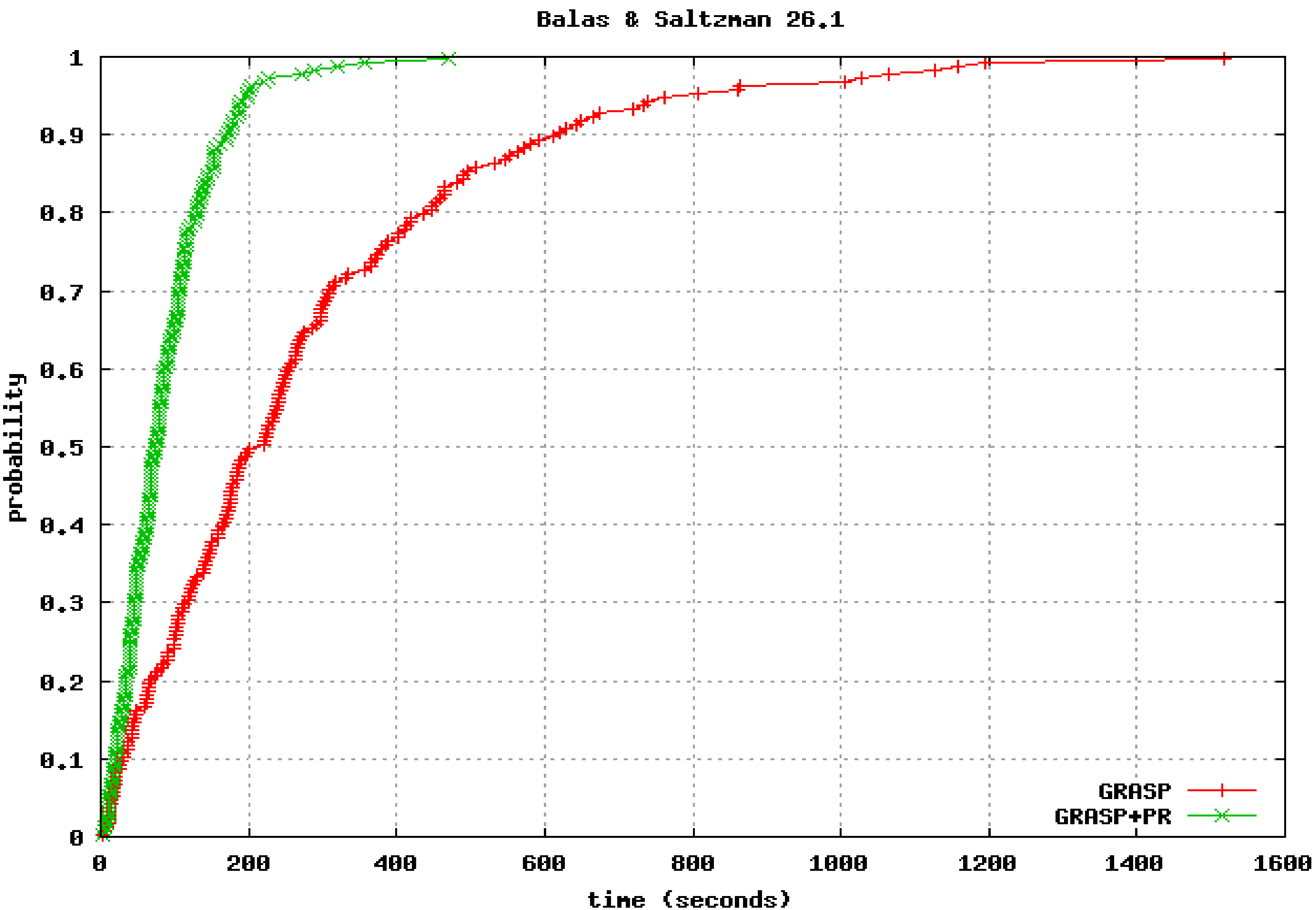
GRASP with path-relinking

- Since 1999, there has been a lot of activity in hybridizing GRASP with path-relinking.
- Survey by R. & Ribeiro in book of Ibaraki, Nonobe, and Yagiura (2005).
- Main observation from experimental studies: GRASP with path-relinking outperforms pure GRASP.

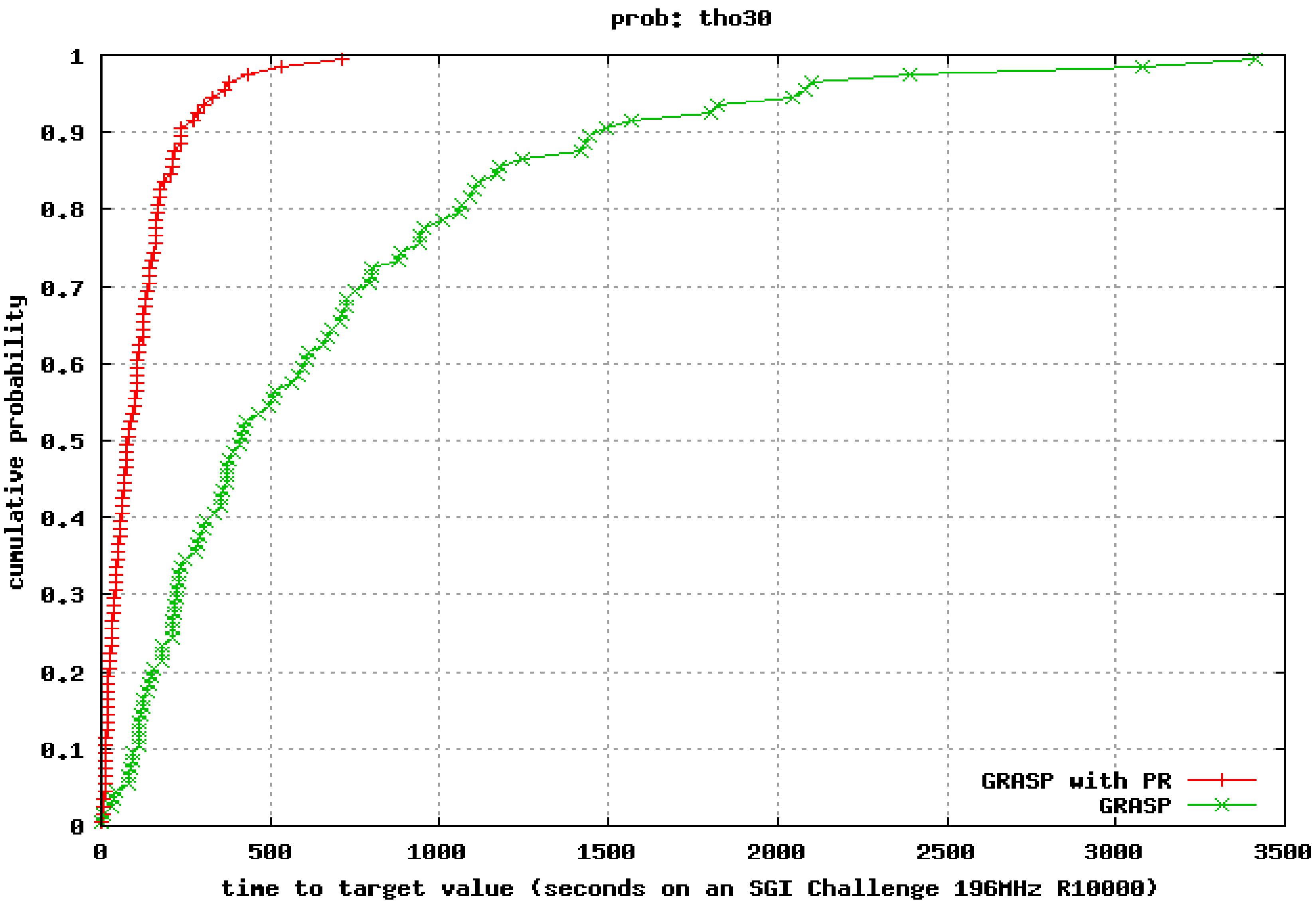
MAX-SAT (Festa, Pardalos, Pitsoulis, and Resende, 2006)



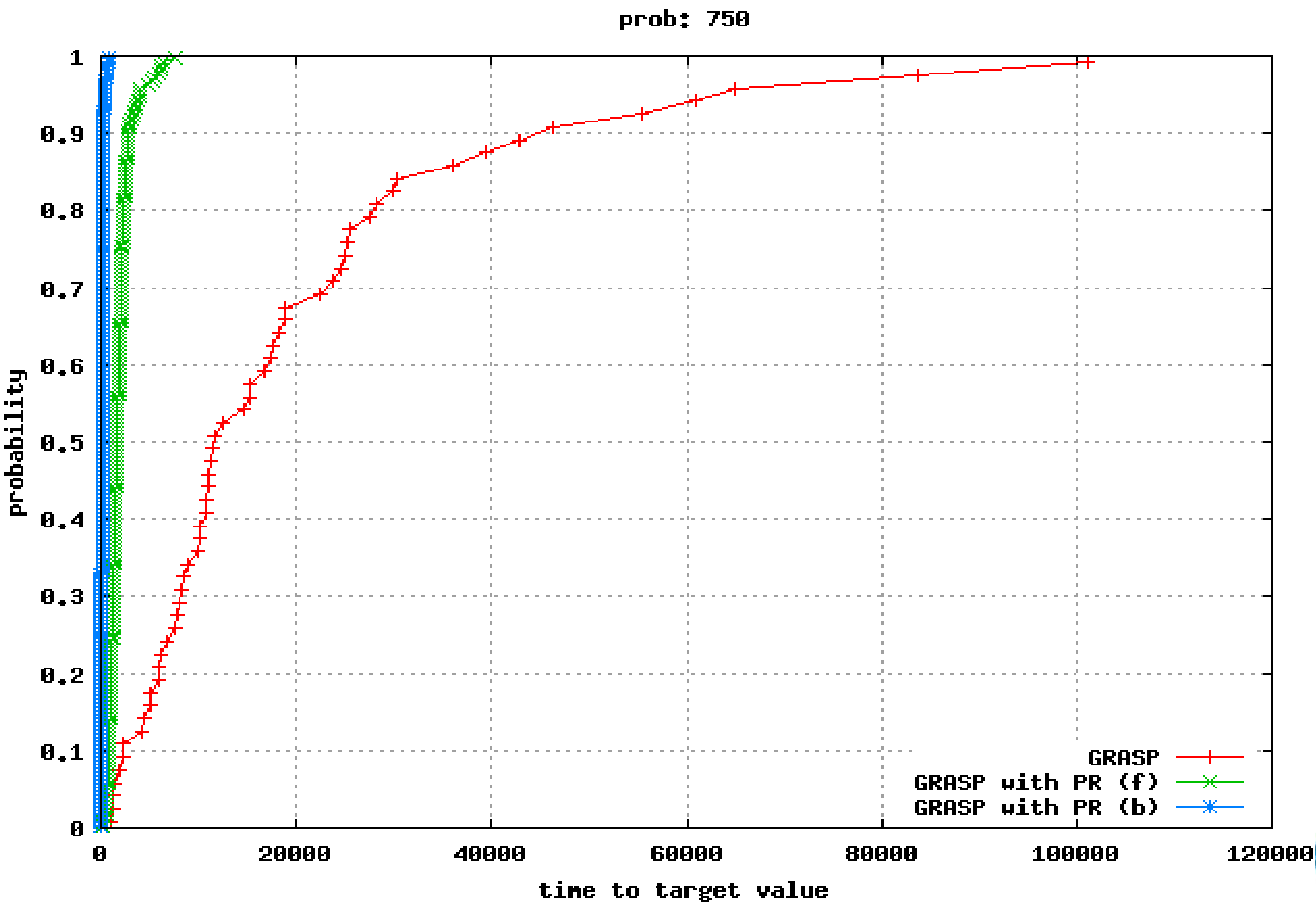
3-index assignment (Aiex, Resende, Pardalos, & Toraldo, 2005)



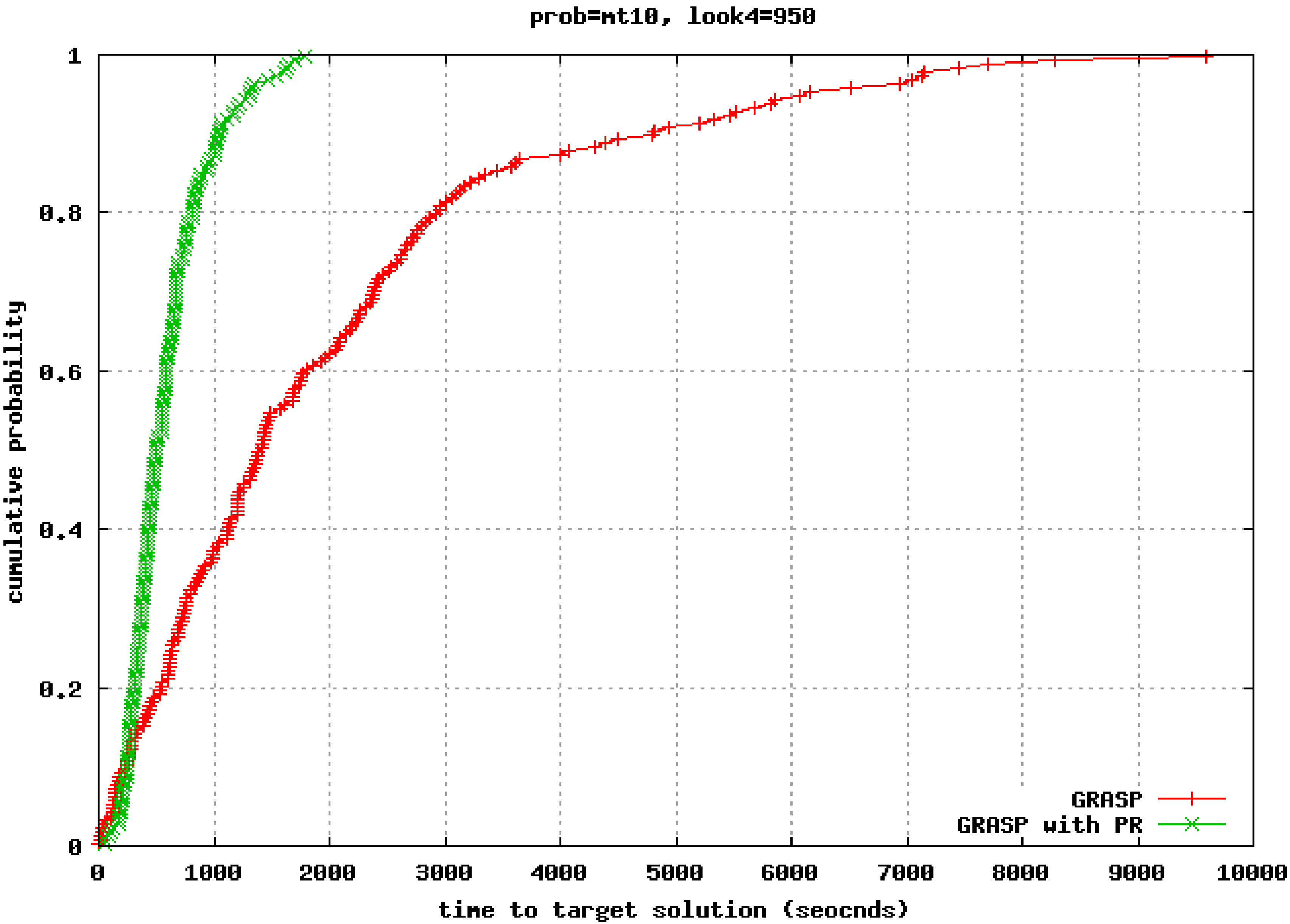
QAP (Oliveira, Pardalos, and Resende, 2004)



Bandwidth packing (Resende and Ribeiro, 2003)



Job shop scheduling (Aiex, Binato, & Resende, 2003)



GRASP with path-relinking

Repeat GRASP with PR loop	1) Construct randomized greedy X 2) Y = local search to improve X 3) Path-relinking between Y and pool solution Z 4) Update pool

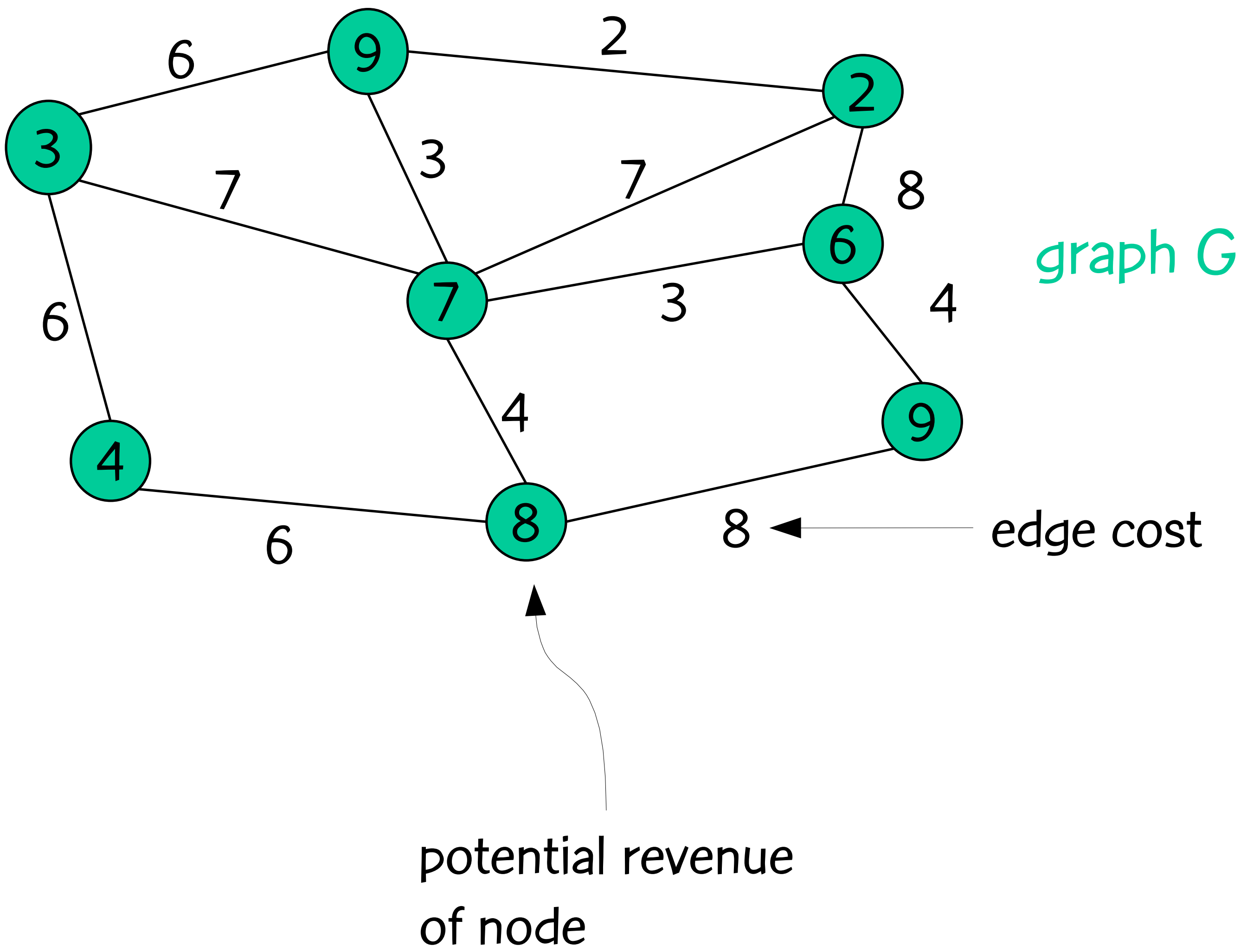
Network design to maximize difference between revenue and network cost:

Prize collecting Steiner problem in graphs

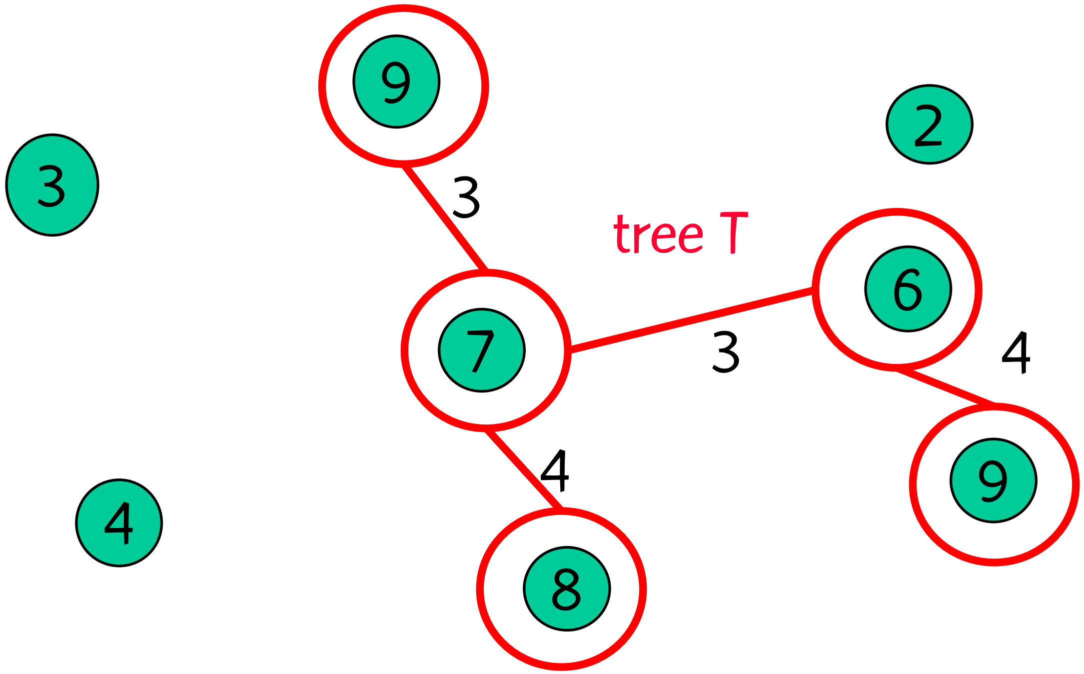
Prize-collecting Steiner tree (PCST) problem

- Given: graph $G = (V, E)$
  - Real-valued cost c_e is associated with edge e
  - Real-valued penalty d_v is associated with vertex v
- A tree is a connected acyclic subgraph of G and its weight is the sum of its edge costs plus the sum of the penalties of the vertices of G not spanned by the tree.
- PCST problem: Find tree of smallest weight.

Input: edge costs, node revenues

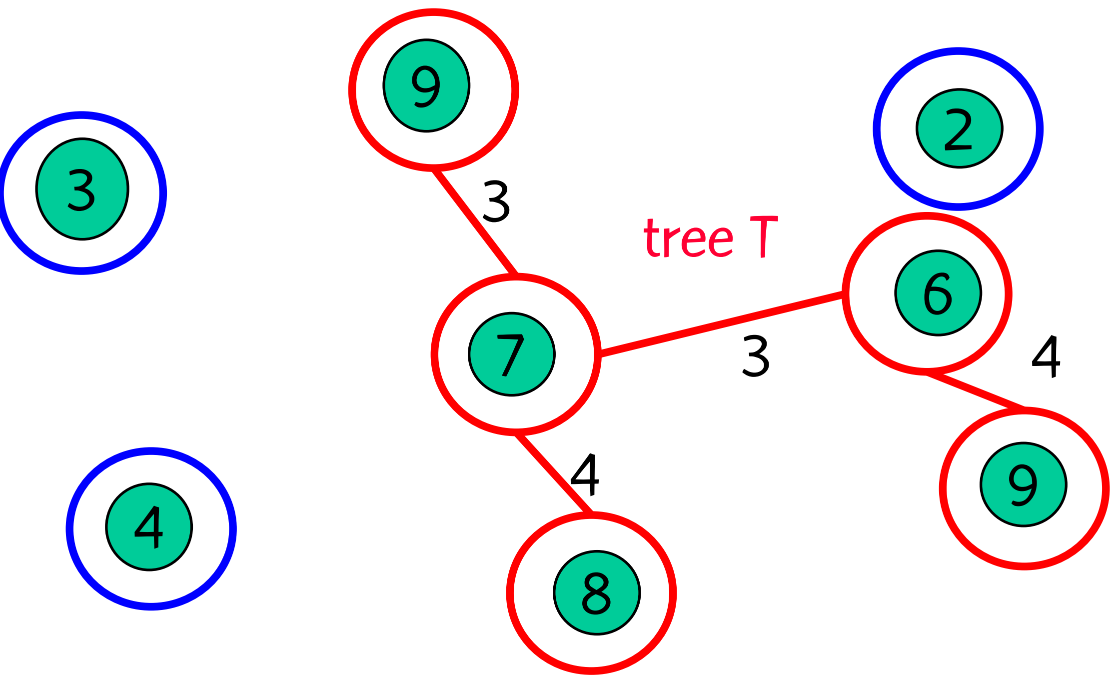


Cost of tree: tree edge cost plus revenue of unreached nodes



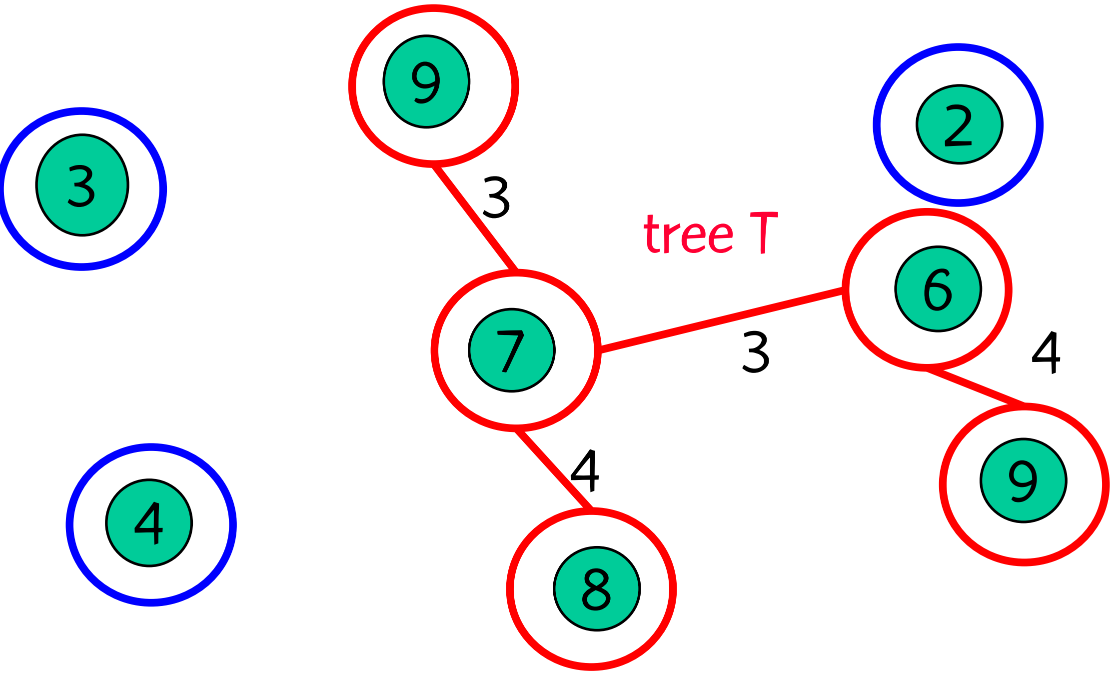
Cost (T) = Cost (edges of T) +

Cost of tree: tree edge cost plus revenue of unreached nodes



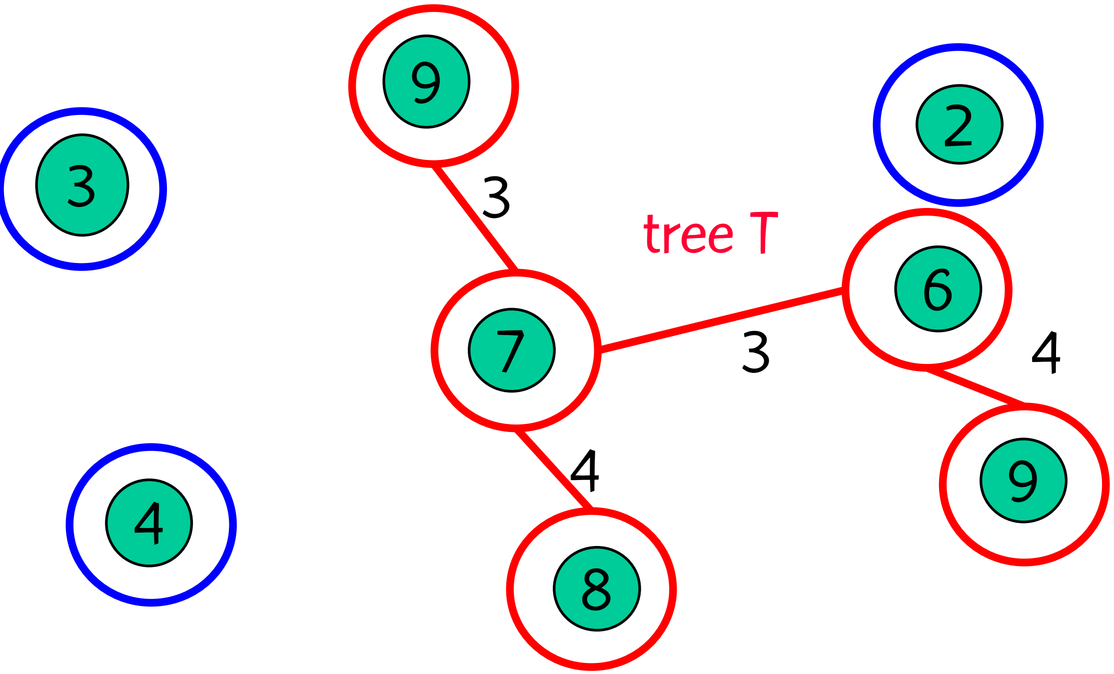
Cost (T) = Cost (edges of T) +
Revenue (nodes not reached by T)

Cost of tree: tree edge cost plus revenue of unreached nodes



Cost (T) = (3 + 3 + 4 + 4) +
Revenue (nodes not reached by T)

Cost of tree: tree edge cost plus revenue of unreached nodes



$$\text{Cost}(T) = (3+3+4+4) + (3+4+2) = 23$$

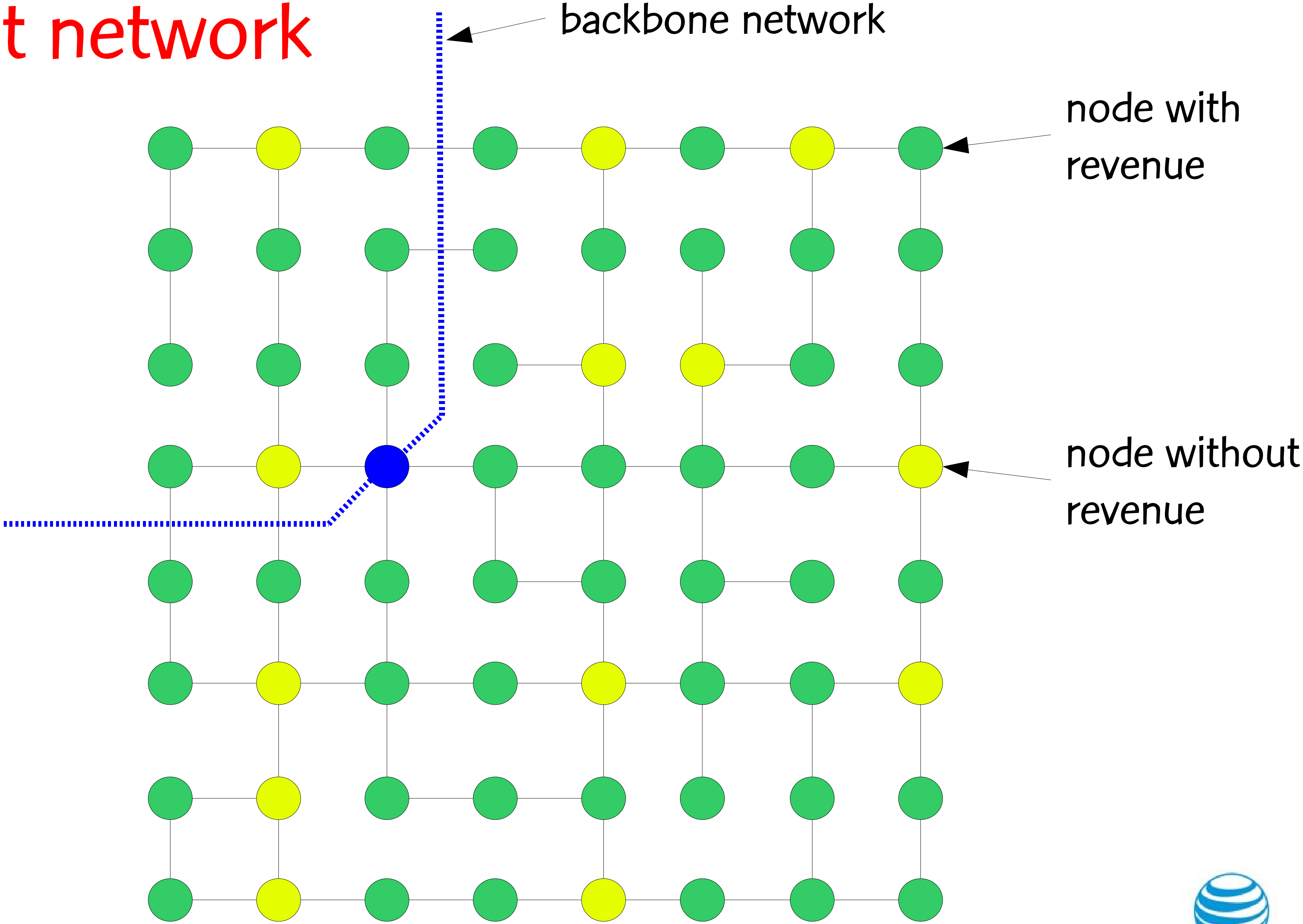
Design of local access telecommunications network

- Build a fiber-optic network for providing broadband connections to business and residential customers.
- Design a local access network taking into account trade-off between:
  - cost of network
  - revenue potential of network

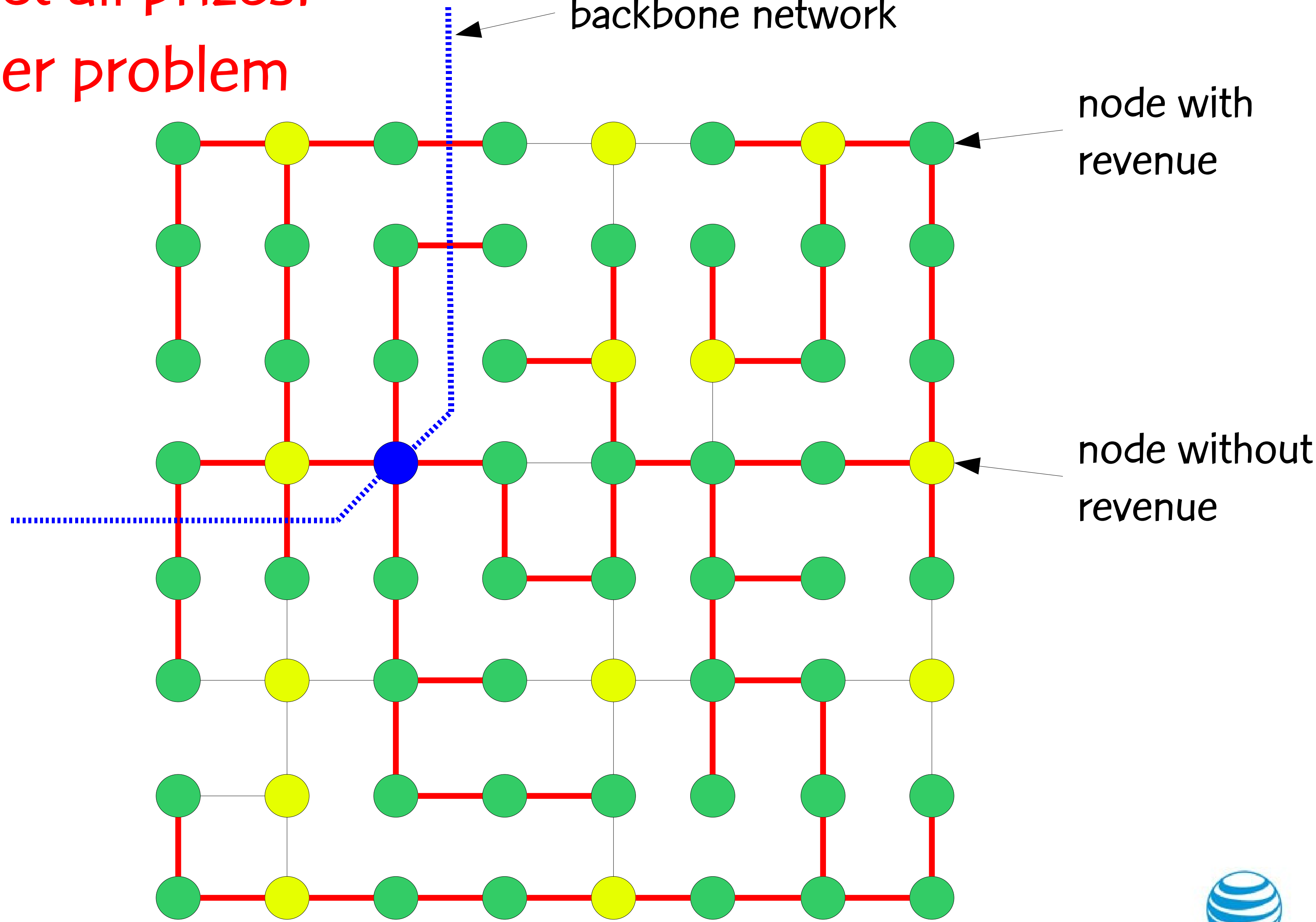
Design of local access telecommunications network

- Graph corresponds to local street map
  - Edges: street segments
    - Edge cost: cost of laying the fiber on the corresponding street segment
  - Vertices: street intersections and potential customer premises
    - Vertex penalty: estimate of potential loss of revenue if the customer were not to be serviced (intersection nodes have no penalty)

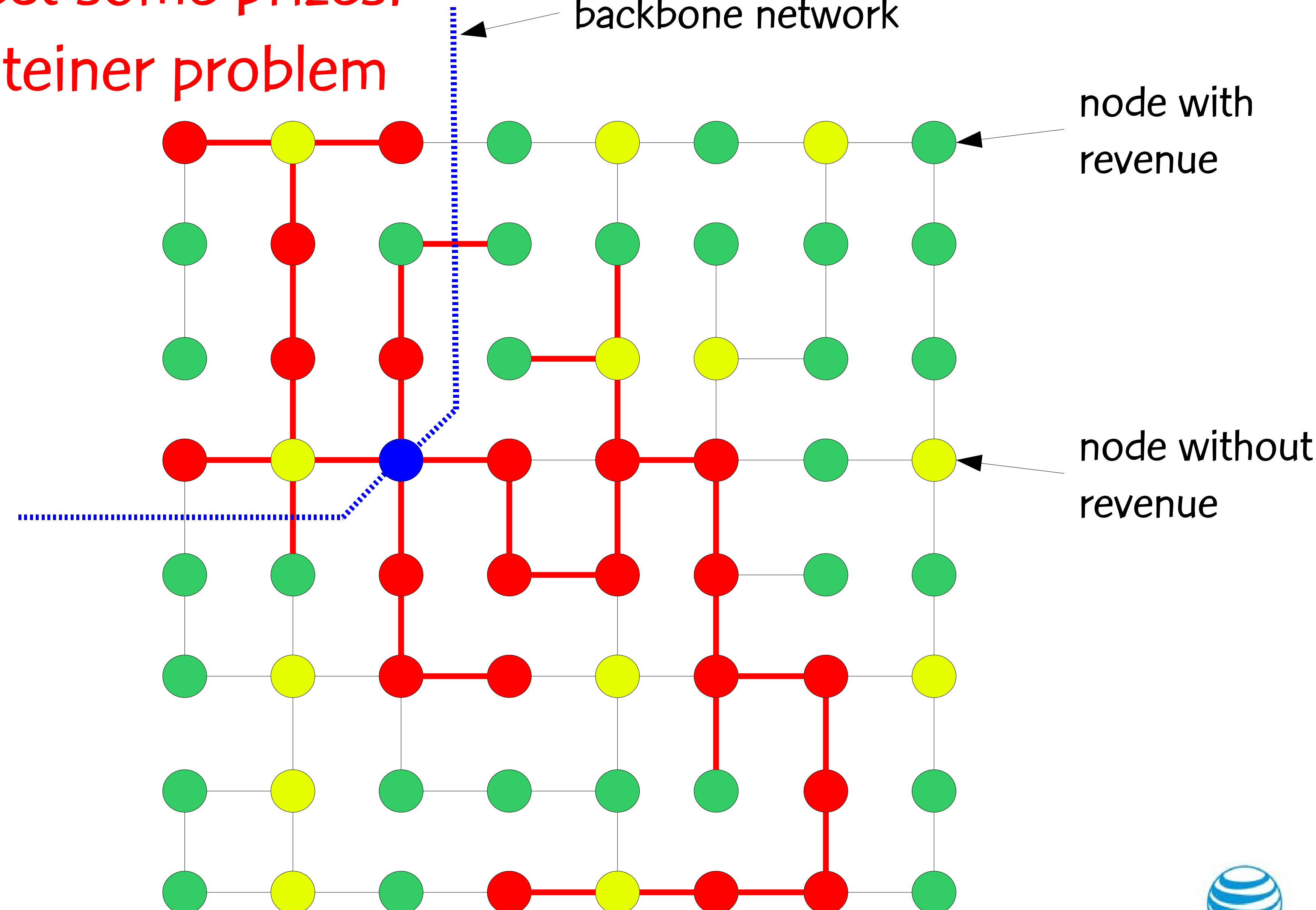
Input network



Collect all prizes: Steiner problem



Collect some prizes: PC Steiner problem



Literature

- Introduced by Bienstock, Goemans, Simchi-Levi, & Williamson (1993)
- Goemans & Williamson (1993, 1996): 5/2 and 2-opt approximation algorithms
- Johnson, Minkoff, & Phillips (1999): an implementation of the 2-opt algorithm of Goemans & Williamson (GW)
- Canuto, R., & Ribeiro (2001): GRASP heuristic that uses a randomized version of GW
- Lucena & R. (2004): polyhedral cutting plane algorithm for computing lower bounds
- Klau et al. (2004): memetic algorithm
- Uchoa (2006): reduction tests
- Ljubic et al. (2006): exact solution via branch and cut algorithm
- Andrade, Lucena, Maculan, and R. (2008): Relax and cut algorithm

Literature

- Introduced by Bienstock, Goemans, Simchi-Levi, & Williamson (1993)
- Goemans & Williamson (1993, 1996): 5/2 and 2-opt approximation algorithms
- Johnson, Minkoff, & Phillips (1999): an implementation of the 2-opt algorithm of Goemans & Williamson (GW)
- Canuto, R., & Ribeiro (2001): GRASP heuristic that uses a randomized version of GW
- Lucena & R. (2004): polyhedral cutting plane algorithm for computing lower bounds
- Klau et al. (2004): memetic algorithm
- Uchoa (2006): reduction tests
- Ljubic et al. (2006): exact solution via branch and cut algorithm
- Andrade, Lucena, Maculan, and R. (2008): Relax and cut algorithm

Solution construction

- Select X, the set of collected nodes
- Connect node in X with minimum weight spanning tree $T(X)$
- Recursively remove from $T(X)$ all degree-1 nodes with prize smaller than its incident edge cost = $T_r(X)$
- Basic strategy:

for ($i = 1$ to MAXITR){

select X_i

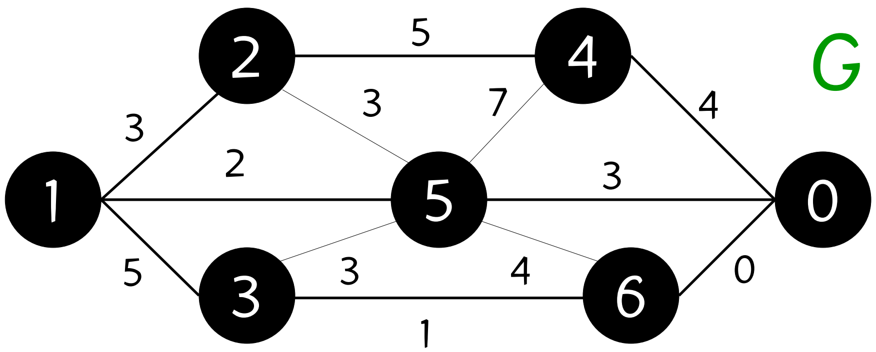
compute $T(X_i)$ and $T_r(X_i)$

}

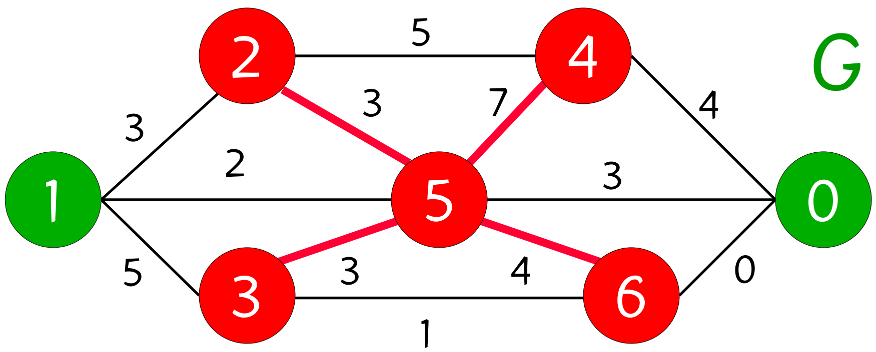
Goemans & Williamson 2-opt algorithm

Kruskal's algorithm

Solution construction



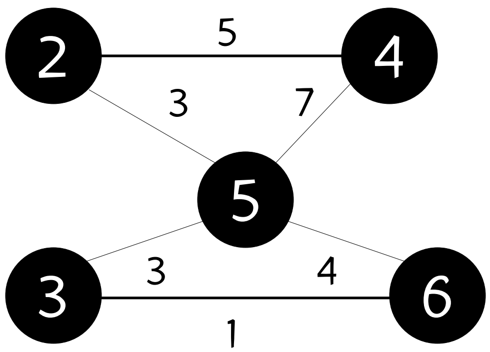
Solution construction



Solution obtained by GW: $X = \{2,3,4,5,6\}$

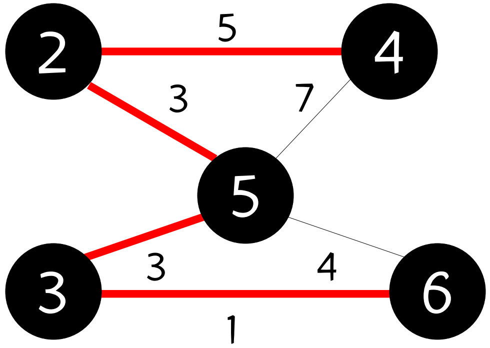
Cost = 18

Solution construction



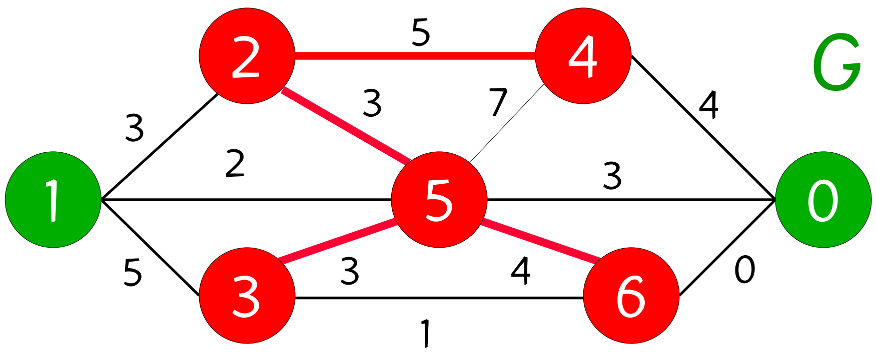
G' = subgraph induced on G by nodes in X

Solution construction



MST on *G'*

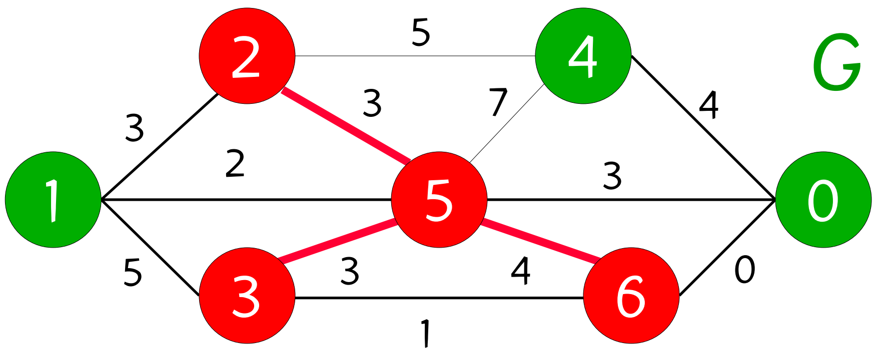
Solution construction



Solution derived from MST on G'

Cost = 13

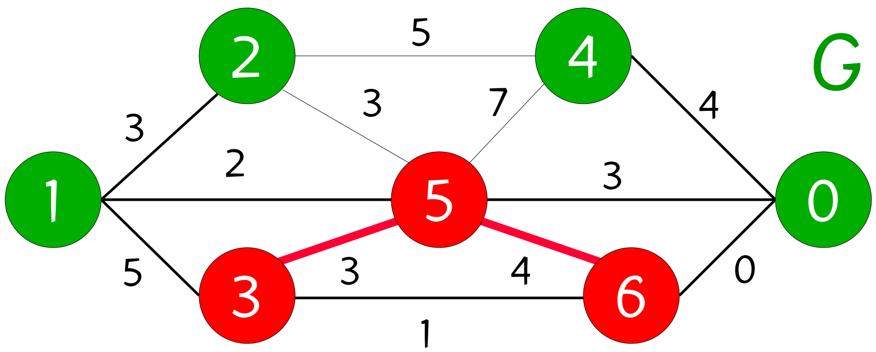
Solution construction



Solution obtained by pruning degree-1 node

Cost = 12

Solution construction



Final solution obtained by pruning another degree-1 node

Cost = 11

Local search

- Representation of solution: set X of vertices in tree $T(X)$
- Neighborhood:
  - $N(X) = \{X' : X$ and X' differ by single node$\}$
  - Moves: insertion & deletion of nodes
- Initial solution: nodes of tree obtained by GW
- Iterative improvement: make move as long as improvement is possible

Local search: input set X and cost(X)

```
improve = T
while ( improve){
    improve = F
    circfor i = 1, ..., |V| while .not. improve
    {       if (i ∈ X ){ X' = X \ {i}}
            else {X' = X ∪ {i}}
            compute tree T(X') & cost(X')
            if (cost(X') < cost(X )){
                    X = X'
                    improve = T
            }
    }
}
```

Multi-start strategy

- Force GW to construct different initial solutions for local search
  - Use original prizes in first iteration
  - Use modified prizes after that
- Modify prizes (two strategies)
  - Introduce noise into prizes

    for $i = 1, \ldots, |V|$ {

    generate $\beta \in [1 - a, 1 + a]$, for $a > 0$

    $d'(i) = d(i) \times \beta$

    }
  - Node elimination
    - Set to zero the prizes of α% of the nodes in nodes(GW) $\cap$ nodes(local search)

GRASP with perturbed costs

```
best = HUGE
d' = d
for ( i = 1, ..., MAXITR ){
    X = GW ( V, E, c, d' )
    X' = LOCALSEARCH(V, E, c, d, X )
    if ( cost(X') < best ){
        X* = X'
    }
    compute perturbations & update d'
}
return X*
```

Approximation algorithm is done on perturbed data.

Local search is done on original data.

Path relinking

- In local search with perturbations let
  - X' be the local optimum found by LOCALSEARCH
  - Y be a solution chosen randomly from a POOL of elite solutions
  - $\Delta = \{i \in V : (i \in X'$ and $i \notin Y)$ or

    $(i \notin X'$ and $i \in Y)\}$
- Construct path between X' (start) and Y (guide):
  - Apply best movement in Δ
  - Verify quality of solution after move
  - Update Δ

GRASP with perturbed costs & path relinking

```
POOL = ϕ
d' = d
for ( i = 1, ..., MAXITR ){
    X = GW ( V, E, c, d' )
    if ( X is new){
        X' = LOCALSEARCH( V, E, c, d, X )
        attempt insert X' into POOL
        X'' ∈ RAND(POOL)
        X_PR = PATHRELINK( X', X'' )
        attempt to insert X_PR into POOL
        }
    }
    compute perturbations & update d'
}
return best solution in POOL
```

Variable neighborhood search

- Can we gain something by going from a static neighborhood to one that is dynamic?
- Consider K neighborhoods:
  - $N^1, N^2, \ldots, N^K$
  - $N^k(X) = \{ X' : X$ and X' differ by k nodes$\}$
- Basic scheme (repeated MAXTRY times):
  - Start with initial solution X and $k = 1$
  - while ($k \leq K$){

    choose $X' \in N^k(X)$

    $X'' = \text{LOCALSEARCH}(V, E, c, d, X')$

    $k = k + 1$

    if cost(X'') < cost(X) { $X = X''$; $k = 1$ }

    }

GRASP with perturbed costs & path relinking & VNS

```
POOL = ϕ
d' = d
for ( i = 1, ..., MAXITR ){
    X = GW ( V, E, c, d' )
    if ( X is new){
        X' = LOCALSEARCH( V, E, c, d, X )
        attempt insert X' into POOL
        X'' ∈ RAND(POOL)
        X_PR = PATHRELINK( X', X'' )
        attemp to insert X_PR into POOL
        }
    }
    compute perturbations & update d'
}
X* = best solution in POOL
X* = VNS( V, E, c, d, X* )
return X*
```

Computational results

- 114 test problems
  - From 100 nodes & 284 edges
  - To 1000 nodes & 25,000 edges
  - Three classes:
    - Johnson, Minkoff, & Phillips (1999) P & K problems
    - Steiner C problems (derived from SPG Steiner C test problems in OR-Library)
    - Steiner D problems (derived from SPG Steiner D test problems in OR-Library)

Computational results

- Heuristic found
  - 89 of 104 known optimal values (86%)
  - solution within 1% of lower bound for 104 of 114 problems

Number of optima found with each additional heuristic

type	num	GW	+LS	+PR	+VNS	tot
C	38	6	2	25	3	36
D	32	5	6	10	4	25
JMP	34	8	6	12	2	28
	104					89

Genetic algorithms with random keys

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Individuals are strings of real-valued numbers (random keys) in the interval [0,1].

$$S = (\ 0.25,\ 0.19,\ 0.67,\ 0.05,\ 0.89\ )$$

$$\quad s(1) \quad s(2) \quad s(3) \quad s(4) \quad s(5)$$

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Individuals are strings of real-valued numbers (random keys) in the interval [0,1].
- Sorting random keys results in a sequencing order.

$$S = (\ 0.25,\ 0.19,\ 0.67,\ 0.05,\ 0.89\ )$$
$$\quad s(1)\quad s(2)\quad s(3)\quad s(4)\quad s(5)$$

$$S' = (\ 0.05,\ 0.19,\ 0.25,\ 0.67,\ 0.89\ )$$
$$\quad s(4)\quad s(2)\quad s(1)\quad s(3)\quad s(5)$$

Sequence: 4 – 2 – 1 – 3 – 5

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)

a = (0.25, 0.19, 0.67, 0.05, 0.89)

b = (0.63, 0.90, 0.76, 0.93, 0.08)

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

$a = (0.25, 0.19, 0.67, 0.05, 0.89)$

$b = (0.63, 0.90, 0.76, 0.93, 0.08)$

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

a = (0.25, 0.19, 0.67, 0.05, 0.89)
b = (0.63, 0.90, 0.76, 0.93, 0.08)
c = ( )

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

a = (0.25, 0.19, 0.67, 0.05, 0.89)
b = (0.63, 0.90, 0.76, 0.93, 0.08)
c = (0.25)

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

a = (0.25, 0.19, 0.67, 0.05, 0.89)
b = (0.63, 0.90, 0.76, 0.93, 0.08)
c = (0.25, 0.90)

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

a = (0.25, 0.19, 0.67, 0.05, 0.89)
b = (0.63, 0.90, 0.76, 0.93, 0.08)
c = (0.25, 0.90, 0.76)

GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

a = (0.25, 0.19, 0.67, 0.05, 0.89)
b = (0.63, 0.90, 0.76, 0.93, 0.08)
c = (0.25, 0.90, 0.76, 0.05)

GAs and random keys

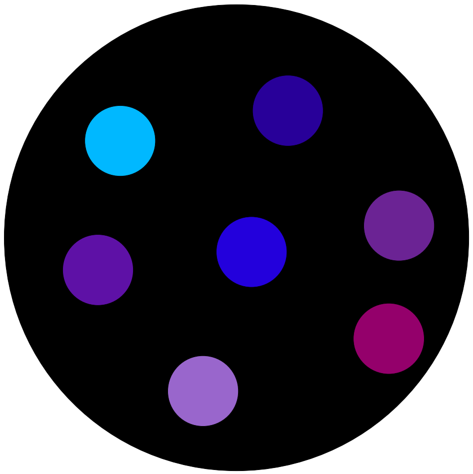
- Introduced by Bean (1994) for sequencing problems.
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passed the allele to the child.

a = (0.25, 0.19, 0.67, 0.05, 0.89)
b = (0.63, 0.90, 0.76, 0.93, 0.08)
c = (0.25, 0.90, 0.76, 0.05, 0.89)

Every random-key array corresponds to a feasible solution: Mating always produces feasible offspring.

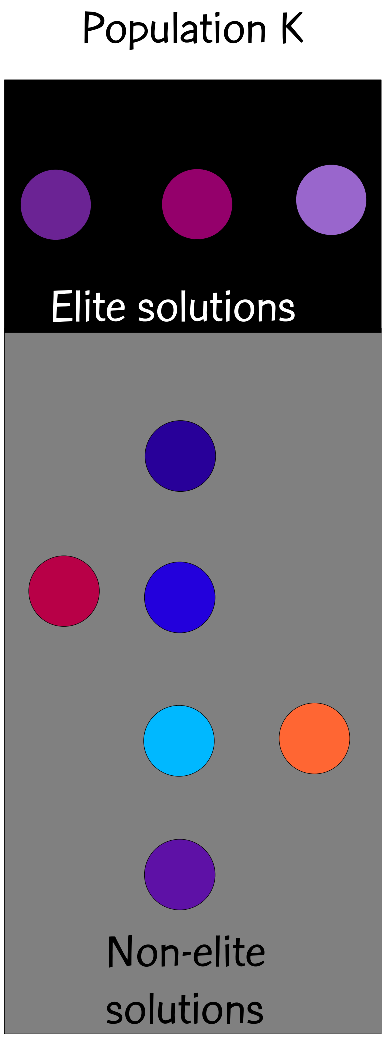
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Initial population is made up of P chromosomes, each with N genes, each having a value (allele) generated uniformly at random in the interval [0,1].



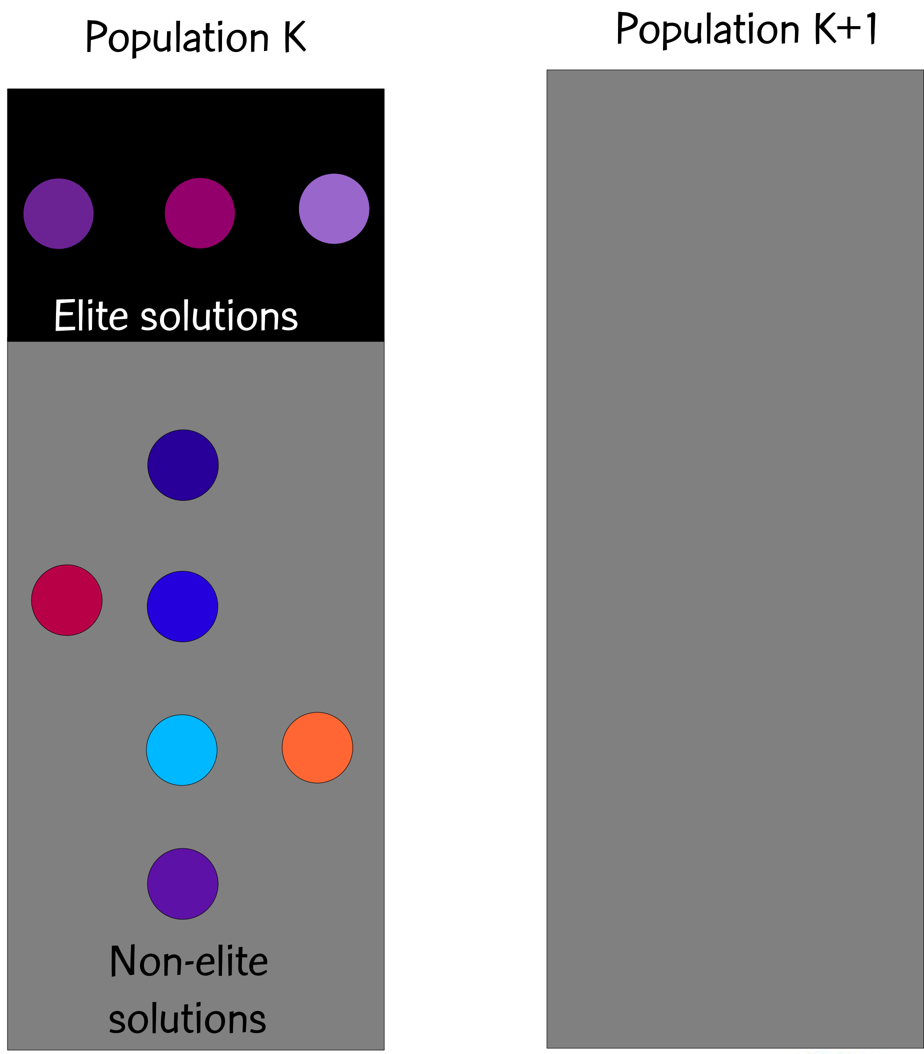
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- At the K-th generation, compute the cost of each solution and partition the solutions into two sets: elite solutions, non-elite solutions. Elite set should be smaller of the two sets and contain best solutions.



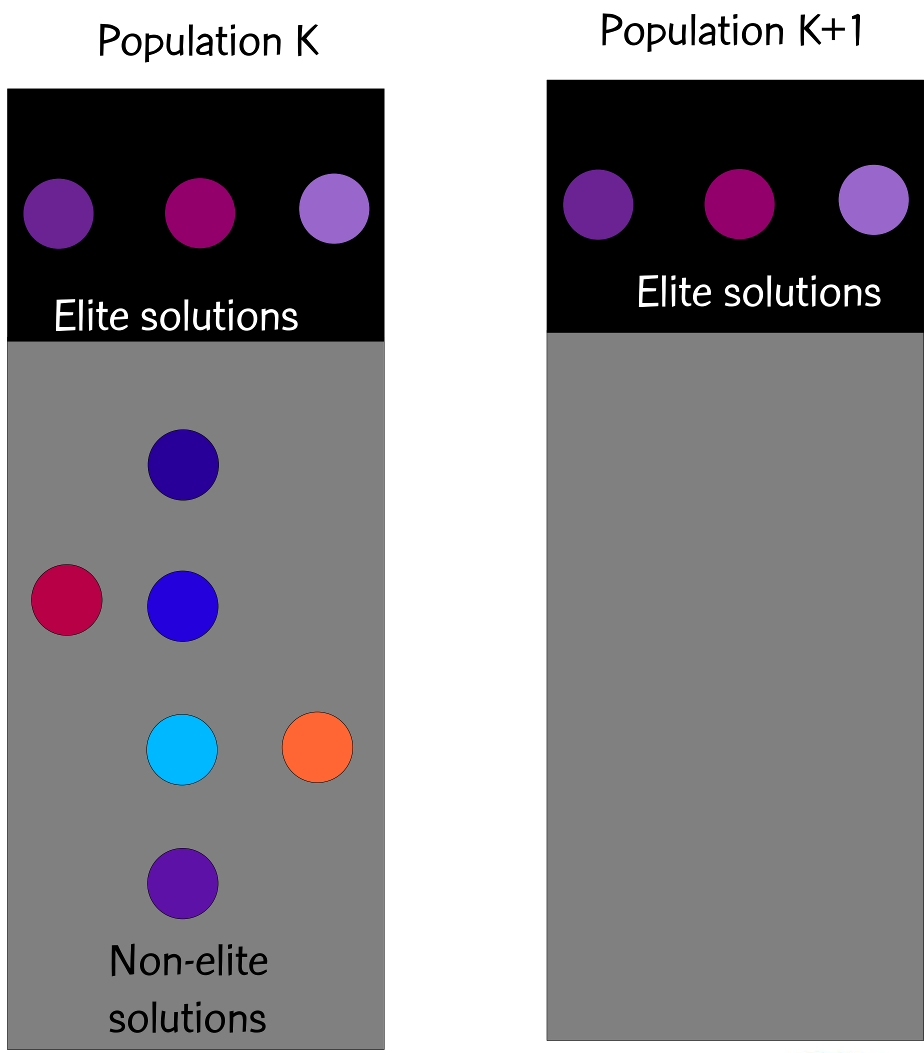
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Evolutionary dynamics



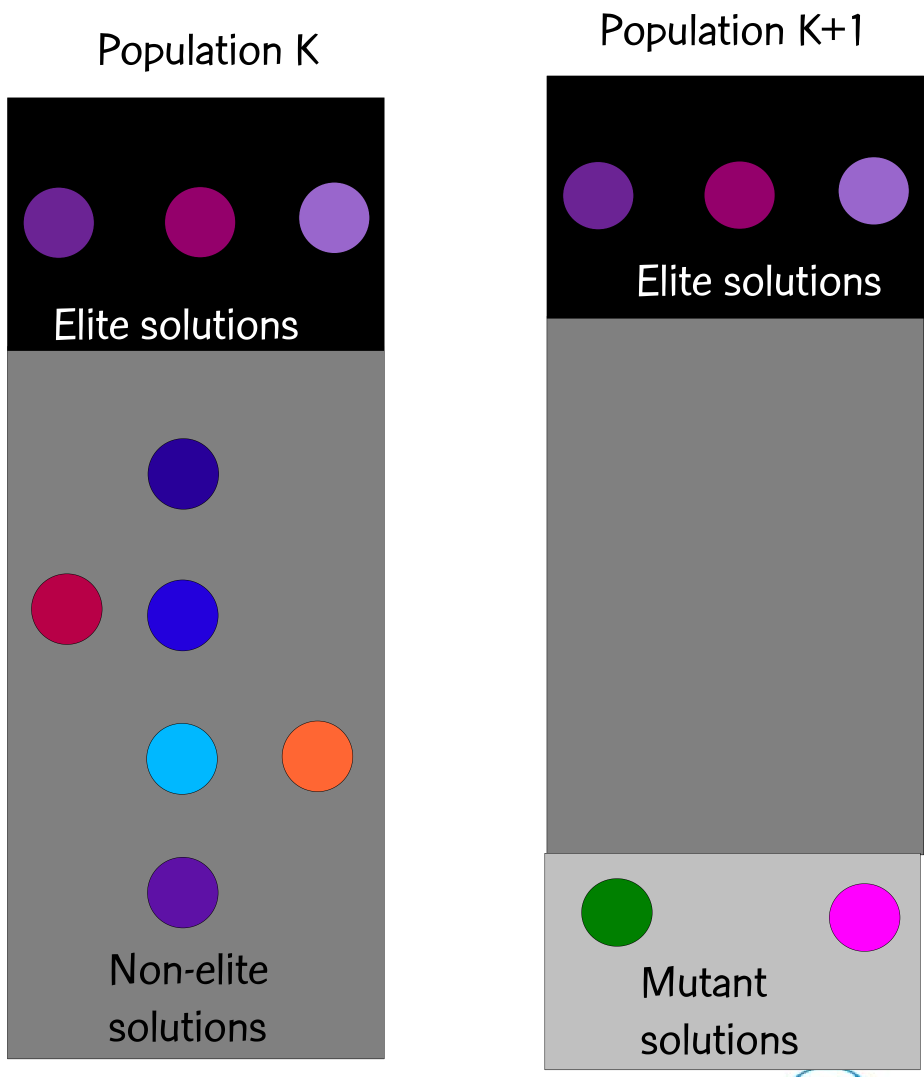
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Evolutionary dynamics
  - Copy elite solutions from population K to population K+1



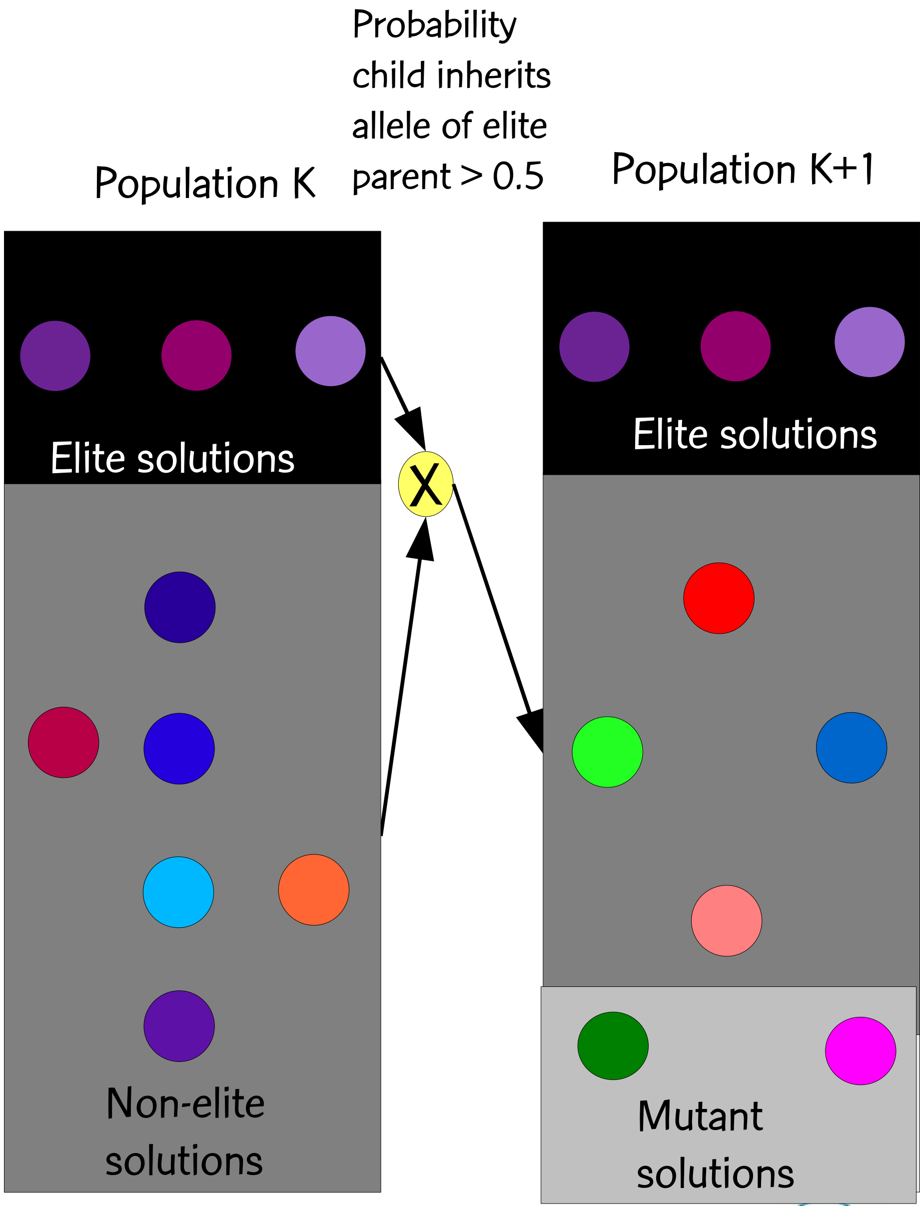
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
- Evolutionary dynamics
  - Copy elite solutions from population K to population K+1
  - Add R random solutions (mutants) to population K+1



GAs and random keys

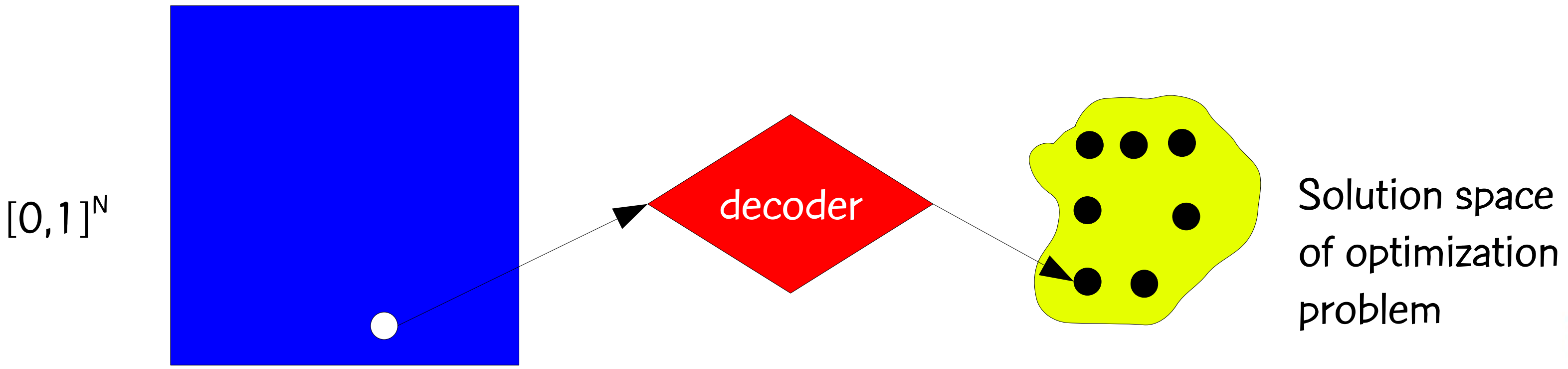
- Introduced by Bean (1994) for sequencing problems.
- Evolutionary dynamics
  - Copy elite solutions from population K to population K+1
  - Add R random solutions (mutants) to population K+1
  - While K+1-th population < P
    - Mate elite solution with non elite to produce child in population K+1. Mates are chosen at random.



at&t Your world. Delivered.

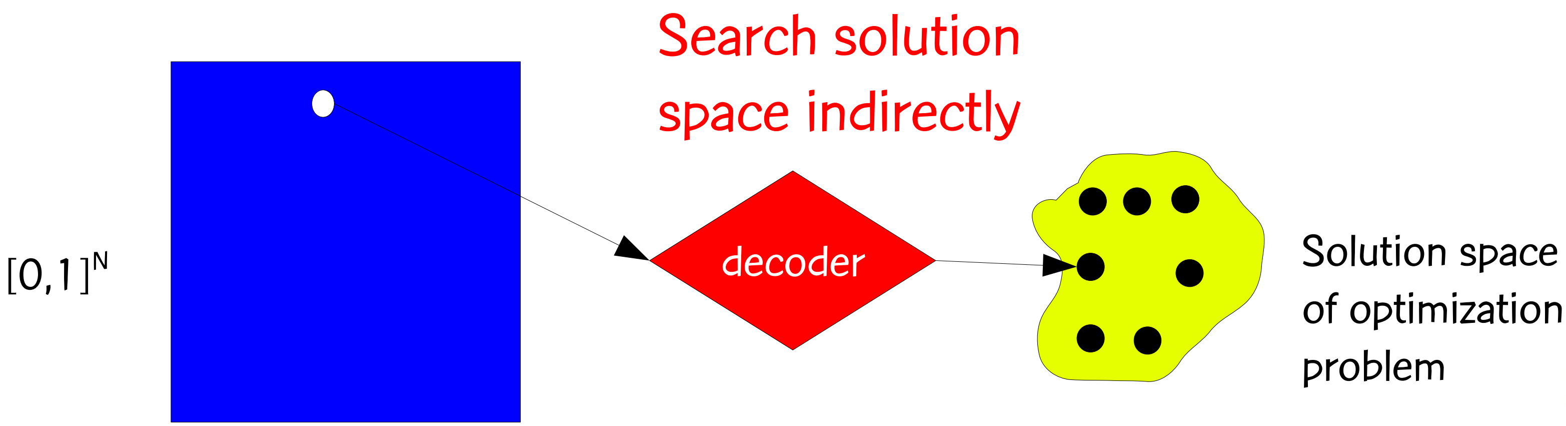
Decoders

- A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.
- Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.
- A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.



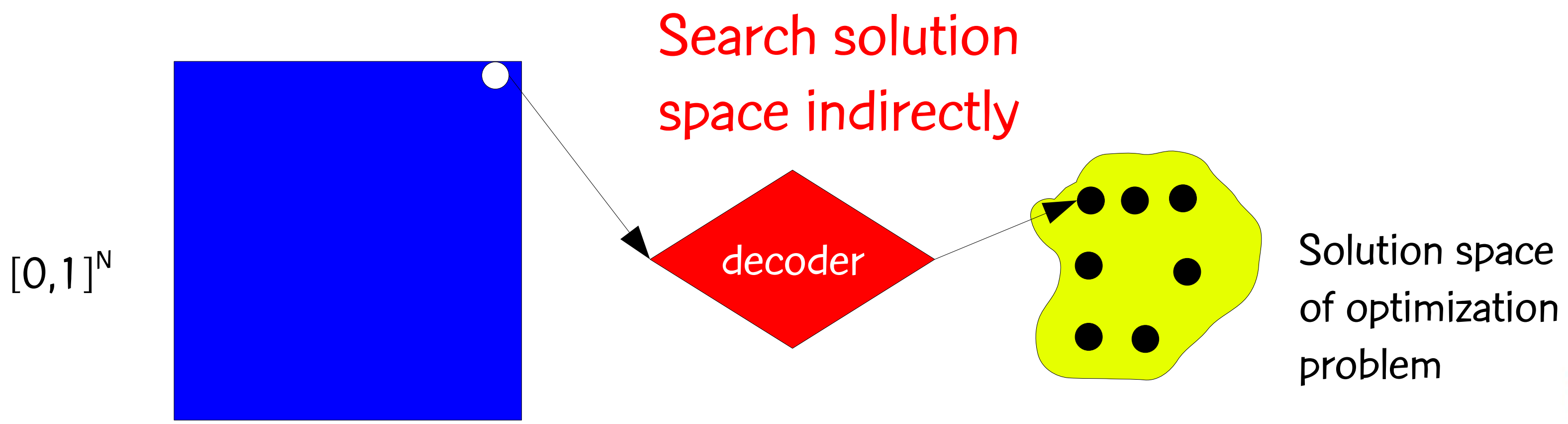
Decoders

- A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.
- Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.
- A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.



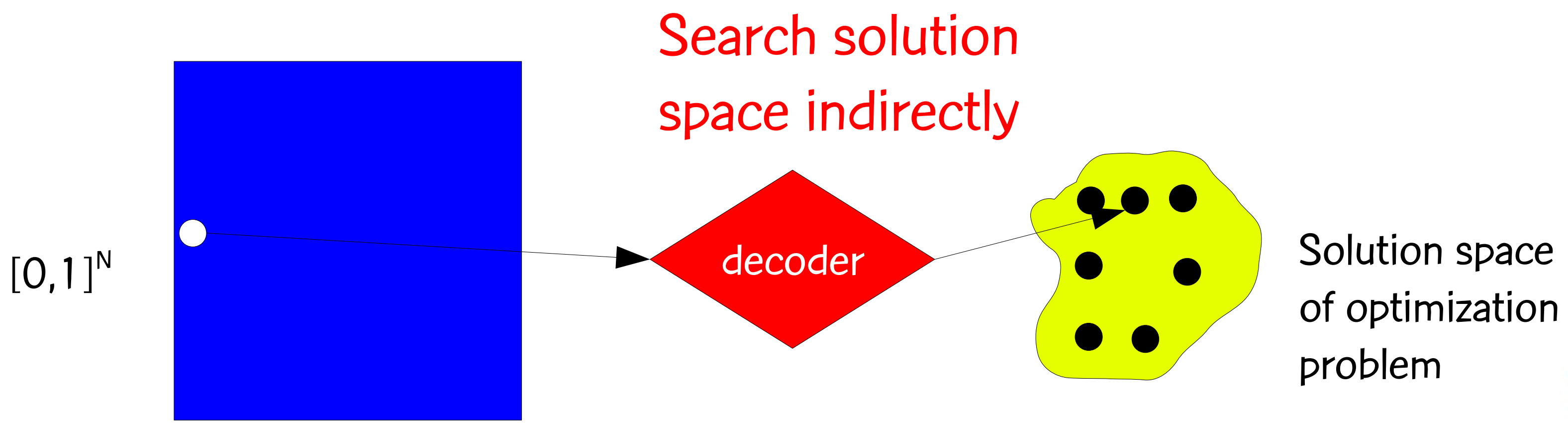
Decoders

- A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.
- Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.
- A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.

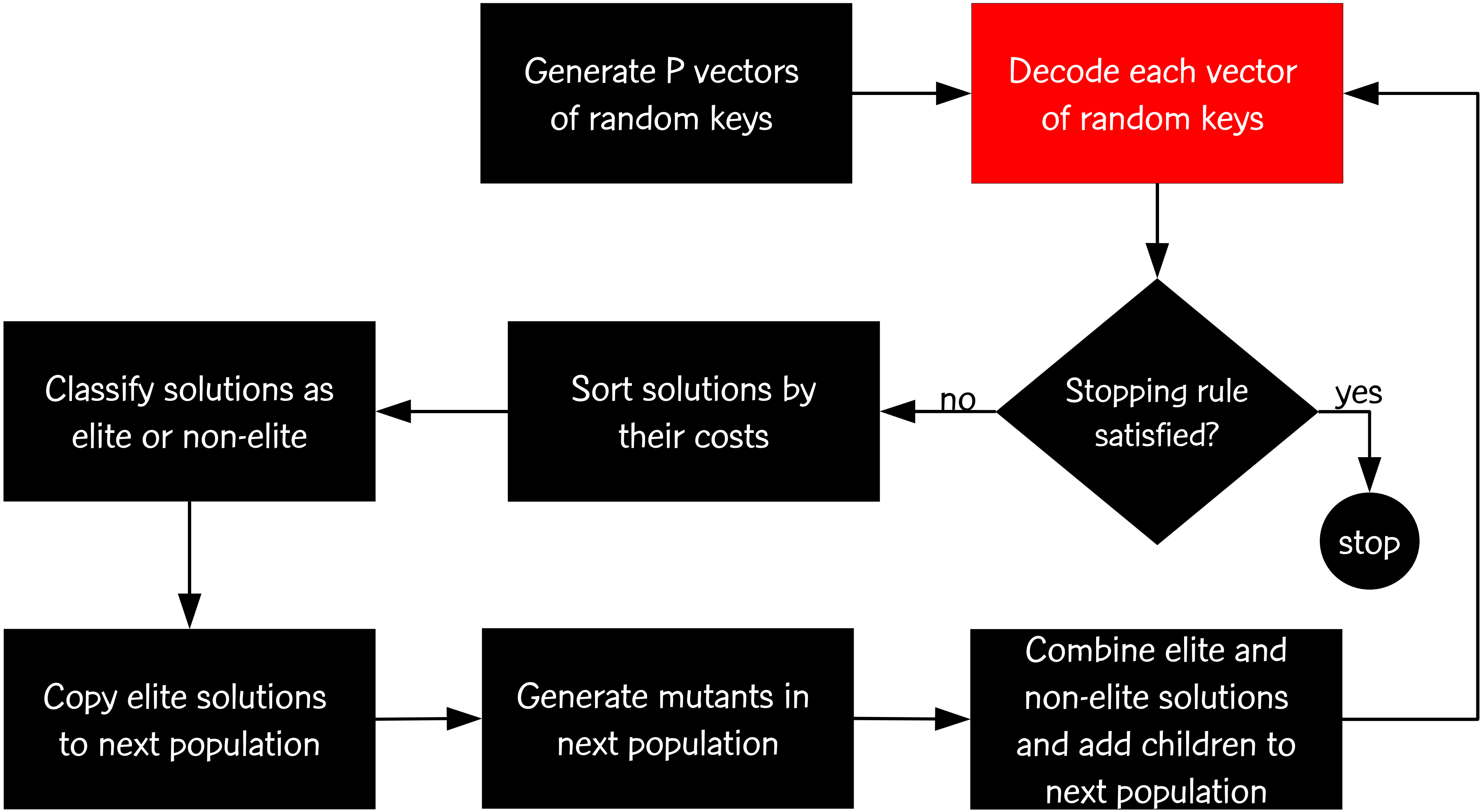


Decoders

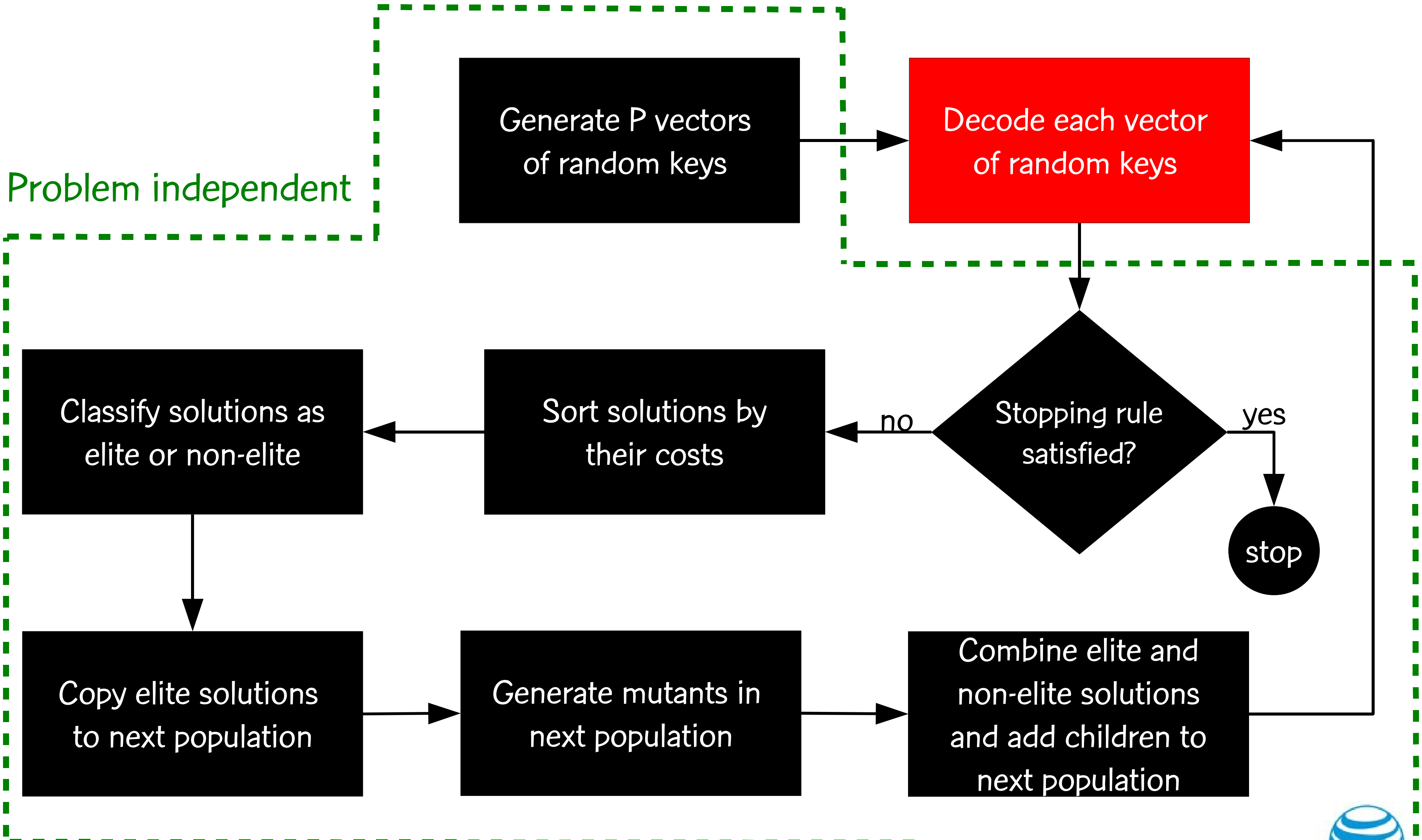
- A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.
- Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.
- A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.



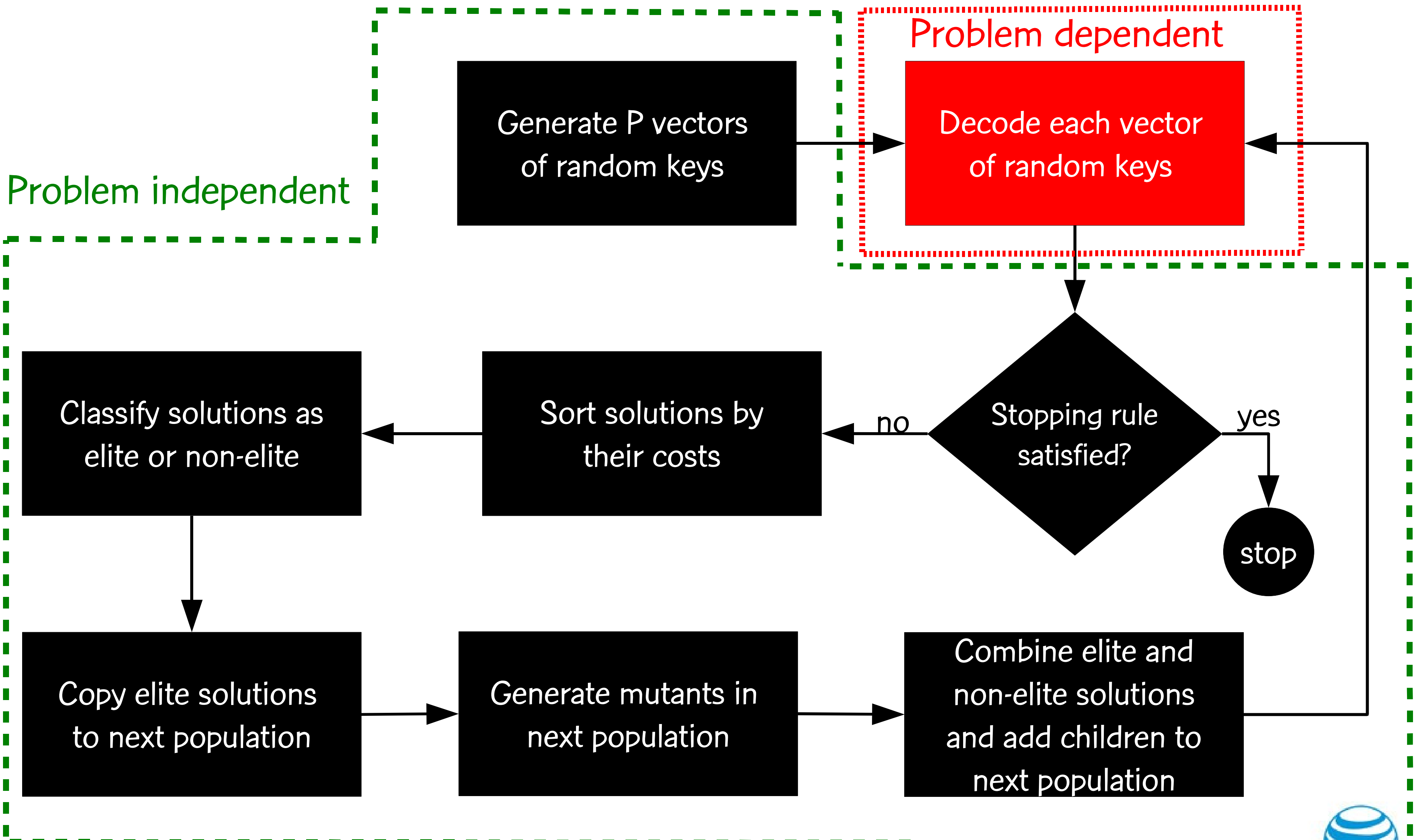
Framework for random-key genetic algorithms



Framework for random-key genetic algorithms



Framework for random-key genetic algorithms



OSPF routing in IP networks

Routing in IP networks

- Protocol: In OSPF, traffic is routed on shortest weight paths from origination router to destination router.
- Splitting: If more than one link out of a router is on a shortest weight path, traffic is evenly distributed on those links.
- Weight setting problem: Determine OSPF weights such that if traffic is routed according to OSPF protocol, network congestion is minimized.

Minimization of congestion

- Consider the directed capacitated network $G = (N,A,c)$, where N are routers, A are links, and c_a is the capacity of link $a \in A$.
- We use the measure of Fortz & Thorup (2000) to compute congestion:

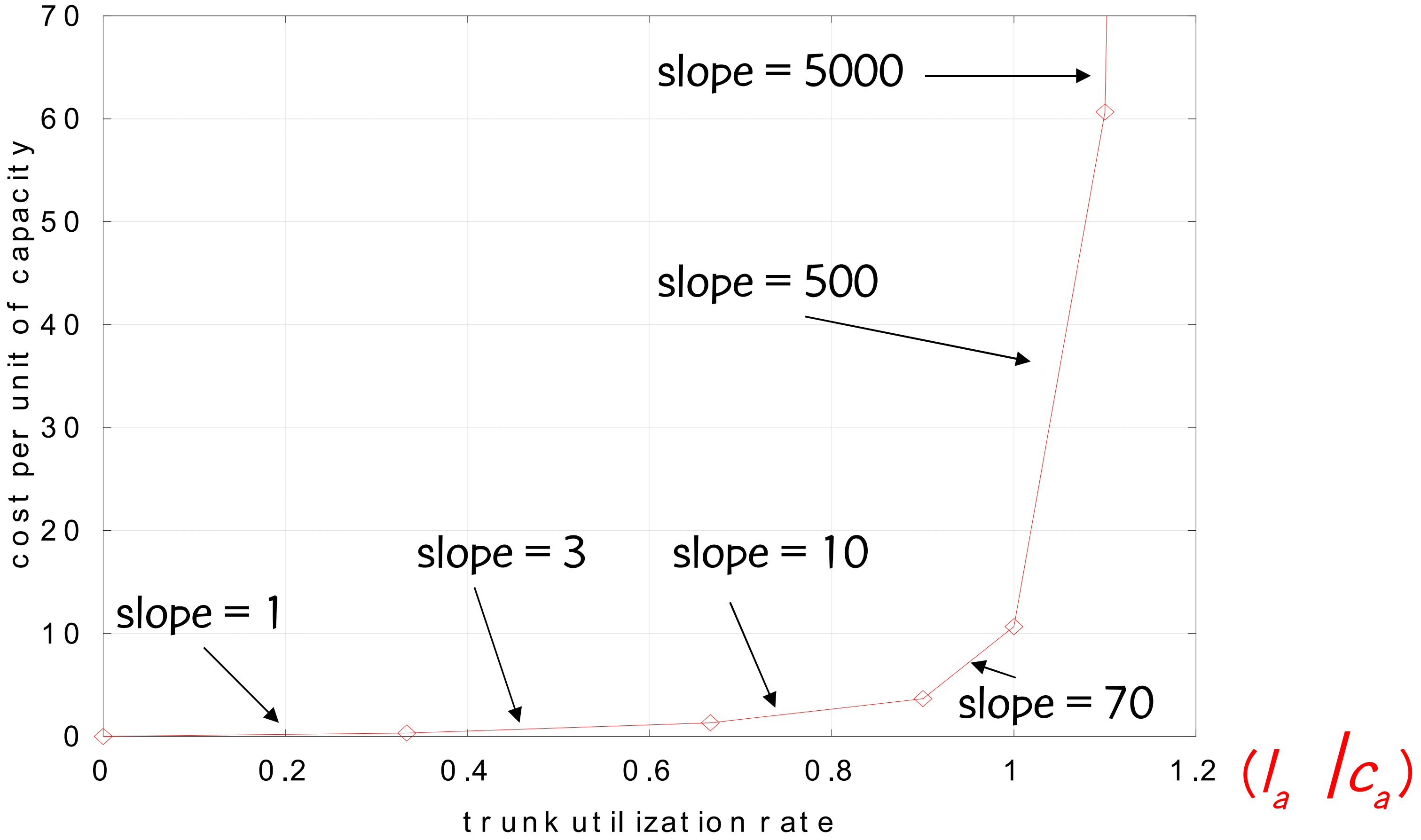
$$\Phi = \Phi_1(l_1) + \Phi_2(l_2) + \ldots + \Phi_{|A|}(l_{|A|})$$

where l_a is the load on link $a \in A$,

$\Phi_a(l_a)$ is piecewise linear and convex,

$\Phi_a(0) = 0$, for all $a \in A$.

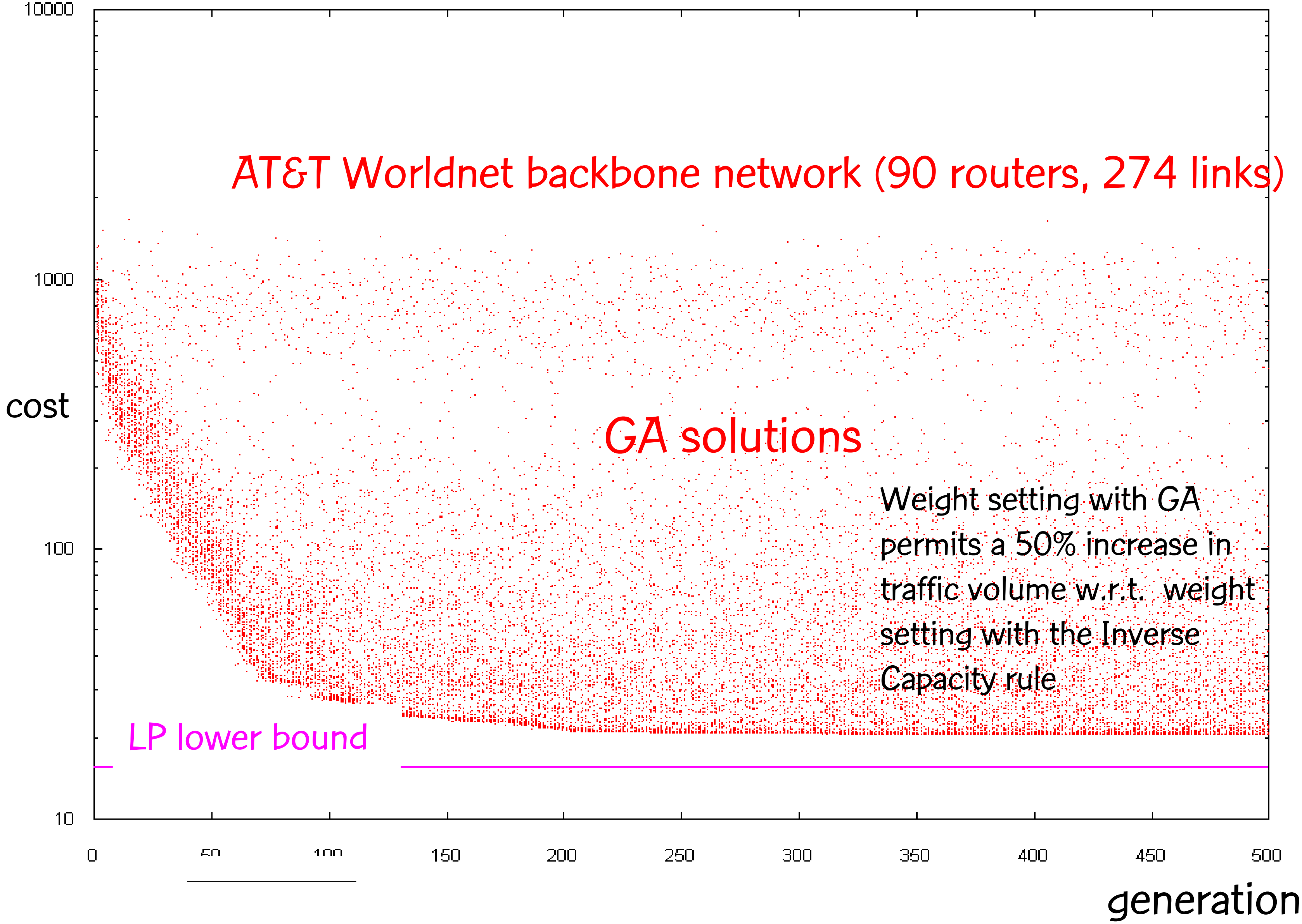
Piecewise linear and convex $\Phi_a(l_a)$ link congestion measure



Genetic algorithm for OSPF routing in IP networks

Ericsson, R., & Pardalos (J. Comb. Opt., 2002)

- Chromosome:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For i = 1,N: set $w(i) = \text{ceil}\,(\,X(i) \times w_{max}\,)$
  - Compute shortest paths and route traffic according to OSPF.
  - Compute load on each link, compute link congestion, add up all link congestions to compute network congestion.

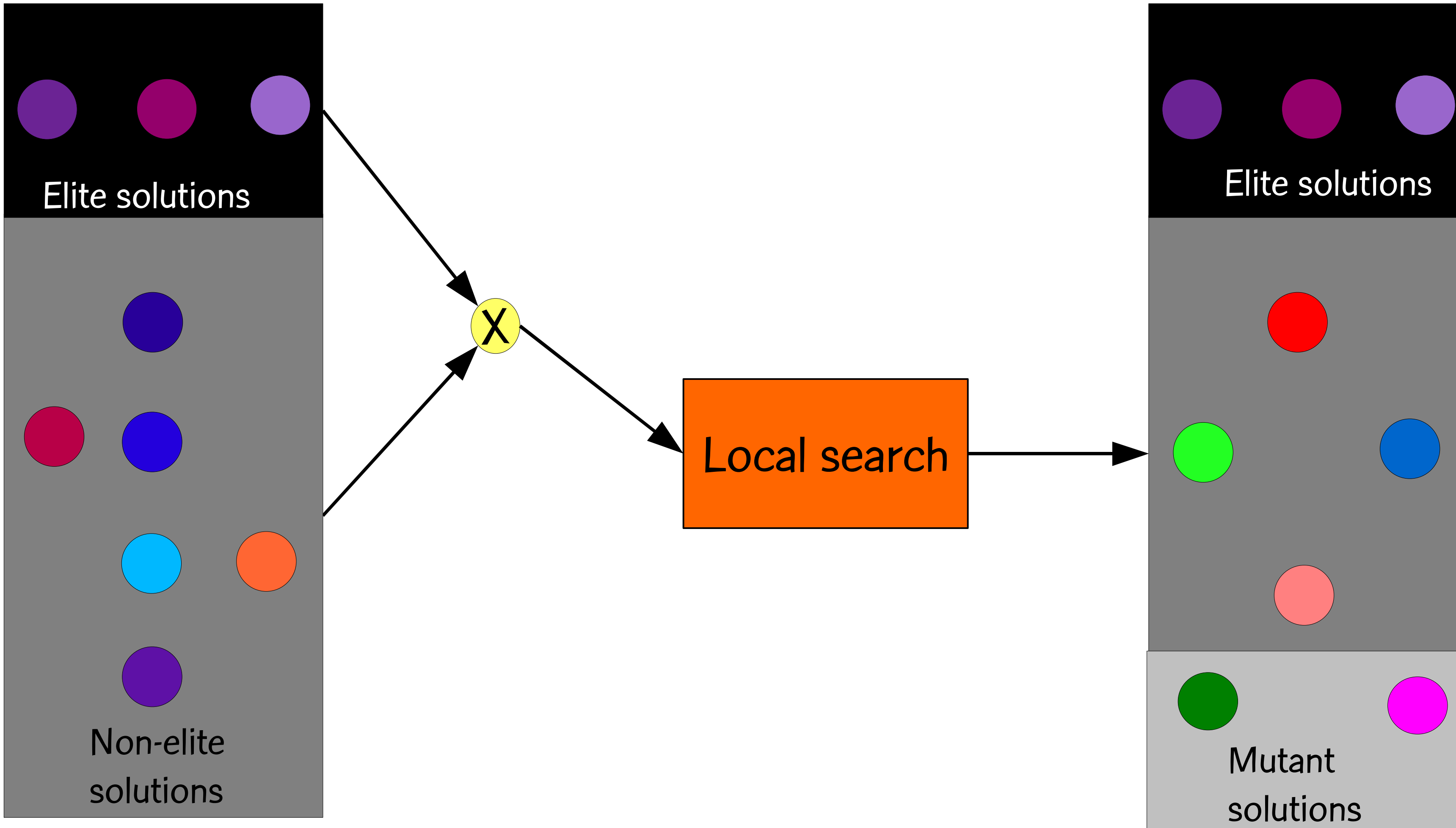

AT&T Worldnet backbone network (90 routers, 274 links)
GA solutions
Weight setting with GA permits a 50% increase in traffic volume w.r.t. weight setting with the Inverse Capacity rule
LP lower bound
cost
10000
1000
100
10
0
50
100
150
200
250
300
350
400
450
500
generation

Memetic algorithms for OSPF routing in IP networks

Buriol, R., Ribeiro, and Thorup (Networks, 2005)

- Chromosome:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For i = 1,N: set $w(i) = \text{ceil}\,(X(i) \times w_{max})$
  - Compute shortest paths and route traffic according to OSPF.
  - Compute load on each link, compute link congestion, add up all link congestions to compute network congestion.
  - Apply fast local search to improve weights.

Memetic algorithm: Optimized crossover = crossover + local search

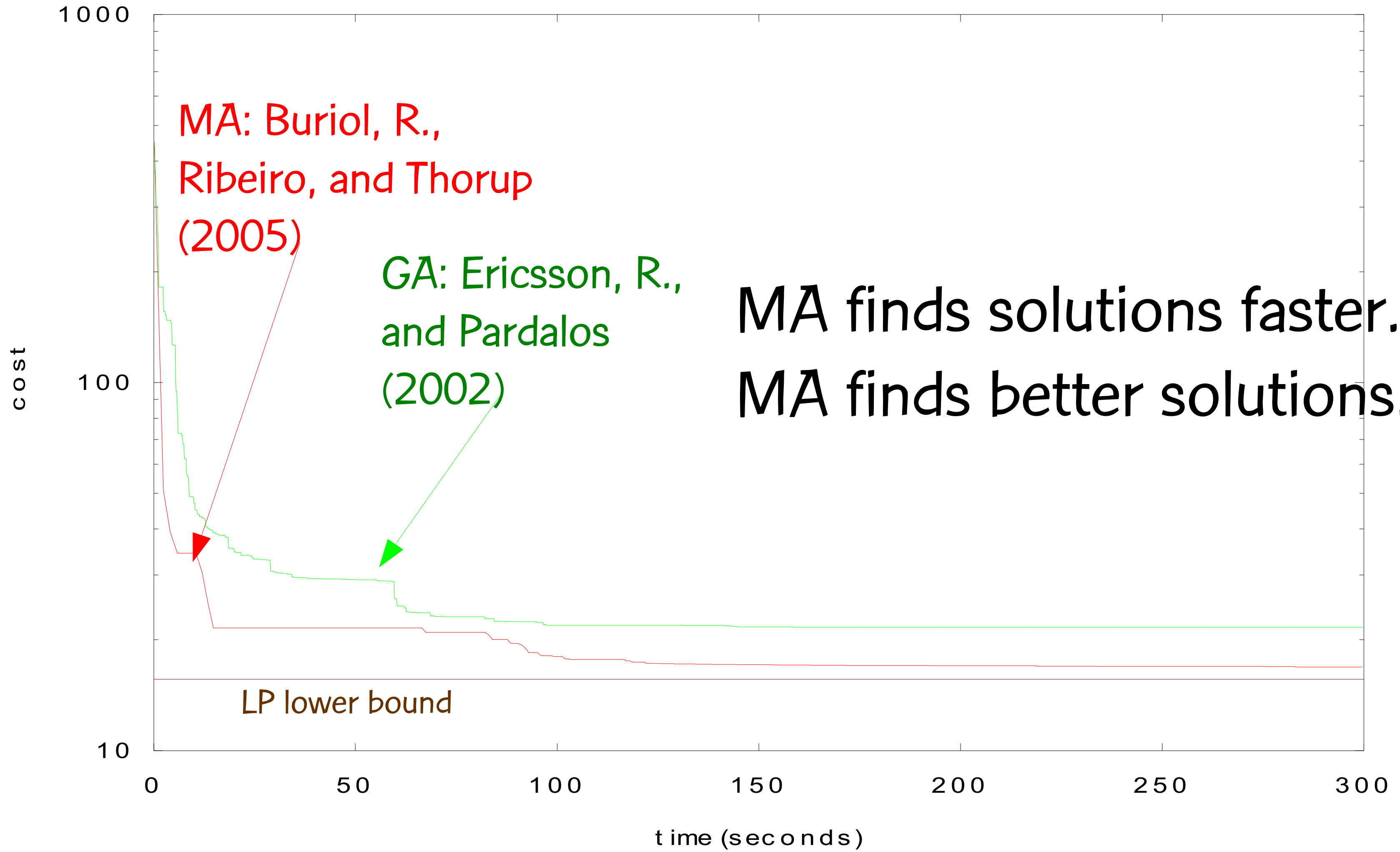


Fast local search

- Let A^* be the set of five arcs $a \in A$ having largest Φ_a values.
- Scan arcs $a \in A^*$ from largest to smallest Φ_a:
  - Increase arc weight, one unit at a time, in the range

    $$[w_a , w_a + \lceil (w_{max} - w_a)/4 \rceil]$$
  - If total cost Φ is reduced, restart local search.

Effect of decoder with fast local search



MA finds solutions faster.
MA finds better solutions.

Survivable IP network design

Survivable IP network design

Buriol, R., & Thorup (Networks, 2007)

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair $(s,t) \in N \times N$, specifies the demand $D(s,t)$ between s and t,
  - a cost $K(a)$ to lay fiber on arc a
  - a capacity increment C for the fiber.
- Determine
  - OSPF weight $w(a)$ to assign to each arc $a \in A$,
  - which arcs should be used to deploy fiber and how many units (multiplicities) $M(a)$ of capacity C should be installed on each arc $a \in A$,
- such that all the demand can be routed on the network even when any single arc fails.
- Min total design cost $= \sum_{a \in A} M(a) \times K(a)$.

Survivable IP network design

- Chromosome:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For i = 1,N: set $w(i) = \text{ceil}\,(X(i) \times w_{max})$
  - For each failure mode: route demand according to OSPF and for each link $i \in A$ determine load on link i.
  - For each link $i \in A$, compute multiplicity M(i) needed to accommodate maximum load over all failure modes.

iterate

  - Network design cost = $\sum_{i \in A} M(i) \times K(i)$.

at&t
Your world. Delivered.

Survivable composite link IP network design

Andrade, Buriol, R., & Thorup (INFORMS Telecom. Conf., 2006)

- Given a load $L(a)$ on arc a, we can compose several different link types that sum up to the needed capacity $c(a) \geq L(a)$:
  - $c(a) = \sum_{t \text{ used in arc } a} M(t,a) \times \gamma(t)$, where
  - $M(t,a)$ is the multiplicity of link type $t \in \{1, 2, \ldots, T\}$ on arc a
  - $\gamma(t)$ is the capacity of link type t: $\{\gamma(1), \gamma(2), \ldots, \gamma(T)\}$: $\gamma(i) < \gamma(i+1)$
- Assumptions
  - Prices / unit length = $\{ p(1), p(2), \ldots, p(T) \}$: $p(i) < p(i+1)$
  - $[p(T)/\gamma(T)] < [p(T-1)/\gamma(T-1)] < \cdots < [p(1)/\gamma(1)]$: economies of scale
  - $\gamma(i) = \alpha \times \gamma(i-1)$, for $\alpha \in N$, $\alpha > 1$, e.g.
    - $\gamma(OC192) = 4 \times \gamma(OC48)$
    - $\gamma(OC48) = 4 \times \gamma(OC12)$
    - $\gamma(OC12) = 4 \times \gamma(OC3)$

Survivable composite link IP network design

- Chromosome:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For i = 1,N: set $w(i) = \text{ceil}(X(i) \times w_{max})$
  - For each failure mode: route demand according to OSPF and for each arc $i \in A$ determine the load on arc i.
  - For each arc $i \in A$, determine the multiplicity M(t,i) for each link type t using the maximum load for that arc over all failure modes.
  - Network design cost = $\sum_{i \in A} \sum_{t \text{ used in arc } i} M(t,i) \times p(t)$

iterate

Concluding remarks

- We have just seen a few metaheuristics applied to network design problems.
- Even though there has been much progress in exact method for network design, I feel that these and other metaheuristics, as well as hybrids of metaheuristics, will continue to play a big role in network design.

The End

These slides and all papers cited in this tutorial can be downloaded from my homepage:

http://mauricioresende.com