# GRASP heuristics for discrete & continuous global optimization

Talk given at Universidade Federal de São Carlos
São Carlos (SP) Brazil ✤ March 26, 2013


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

# Summary

## Combinatorial optimization and a review of GRASP

Neighborhoods, local search, greedy randomized construction and diversification

## Hybrid construction

Other greedy randomized constructions, reactive GRASP, long-term memory in construction, biased sampling, cost perturbation

## Hybrid local search

Variable neighborhood descent, variable neighborhood search, short-term memory tabu search, simulated annealing, iterated local search, very large-scale neighborhood search

# Summary

Hybridization with path-relinking

Elite sets, forward, backward, back and forward, mixed, greedy randomized adaptive path-relinking, evolutionary path-relinking

Continuous GRASP for bound constrained global optimization

Concluding remarks

# Combinatorial Optimization

# Combinatorial Optimization

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

**Applications:** e.g. routing, scheduling, packing, inventory and production management, location, logic, and assignment of resources.

**Economic impact:** e.g. transportation (airlines, trucking, rail, and shipping), forestry, manufacturing, logistics, aerospace, energy (electrical power, petroleum, and natural gas), agriculture, biotechnology, financial services, and telecommunications.

# Combinatorial Optimization

Given:

discrete set of solutions $X$

objective function $f(x)$: $x \in X \rightarrow R$

Objective (minimization):

find $x \in X : f(x) \leq f(y), \forall\, y \in X$

# Combinatorial Optimization

Much progress in recent years on finding exact (provably optimal) solutions: dynamic programming, cutting planes, branch and cut, ...

Many hard combinatorial optimization problems are still not solved exactly and require good solution methods.

# Combinatorial Optimization

Approximation algorithms are guaranteed to find in polynomial-time a solution within a given factor of the optimal.

# Combinatorial Optimization

Approximation algorithms are guaranteed to find in polynomial-time a solution within a given factor of the optimal.

Sometimes the factor is too big, i.e. guaranteed solutions are far from optimal

Some optimization problems (e.g. max clique, covering by pairs) cannot have approximation schemes unless P=NP

# 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 heuristics to devise heuristics.

Examples: simulated annealing, genetic algorithms, tabu search, scatter search, ant colony optimization, variable neighborhood search, and GRASP.

# 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, genetic algorithms, tabu search, scatter search, ant colony optimization, variable neighborhood search, and GRASP.

# 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, genetic algorithms, tabu search, scatter search, ant colony optimization, variable neighborhood search, and GRASP.

# Review of GRASP: Local Search

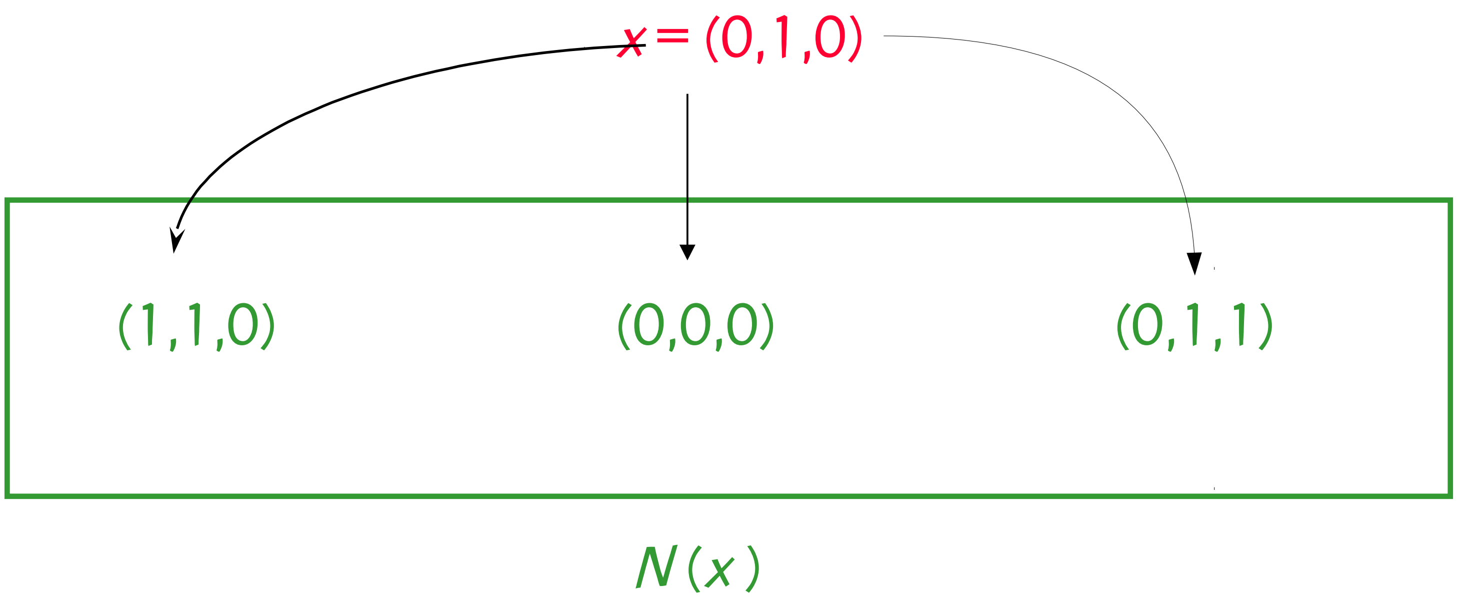
# Local Search

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.

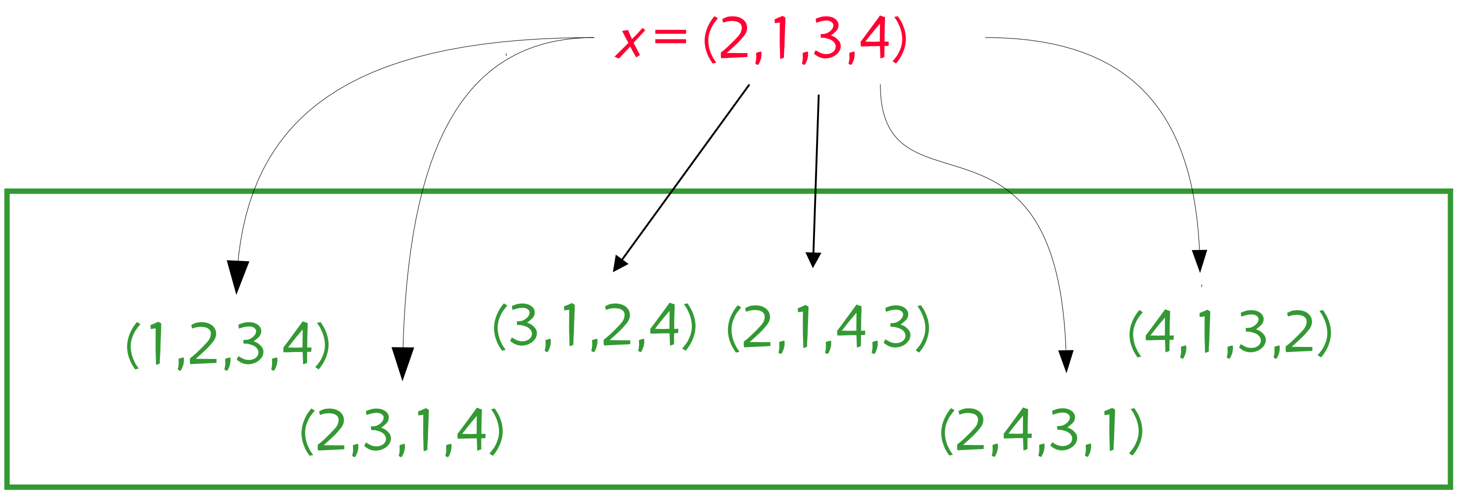
# Local Search Neighborhoods

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



# Local Search Neighborhoods

Consider $x = (2,1,3,4)$ and the 2-swap neighborhood of a permutation array.




$$N(x) = C(4,2) = 6$$

# Local Search

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

~~check~~ for better solution in neighborhood of

$x = x_0$ ;

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

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

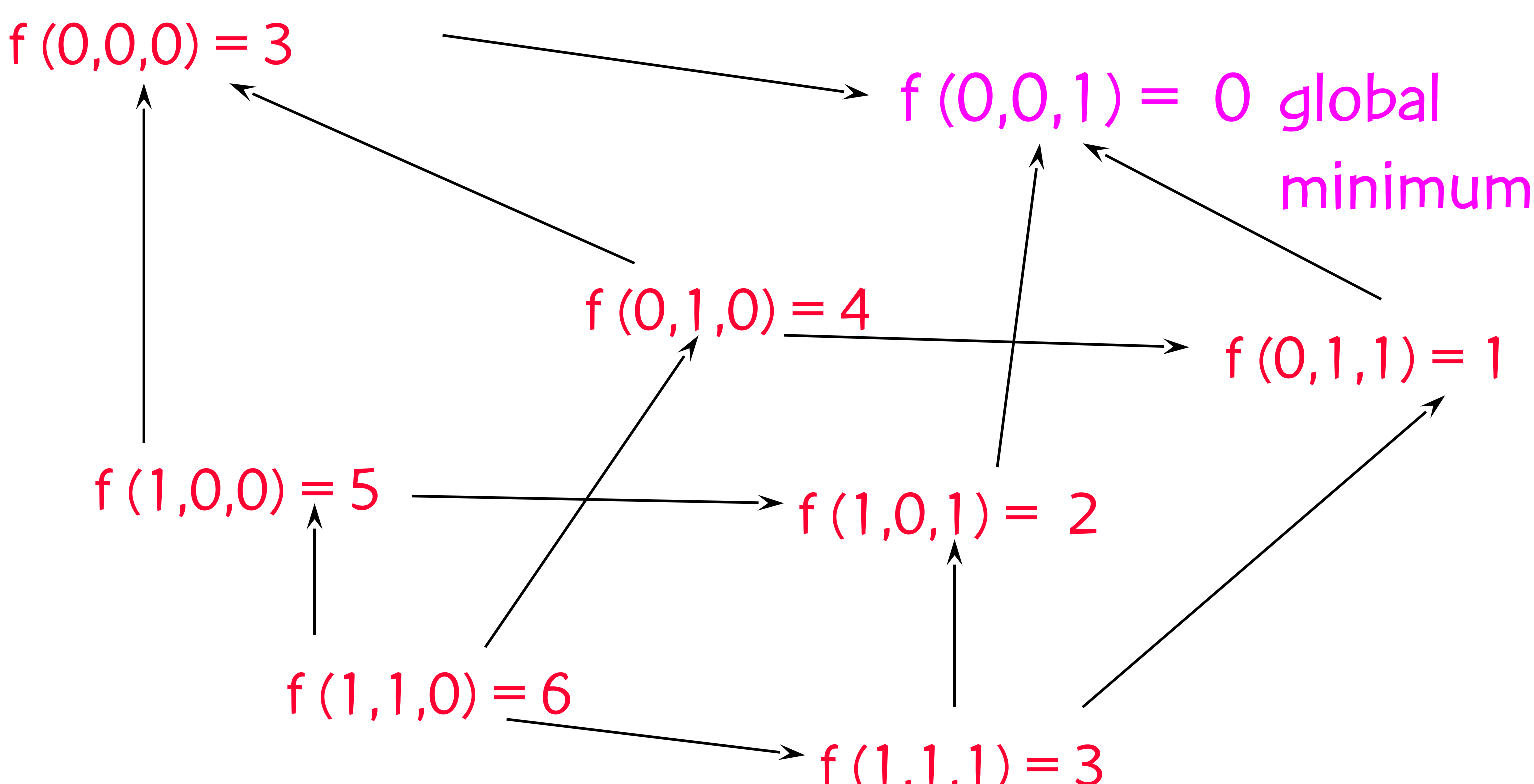
}

Time complexity of local search can be exponential.

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

# Local Search

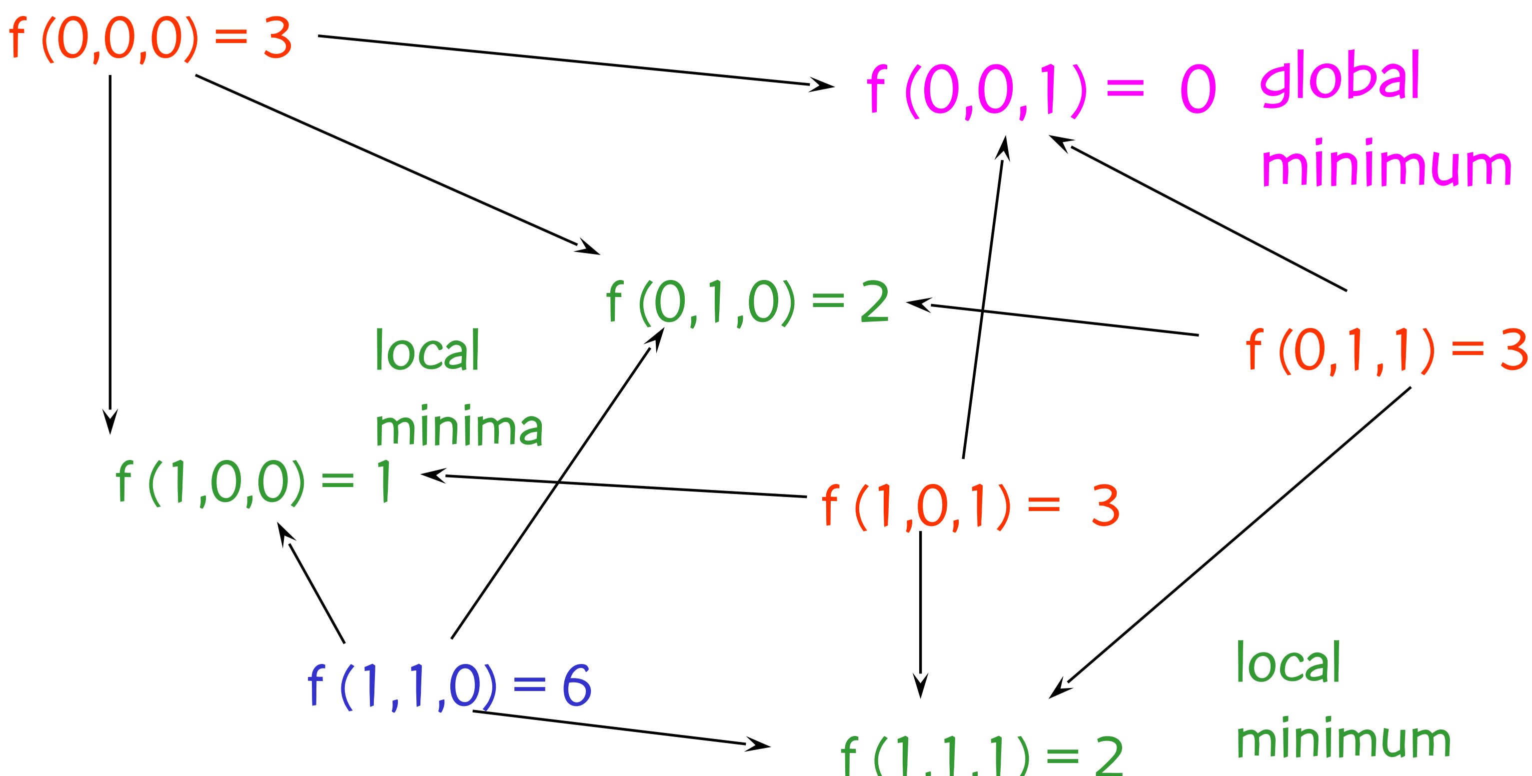
(ideal situation)




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

function to be minimized

starting solution

usually pre-determined

usually easier to control

# Multi-start method

$c^* = \infty$

repeat

$x = \text{method}()$

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

    $x^* = x$

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

endif

# 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 by random selection

Suppose $x = (0/1, 0/1, 0/1, 0/1, 0/1)$ and let the unique optimum be $x^* = (1,0,0,1,1)$.

The prob of finding the opt at random is $1/32 = .031$ and the prob of not finding it is $31/32$.

After k trials, the probability of not finding the opt is $(31/32)^k$ and hence the prob of find it at least once is $1 - (31/32)^k$

For $k = 5$, $p = .146$; for $k = 10$, $p = .272$; for $k = 20$, $p = .470$; for $k = 50$, $p = .796$; for $k = 100$, $p = .958$; for $k = 200$, $p = .998$

# 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 |

# Greedy algorithm

# The greedy algorithm

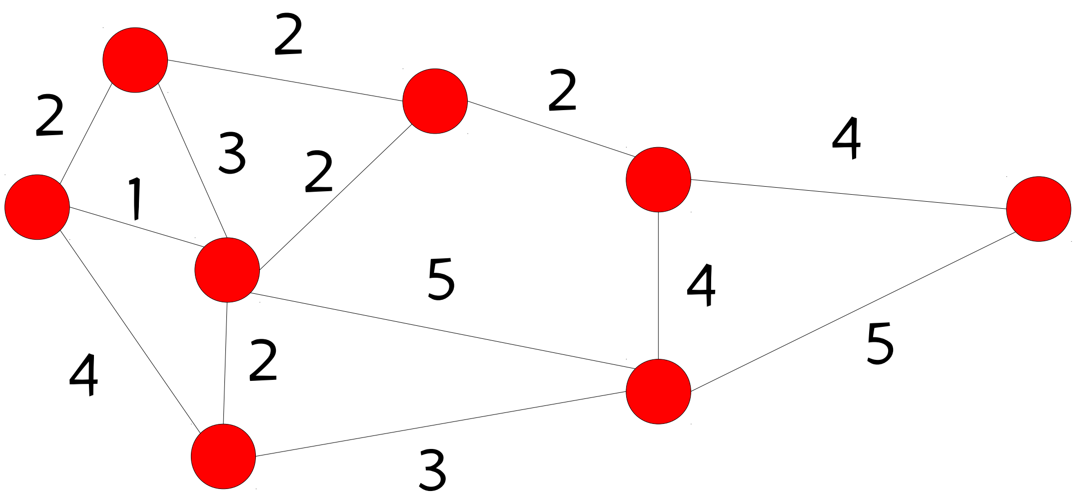
Constructs a solution, one element at a time:

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.

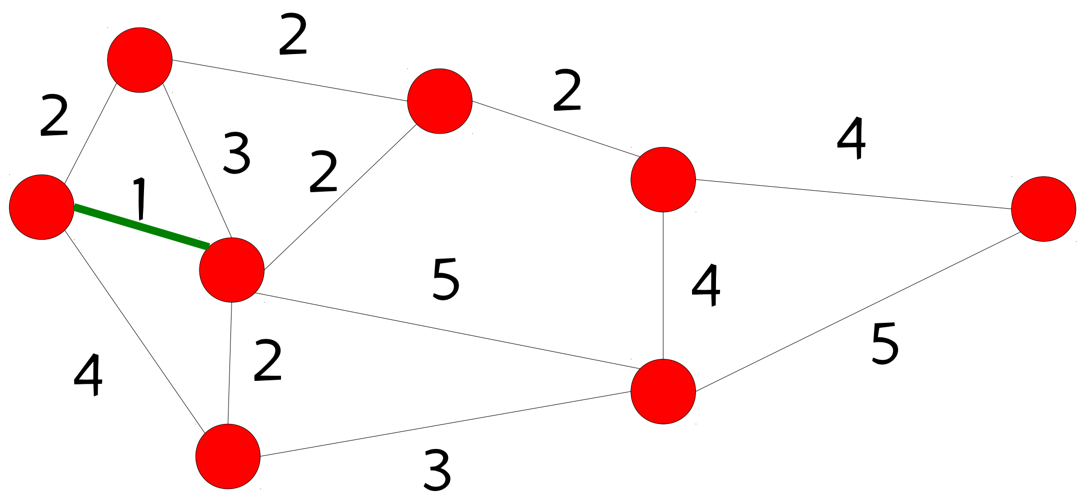
# The greedy algorithm

## An example: minimum weight spanning tree



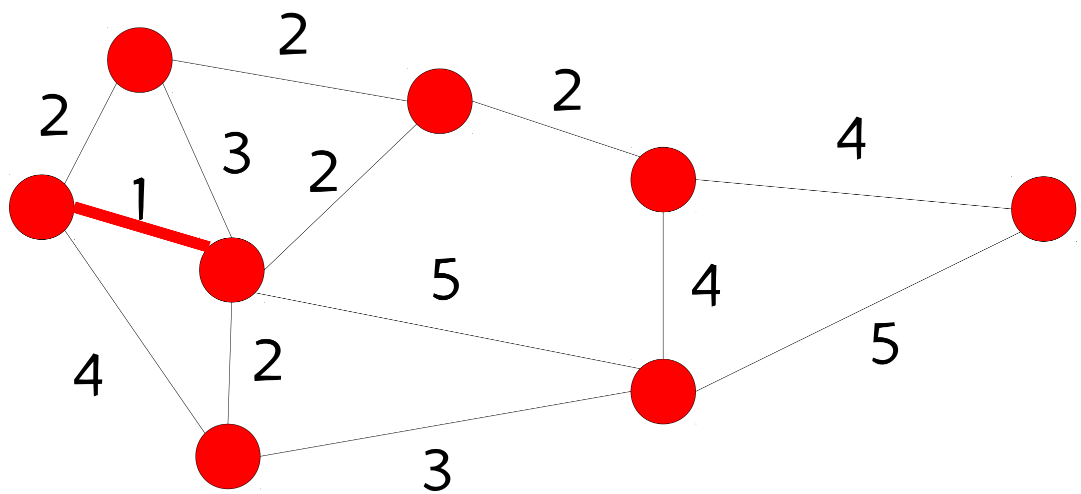
# The greedy algorithm

## An example: minimum weight spanning tree



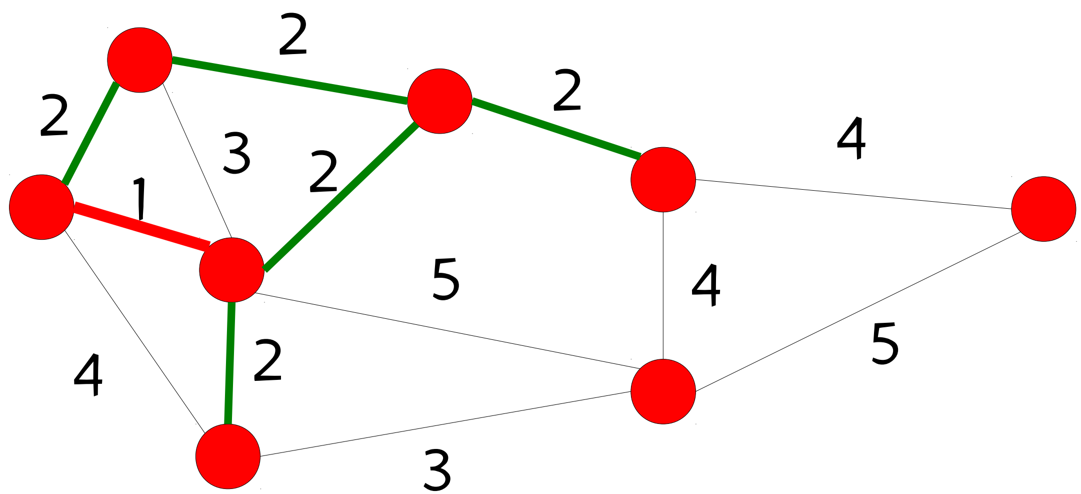
# The greedy algorithm

## An example: minimum weight spanning tree



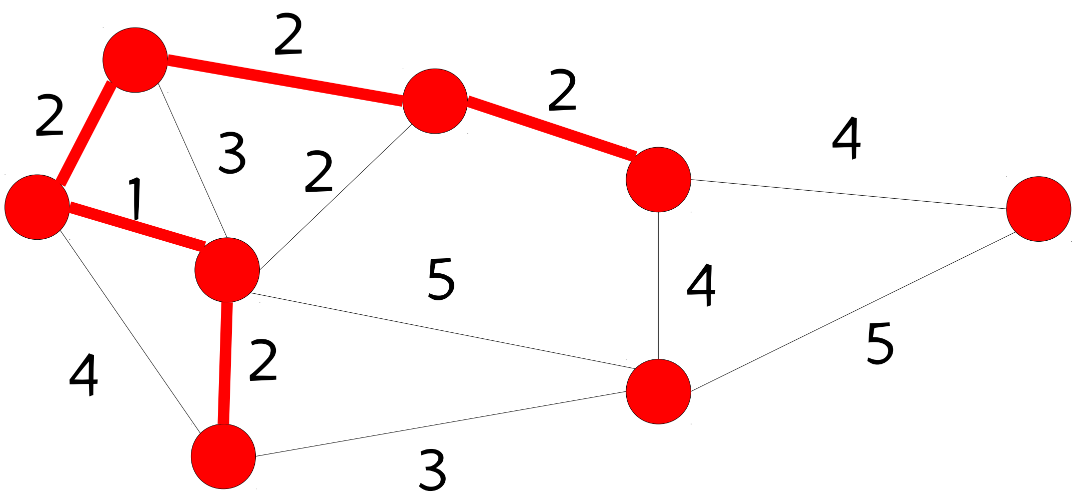
# The greedy algorithm

## An example: minimum weight spanning tree



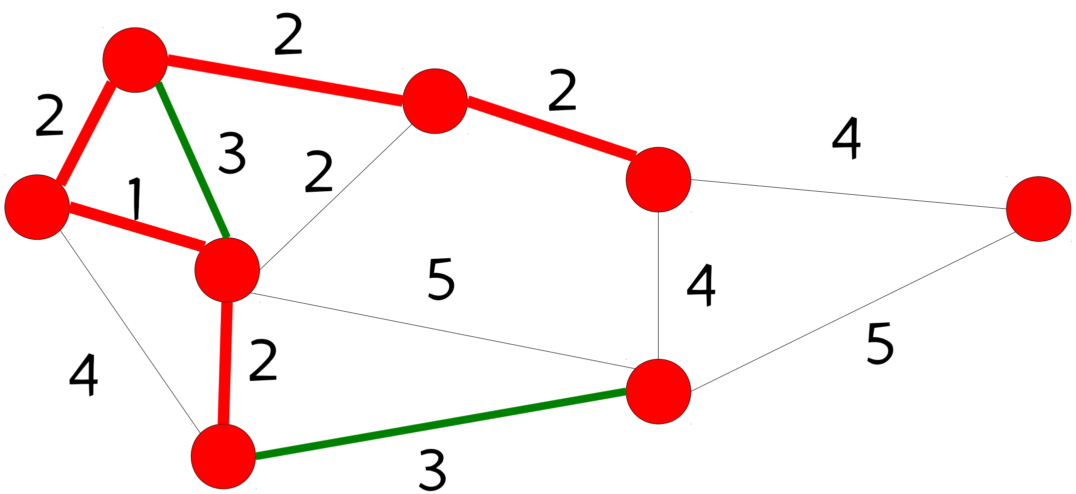
# The greedy algorithm

## An example: minimum weight spanning tree



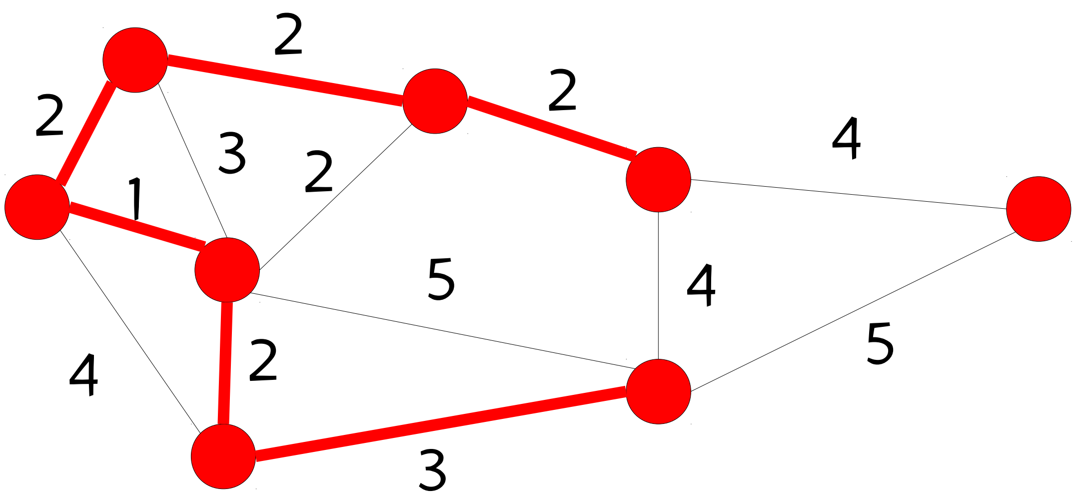
# The greedy algorithm

## An example: minimum weight spanning tree



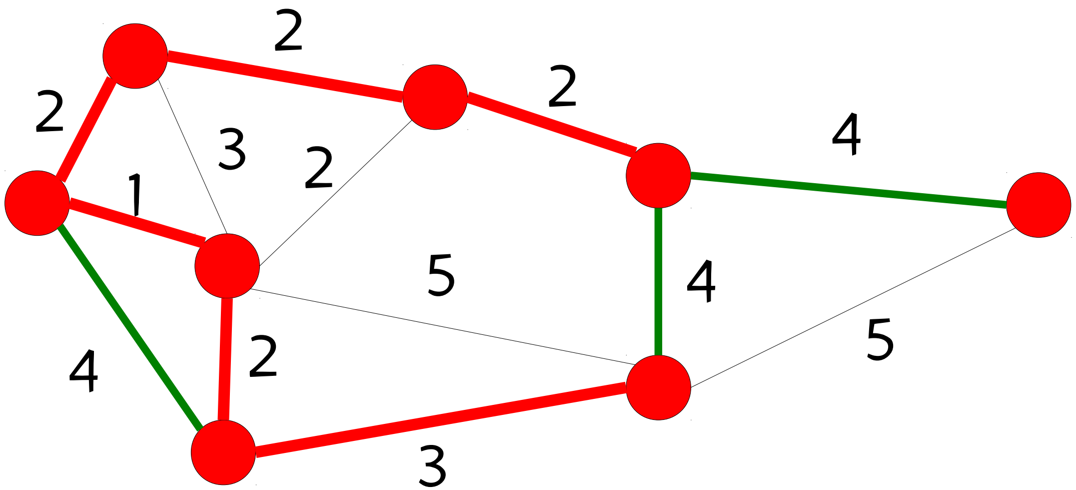
# The greedy algorithm

## An example: minimum weight spanning tree



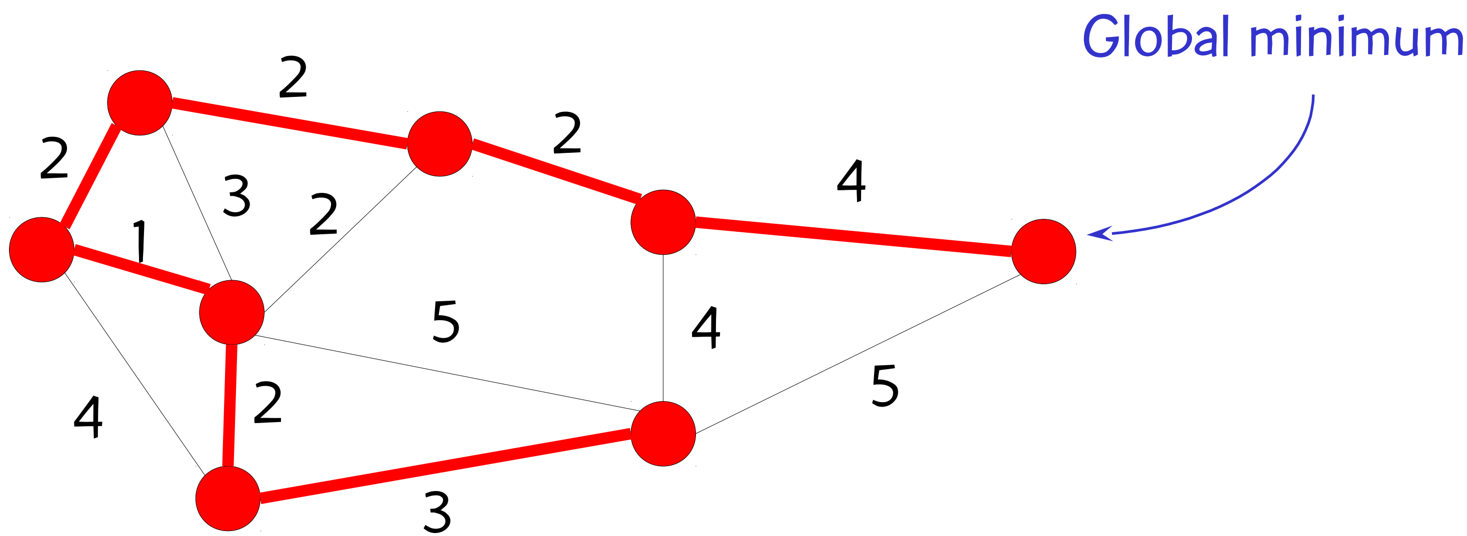
# The greedy algorithm

## An example: minimum weight spanning tree



# The greedy algorithm

## An example: minimum weight spanning tree



# The greedy algorithm

## Another example: Maximum clique

Given graph $G = (V, E)$, find largest subgraph of $G$ such that all vertices are mutually adjacent.

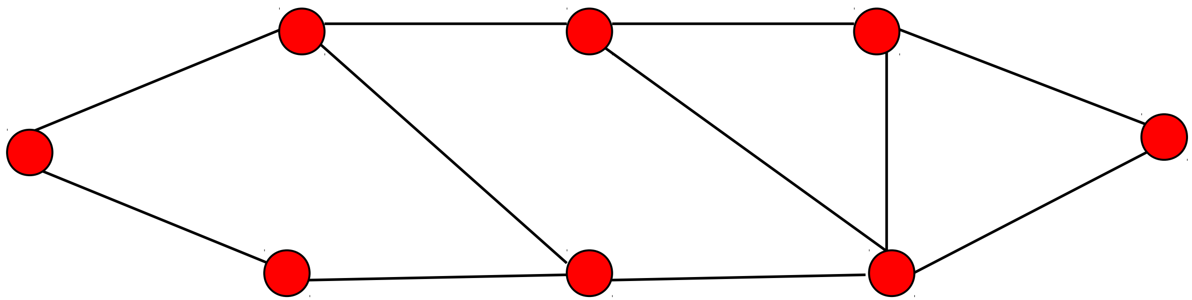
greedy algorithm builds solution, one element (vertex) at a time

candidate set: unselected vertices adjacent to all selected vertices

greedy function: vertex degree with respect to other candidate set vertices.

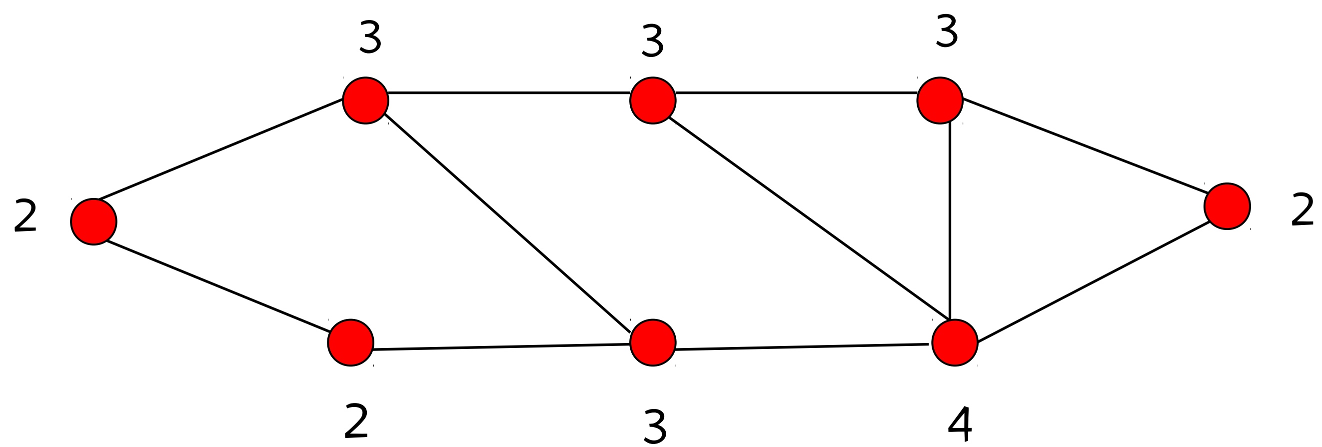
# The greedy algorithm

## Another example: Maximum clique



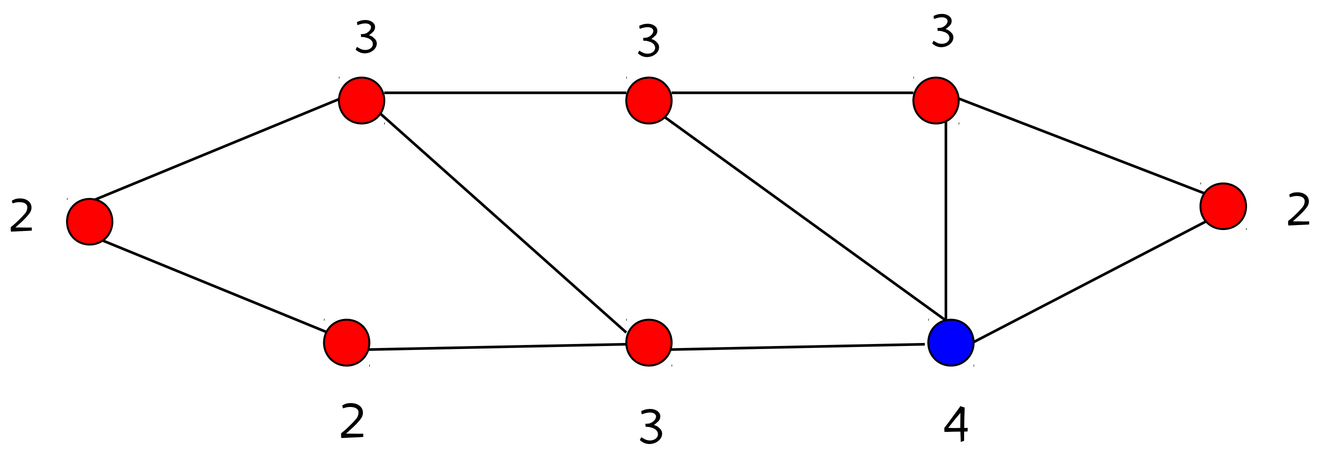
# The greedy algorithm

## Another example: Maximum clique



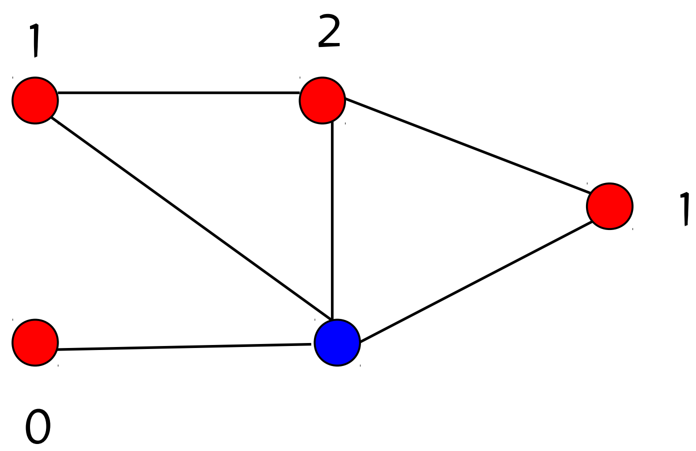
# The greedy algorithm

## Another example: Maximum clique



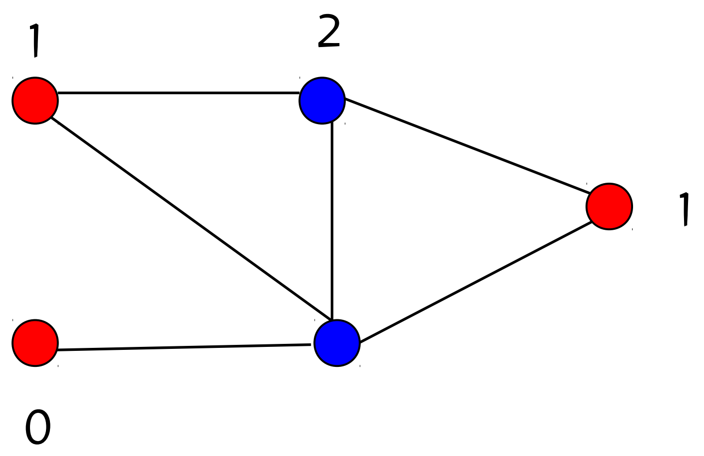
# The greedy algorithm

## Another example: Maximum clique



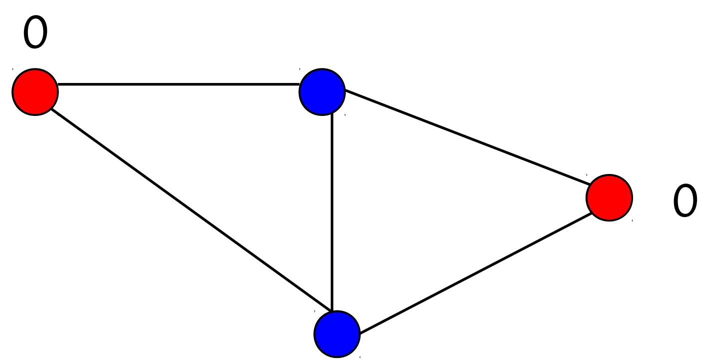
# The greedy algorithm

## Another example: Maximum clique



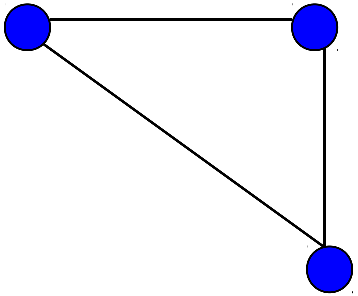
# The greedy algorithm

## Another example: Maximum clique



# The greedy algorithm

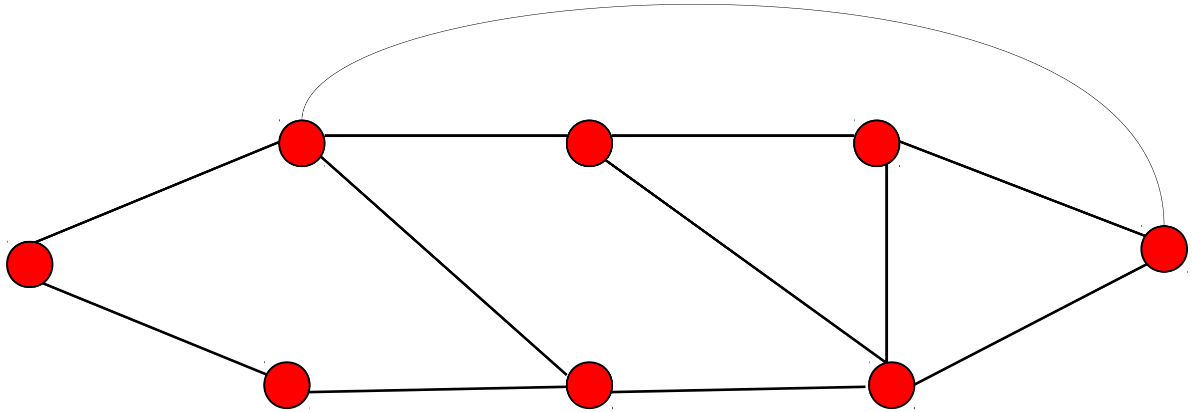
## Another example: Maximum clique



global maximum

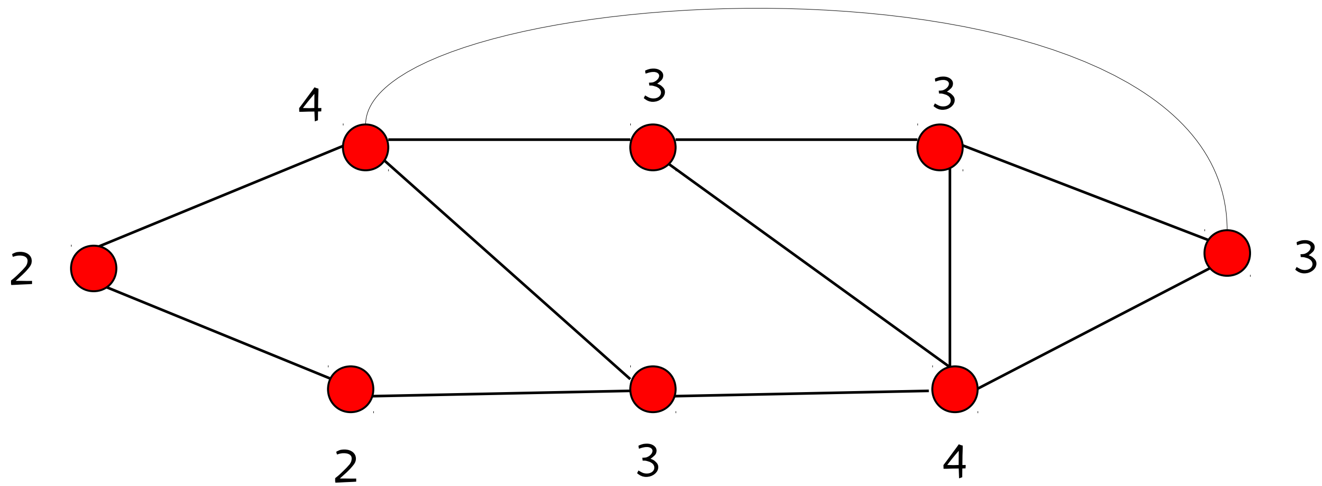
# The greedy algorithm

## Another example: Maximum clique



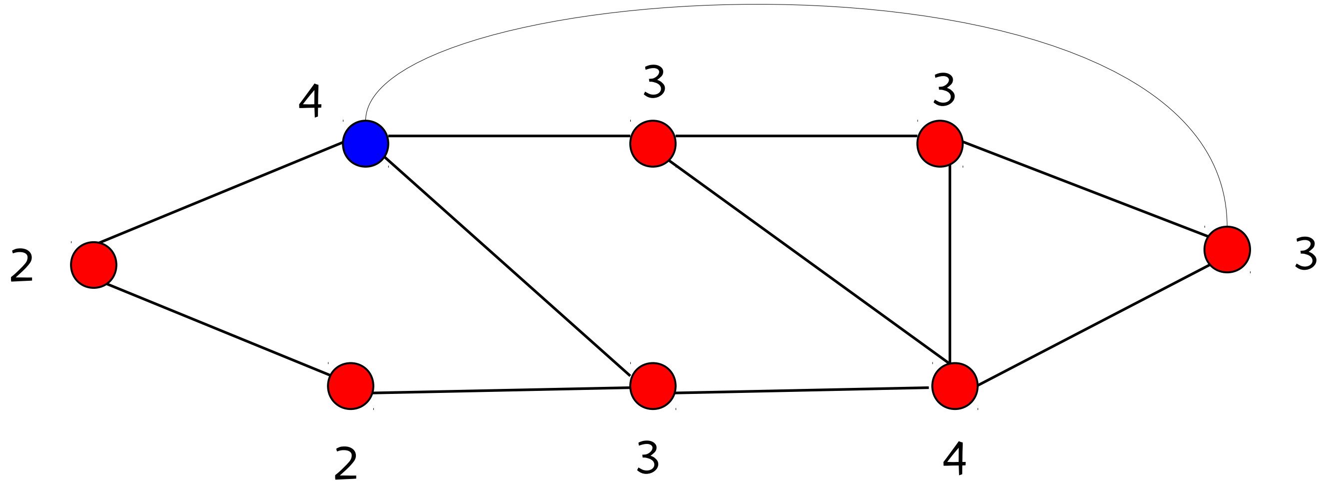
# The greedy algorithm

## Another example: Maximum clique



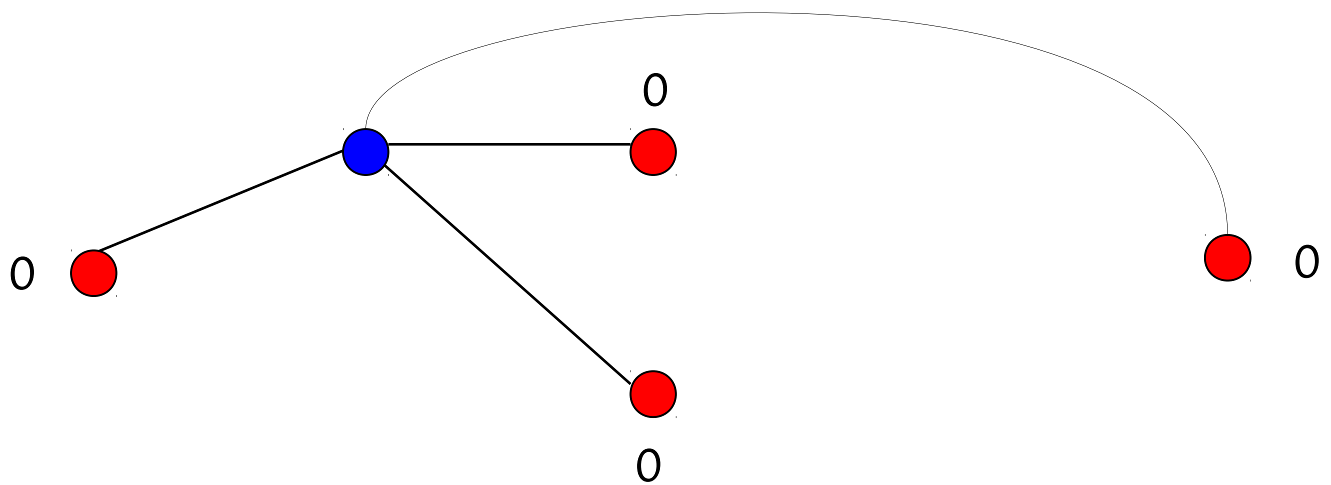
# The greedy algorithm

## Another example: Maximum clique



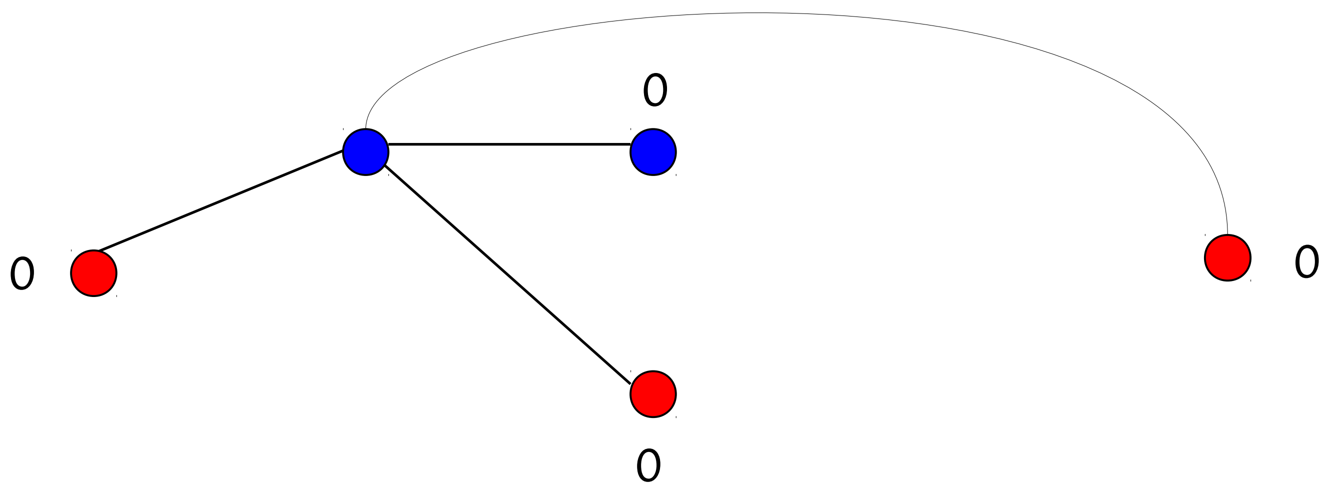
# The greedy algorithm

## Another example: Maximum clique



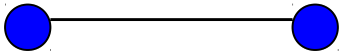
# The greedy algorithm

## Another example: Maximum clique



# The greedy algorithm

## Another example: Maximum clique



sub-optimal
clique

# Semi-greedy heuristic

A semi-greedy heuristic tries to get around convergence to non-global local minima.

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

# Semi-greedy heuristic

Hart & Shogan (1987) propose two mechanisms for building the RCL:

Cardinality based: place k best candidates in RCL

Value based: place all candidates having greedy values better than $\alpha \cdot$best_value in RCL, where $\alpha \in [0,1]$.

Feo & Resende (1989) proposed semi-greedy construction as a basic component of GRASP.

# Hart-Shogan Algorithm

$c^* = \infty$

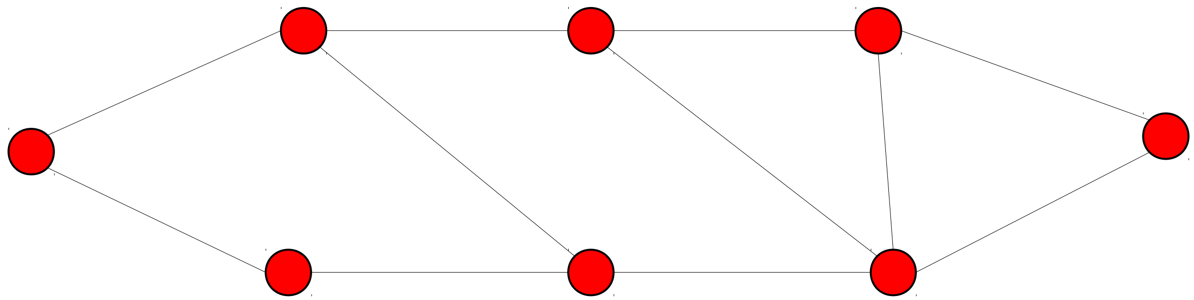
repeat

x = semi_greedy_construction()
if (x is infeasible) then
    x = repair(x)

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

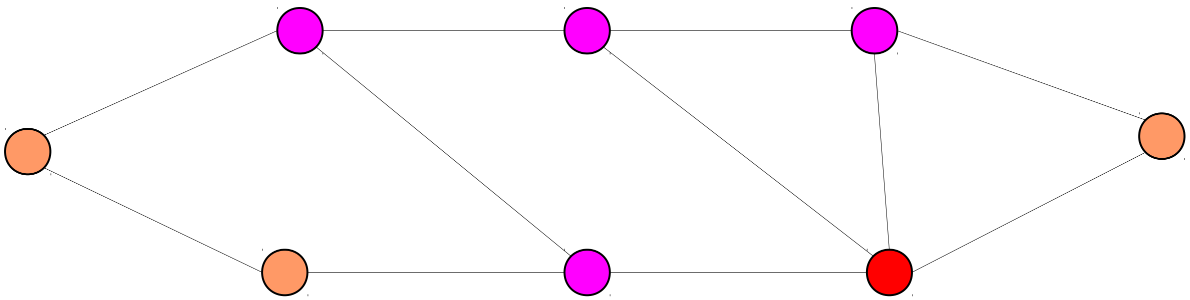
# The semi-greedy algorithm

## Maximum clique example



# The semi-greedy algorithm

## Maximum clique example

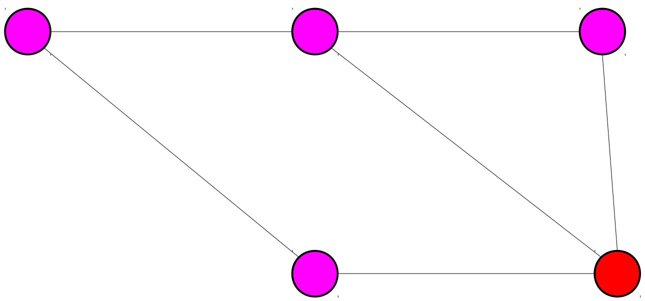


Build clique, one node at a time.

Candidates: nodes adjacent to clique.

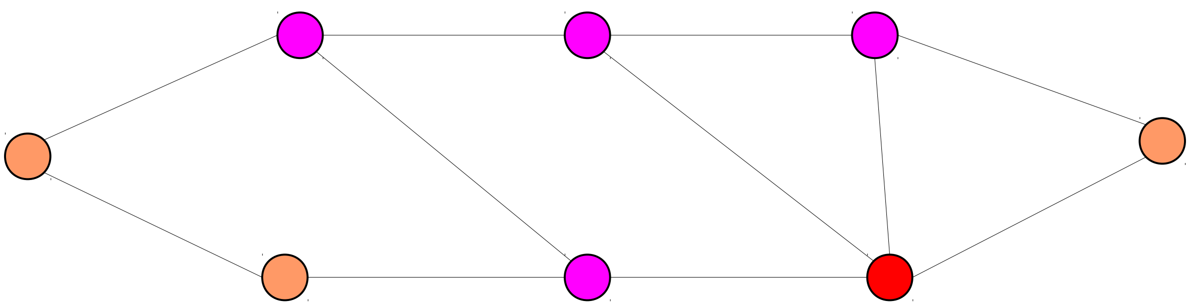
Greedy function: degree with respect to candidate nodes.

RCL =



# The semi-greedy algorithm

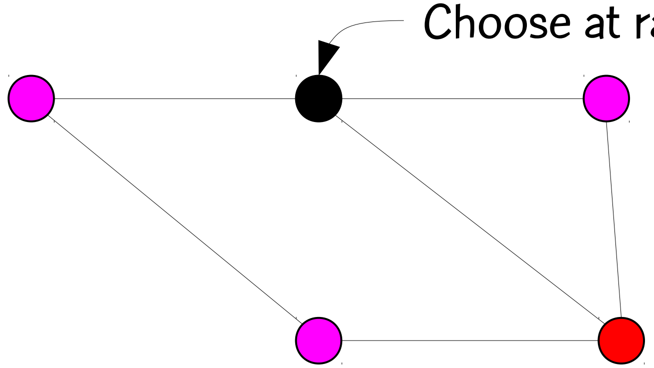
## Maximum clique example



Build clique, one node at a time.

Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

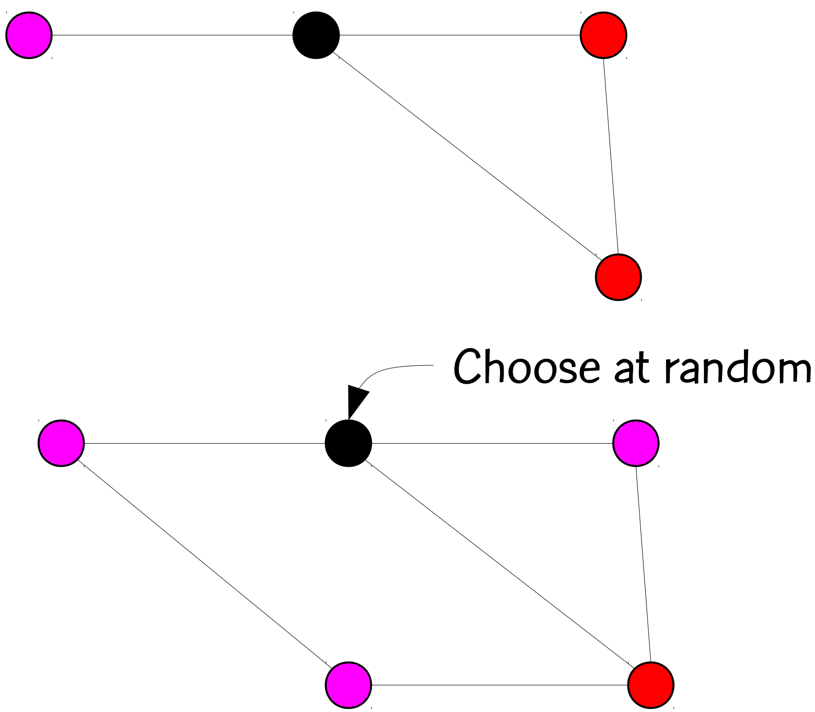



RCL =

Semi-greedy iteration 1

# The semi-greedy algorithm

## Maximum clique example




RCL =

Build clique, one node at a time.

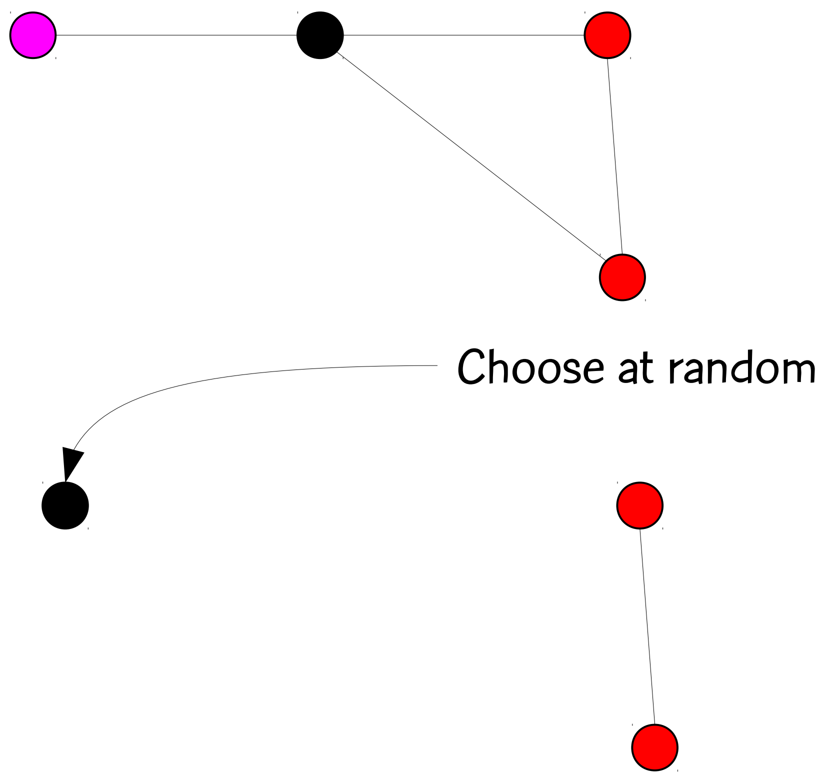
Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 1

# The semi-greedy algorithm

## Maximum clique example




RCL =

Build clique, one node at a time.

Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 1

# The semi-greedy algorithm

## Maximum clique example

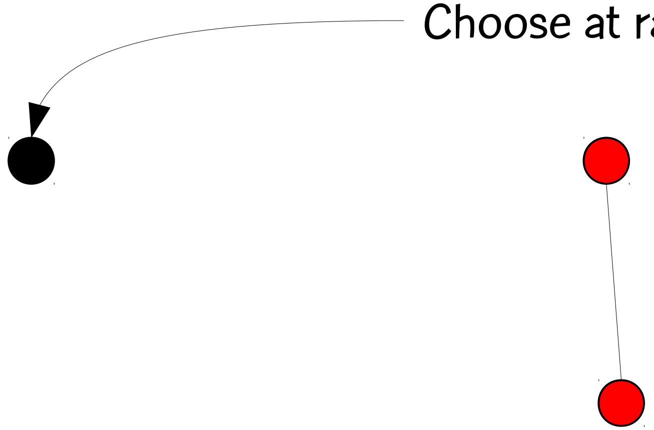
Clique of size 2

Build clique, one node at a time.

Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Choose at random

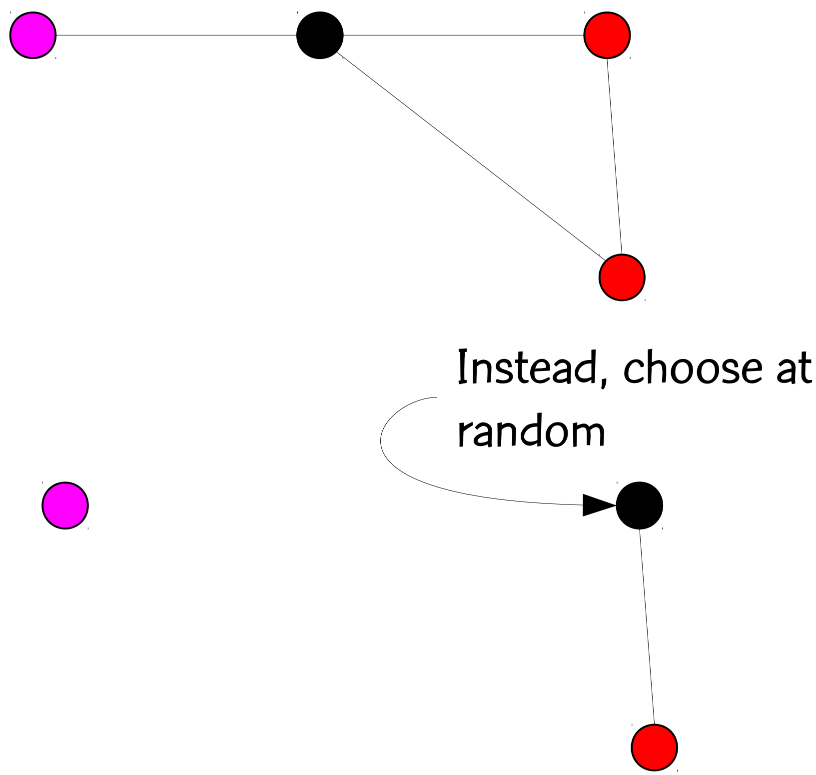


RCL =

Semi-greedy iteration 1

# The semi-greedy algorithm

## Maximum clique example




Build clique, one node at a time.

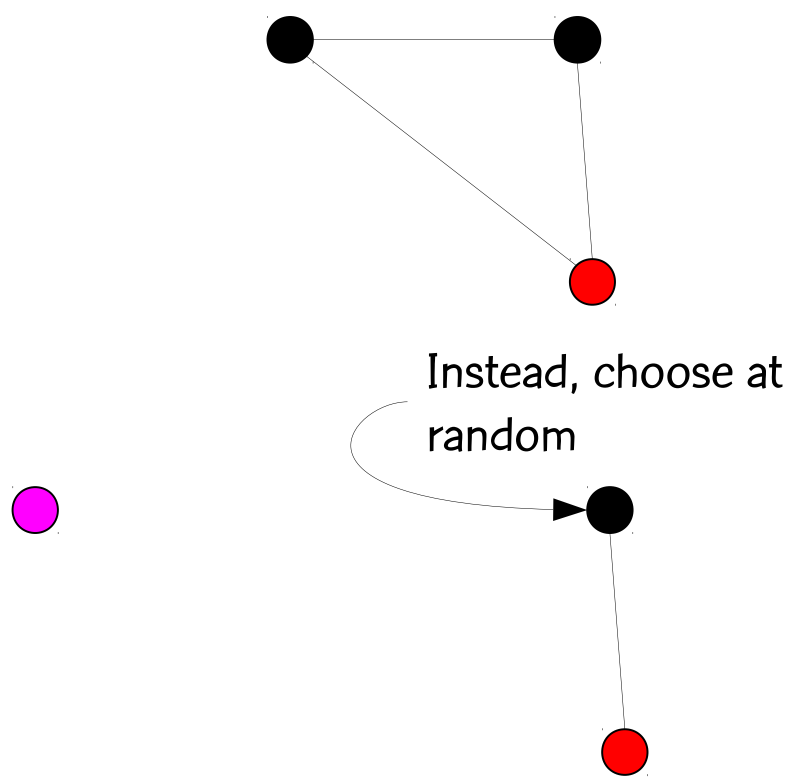
Candidates:  nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

RCL =

Semi-greedy iteration 2

# The semi-greedy algorithm
## Maximum clique example




Build clique, one node at a time.

Candidates: nodes adjacent to clique.

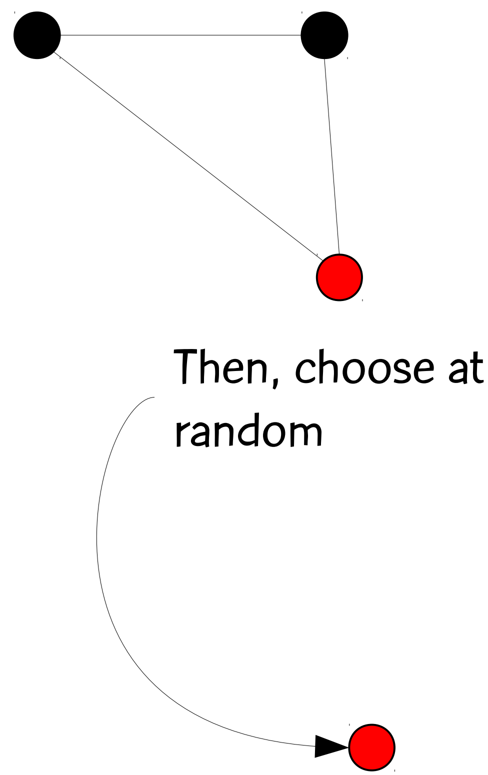
Greedy function: degree with respect to candidate nodes.

RCL =

Semi-greedy iteration 2

# The semi-greedy algorithm

## Maximum clique example




Build clique, one node at a time.

Candidates: nodes adjacent to clique.

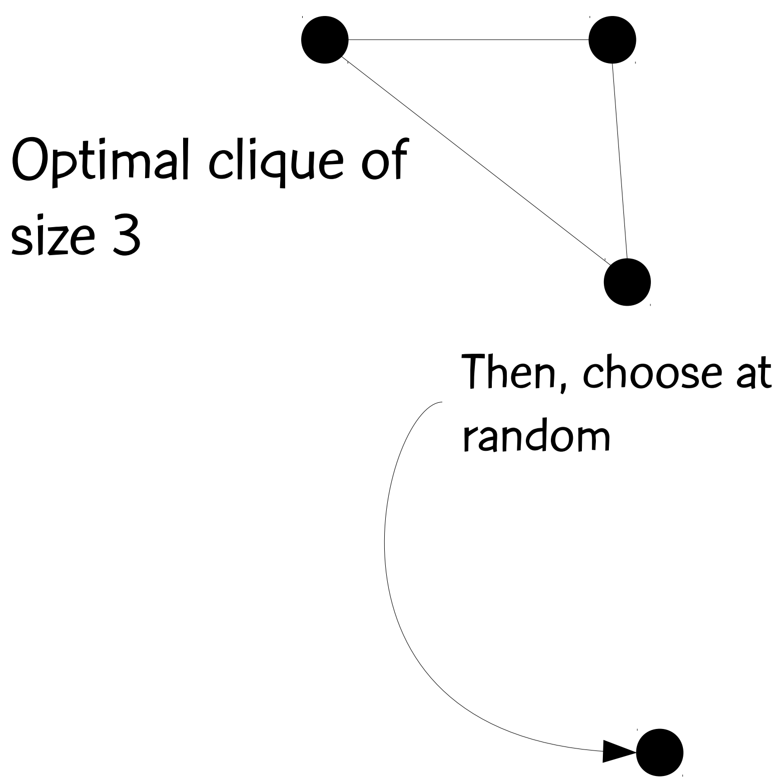
Greedy function: degree with respect to candidate nodes.

RCL =

Semi-greedy iteration 2

# The semi-greedy algorithm

## Maximum clique example




RCL =

Build clique, one node at a time.

Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 2

# GRASP

# GRASP: Basic algorithm

```
c* = ∞
repeat
    x = semi_greedy_construction()
    if (x is infeasible) then
            x = repair(x)
    x = local_search(x)
    if f(x) < c* then
        x* = x
        c* = f(x*)
    endif
```

Semi-greediness is more general in GRASP

# GRASP: Basic algorithm

## Construction phase: greediness + randomization

Builds a feasible solution combining greediness and randomization

## Local search: search in the current neighborhood until a local optimum is found

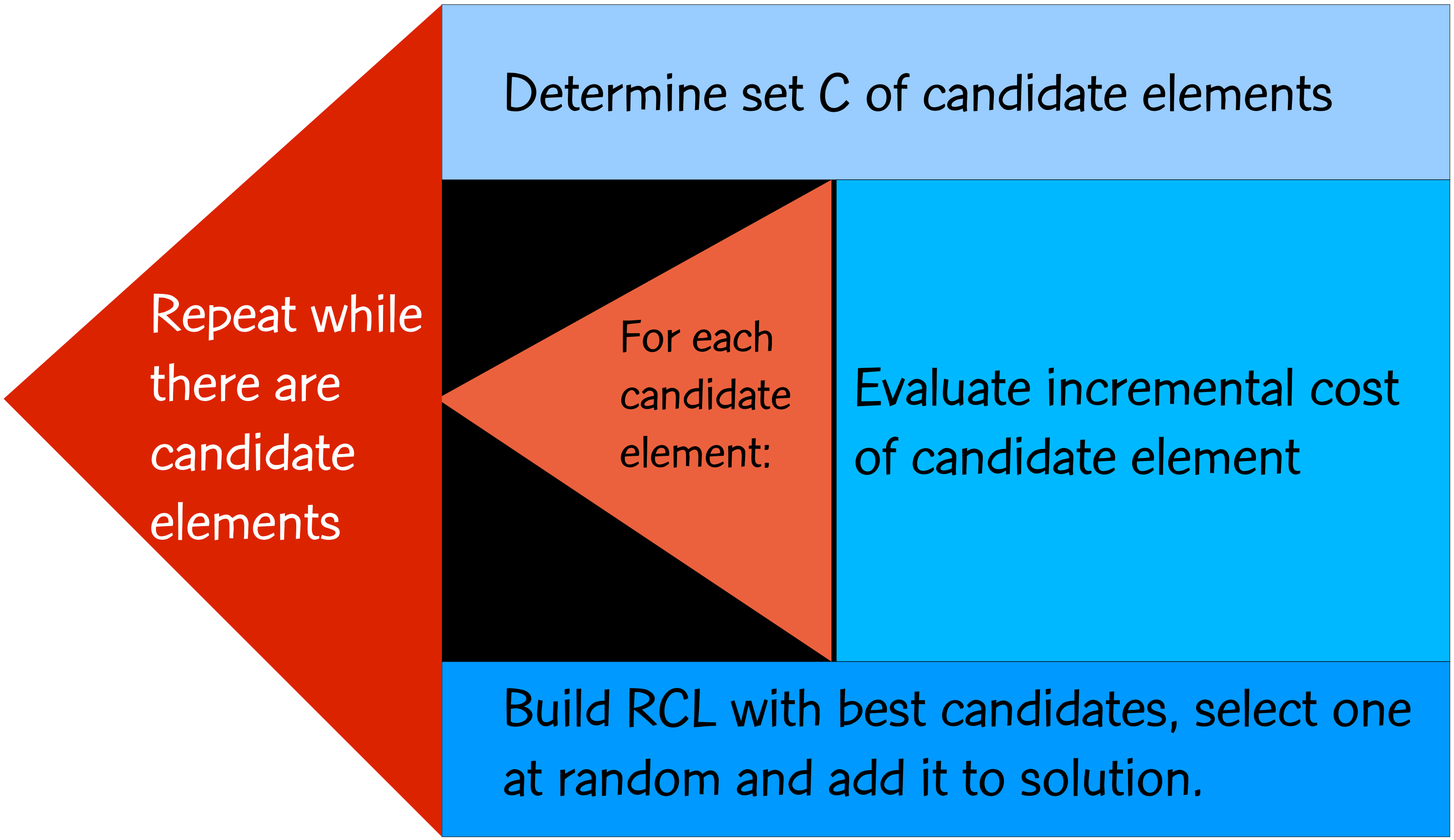
Solutions generated by the construction procedure are not necessarily optimal:

Effectiveness of local search depends on: neighborhood structure, search strategy, and fast evaluation of neighbors, but also on the construction procedure itself.

# GRASP Construction

# Construction phase: RCL based

restricted candidate list



# Construction phase: RCL based

Minimization problem

Basic construction procedure:

Greedy function $c(e)$: incremental cost associated with the incorporation of element $e$ into the current partial solution under construction

$c^{min}$ (resp. $c^{max}$): smallest (resp. largest) incremental cost

RCL made up by the elements with the smallest incremental costs.

# Construction phase

Cardinality-based construction:

p elements with the smallest incremental costs

Quality-based construction:

Parameter $\alpha$ defines the quality of the elements in RCL.

RCL contains elements with incremental cost

$c^{min} \leq c(e) \leq c^{min} + \alpha\, (c^{max} - c^{min})$

$\alpha = 0$ : pure greedy construction

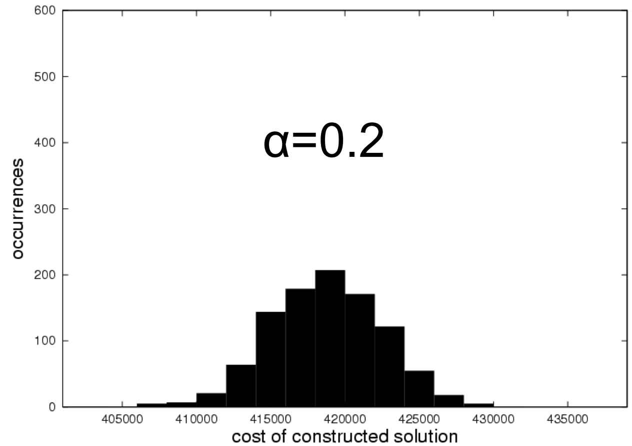
$\alpha = 1$ : pure randomized construction

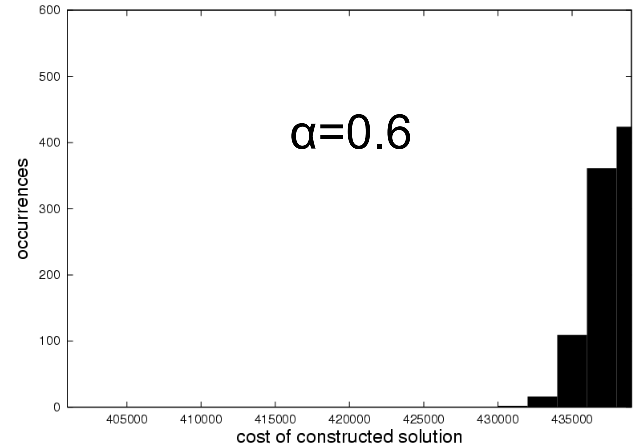
Select at random from RCL using uniform probability distribution



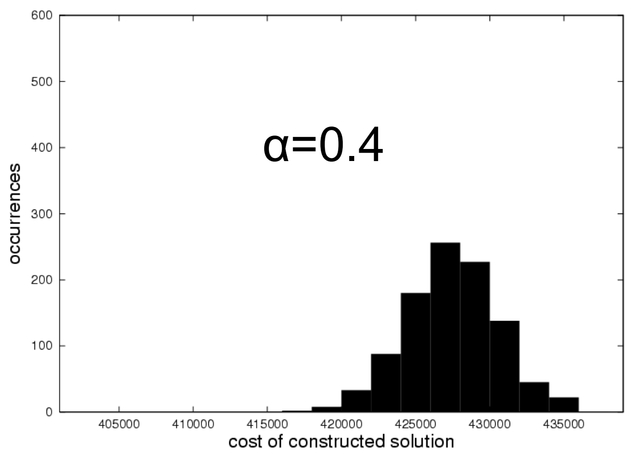
at&t Your world. Delivered.

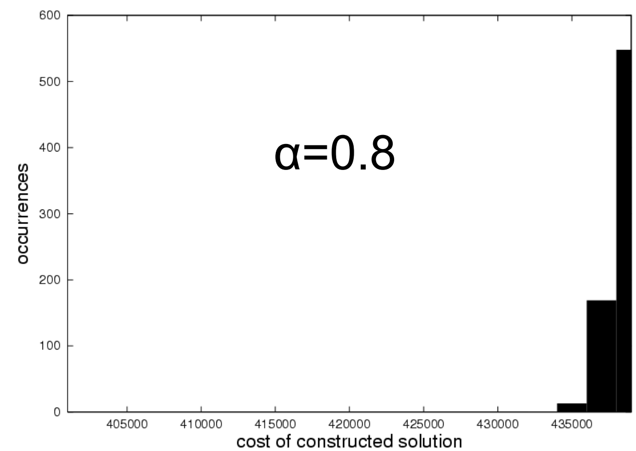
# Illustrative results: RCL parameter



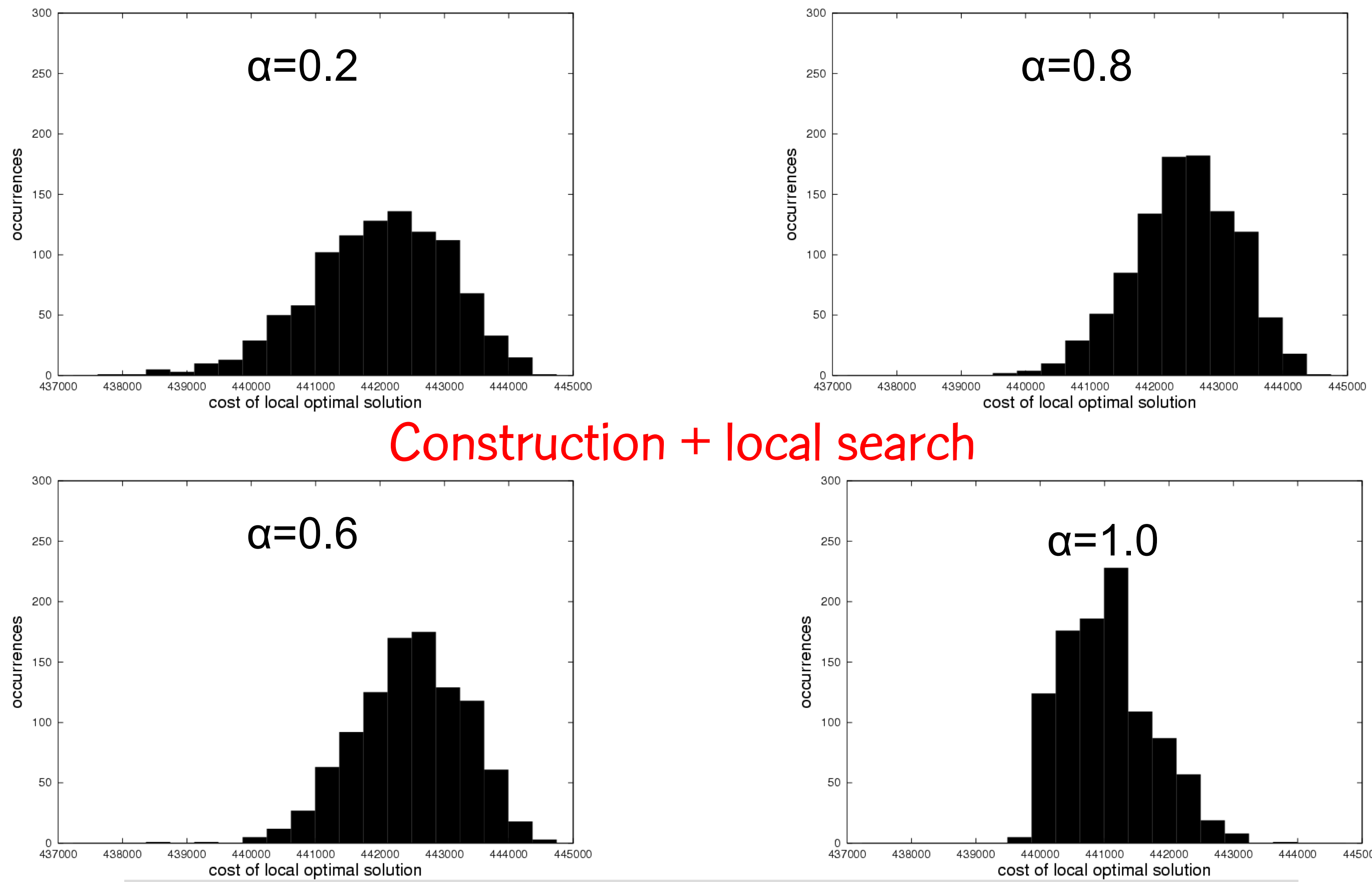

Construction phase only




weighted MAX-SAT instance, 1000 GRASP iterations

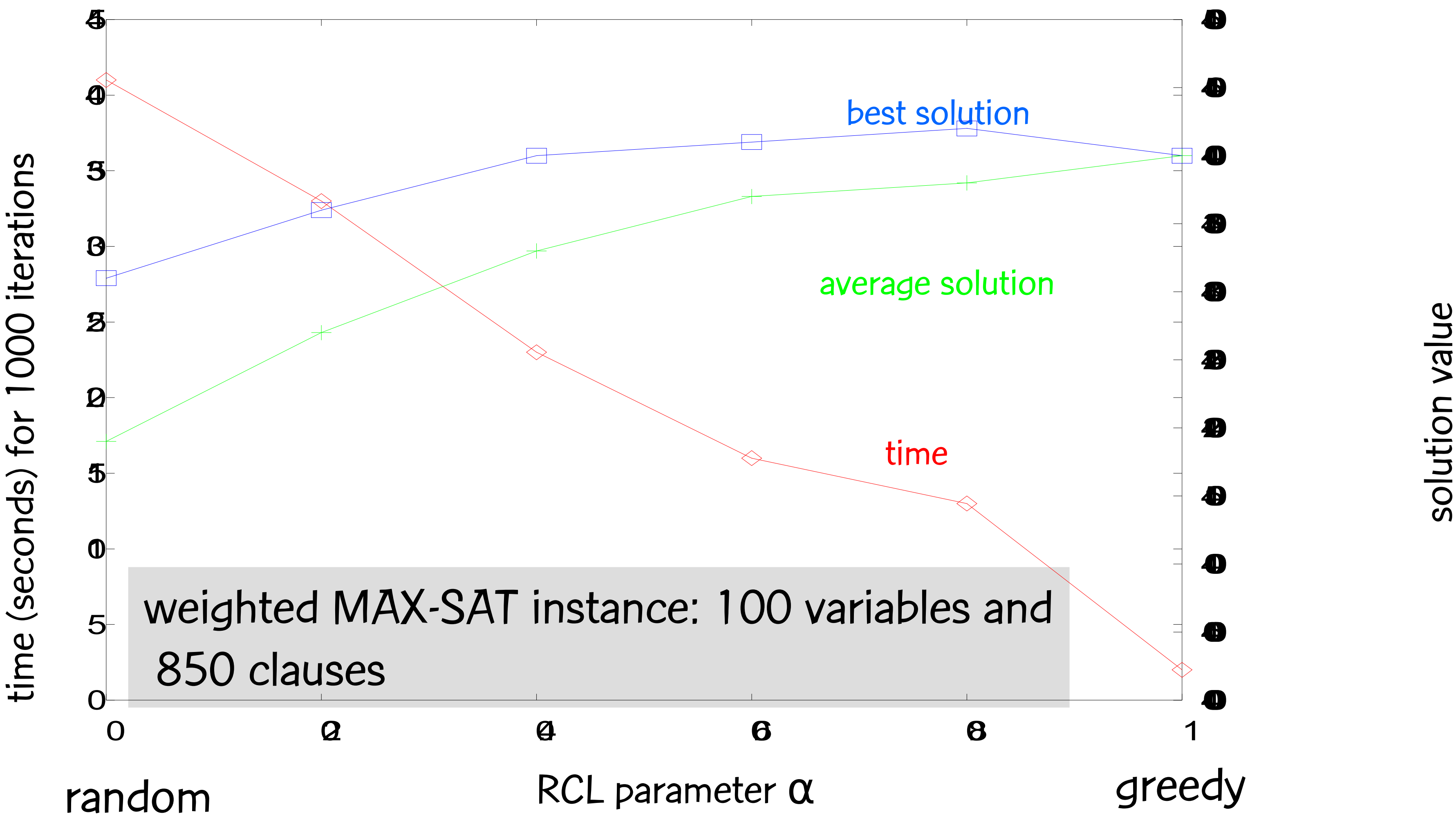
# Illustrative results: RCL parameter




Construction + local search

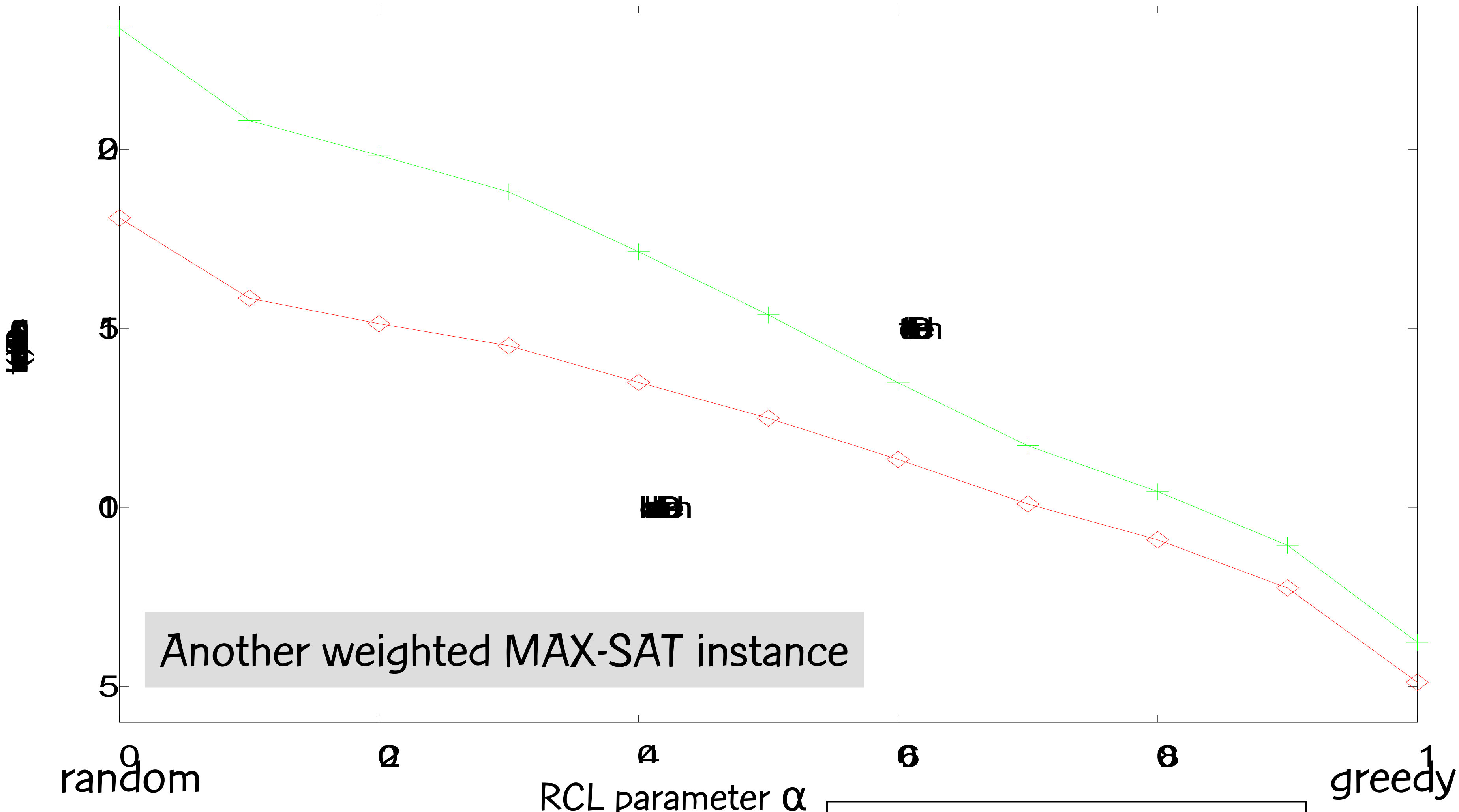
weighted MAX-SAT instance, 1000 GRASP iterations

# Illustrative results: RCL parameter



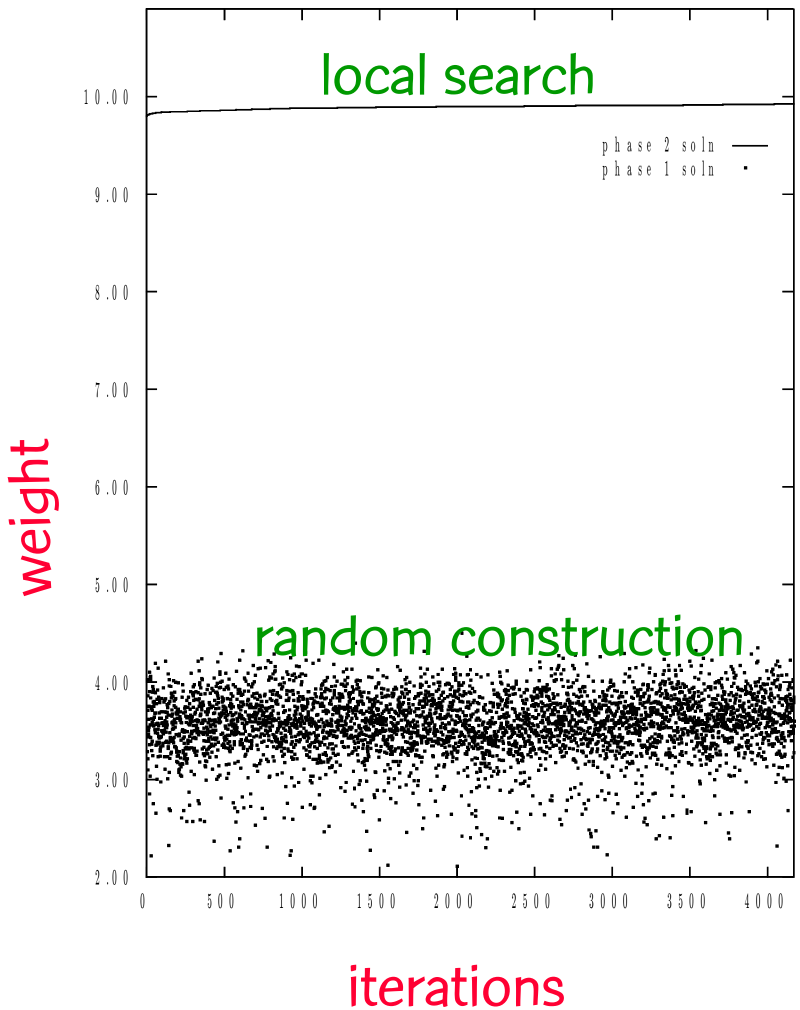

SGI Challenge 196 MHz

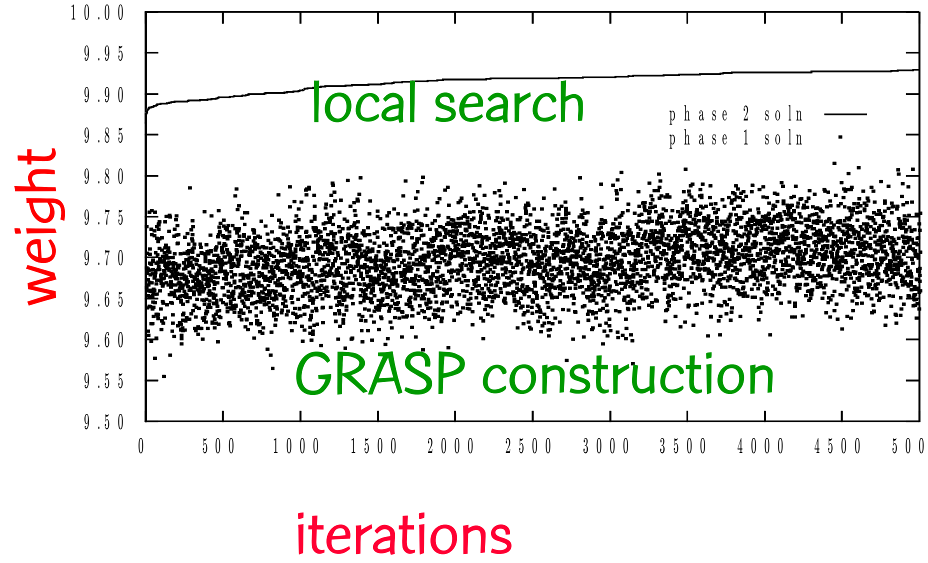
# Illustrative results: RCL parameter




SGI Challenge 196 MHz

# GRASP: Basic algorithm



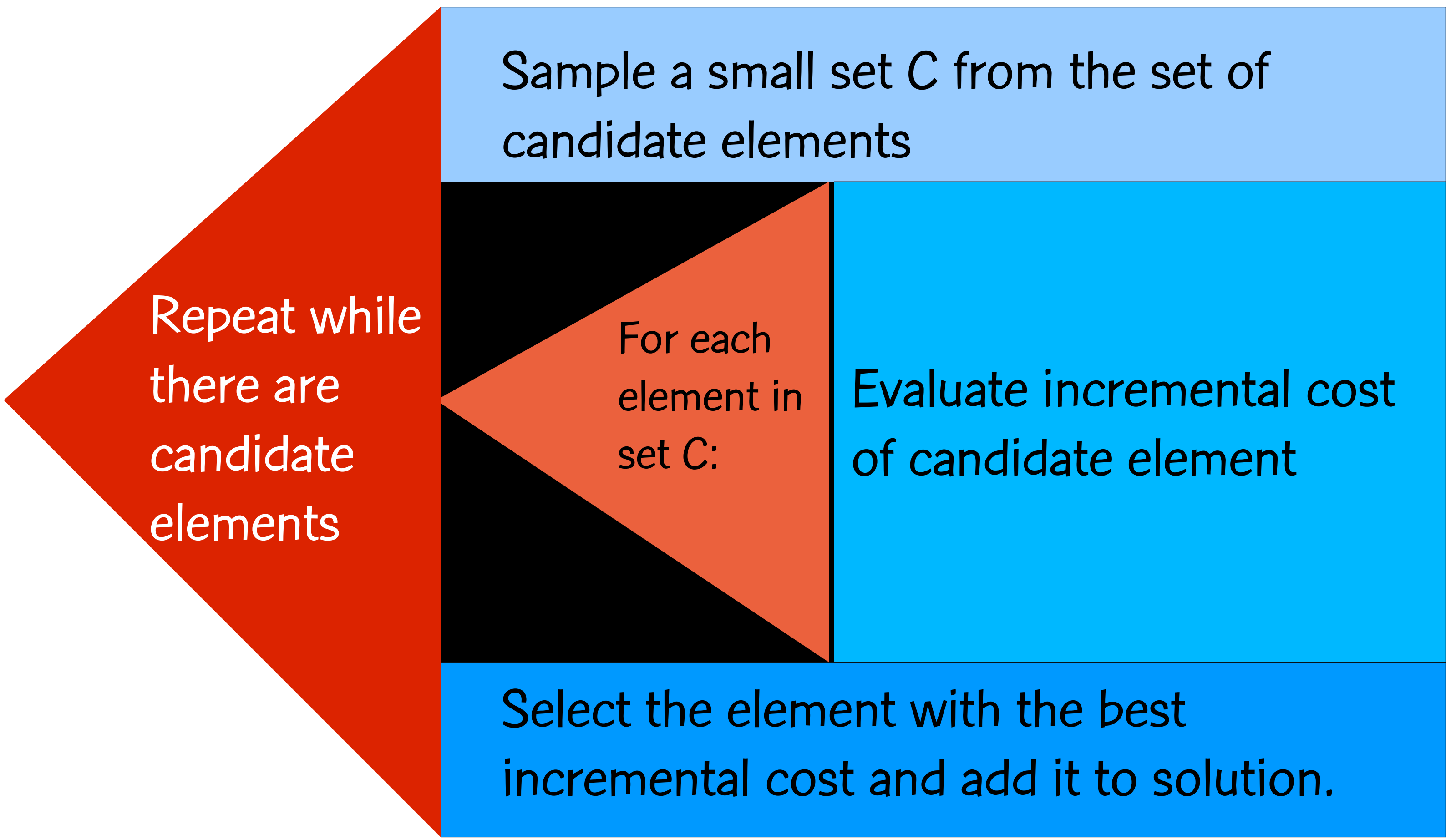

Effectiveness of greedy randomized vs purely randomized construction:

Application: modem placement
max weighted covering problem
maximization problem: $\alpha = 0.85$

# Hybrid construction schemes

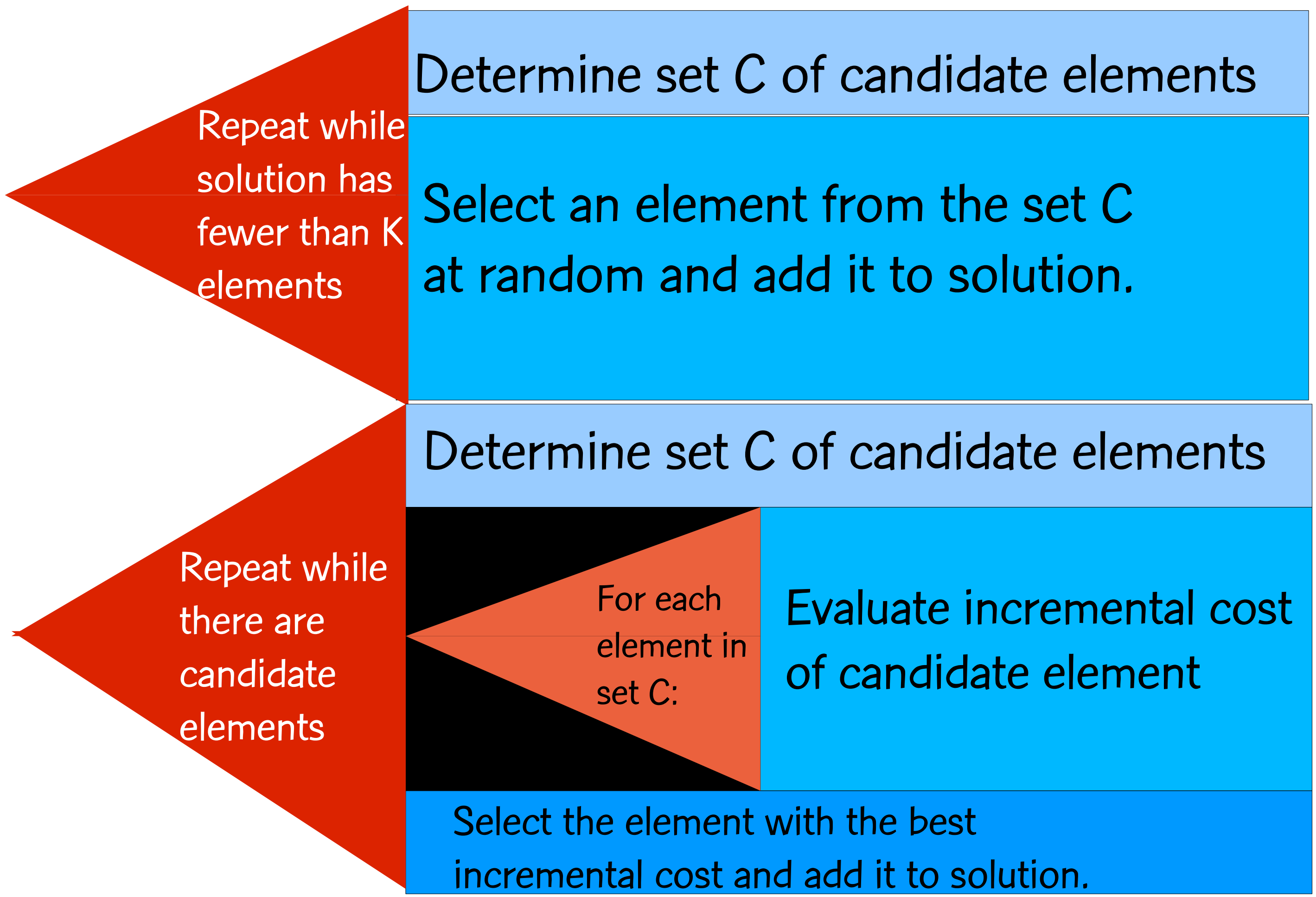
# Construction phase: sampled greedy

[Resende & Werneck, 2004]

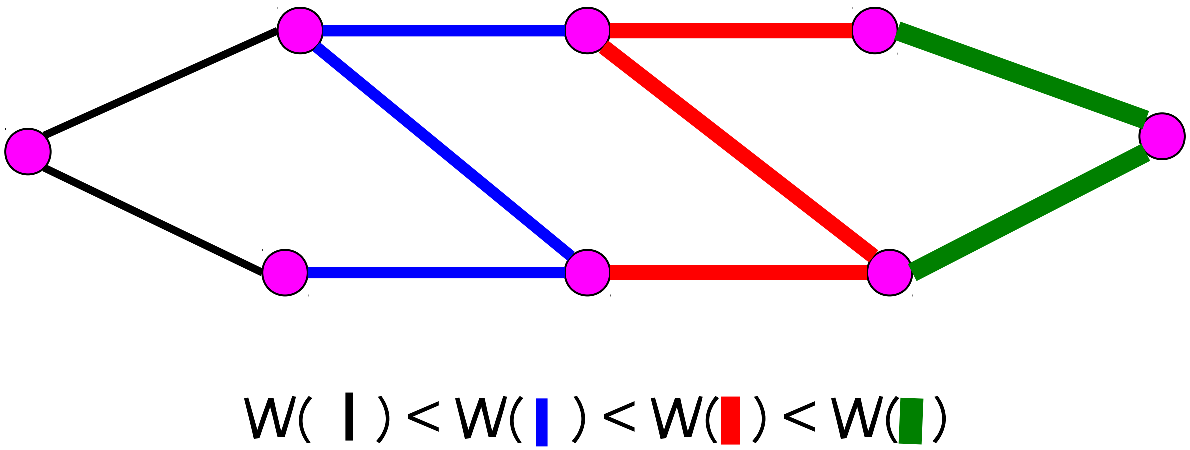


# Construction phase: random+greedy

[Resende & Werneck, 2004]

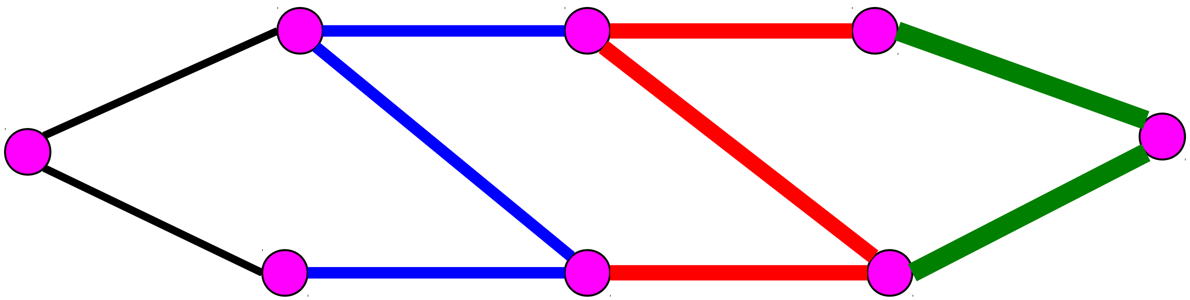


# Construction with cost perturbation



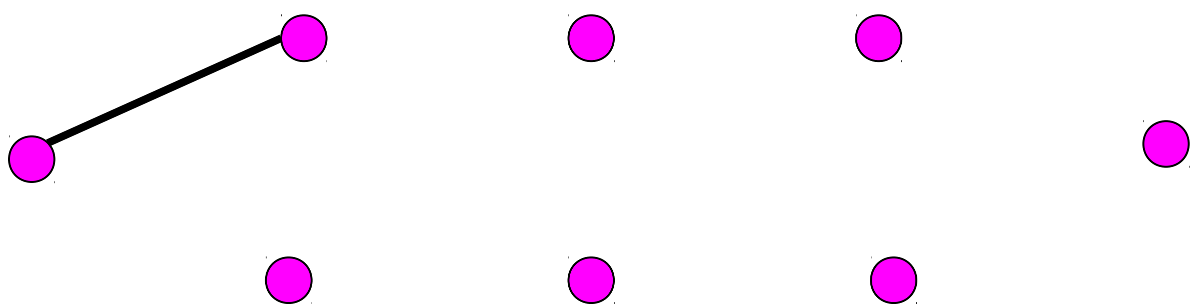

Perturb with costs increasing from top to bottom.

# Construction with cost perturbation

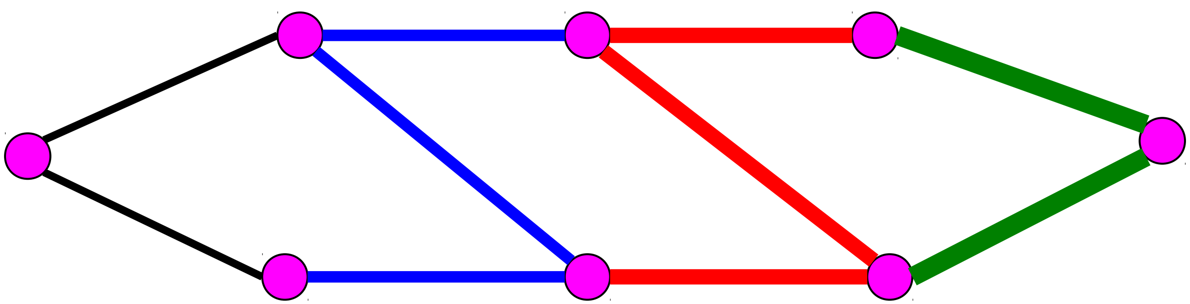


Perturb with costs increasing from top to bottom.

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

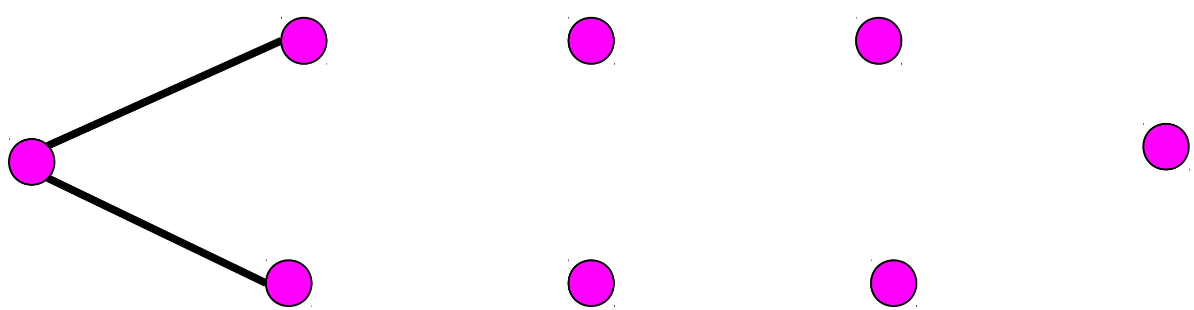


# Construction with cost perturbation

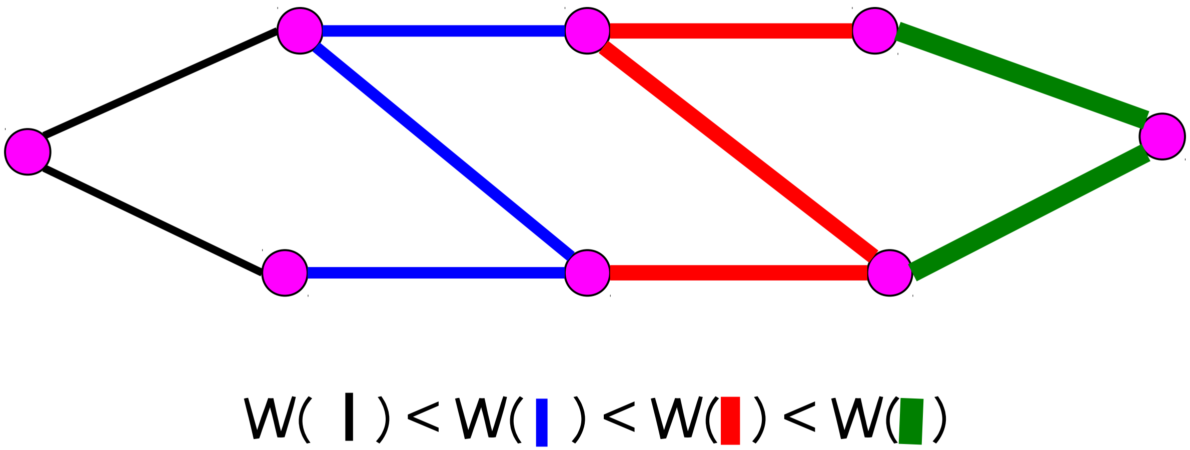


Perturb with costs increasing from top to bottom.

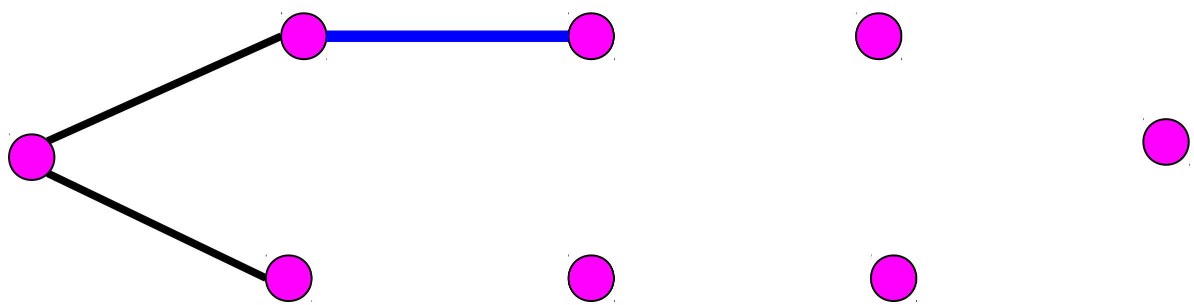
W( ) < W( ) < W( ) < W( )



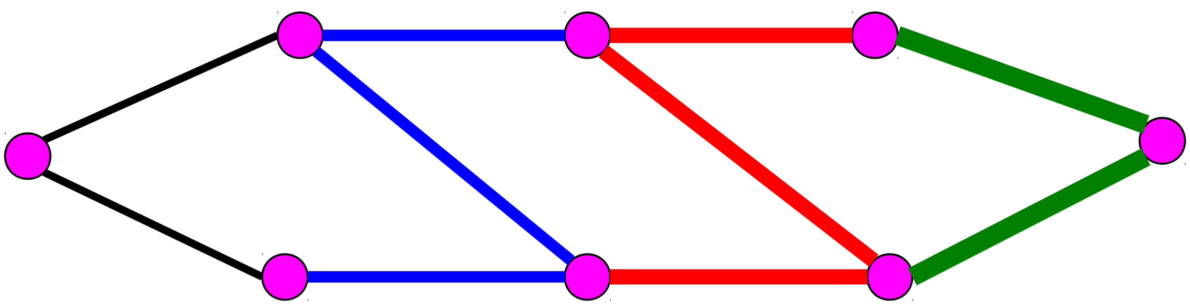
# Construction with cost perturbation




Perturb with costs increasing from top to bottom.

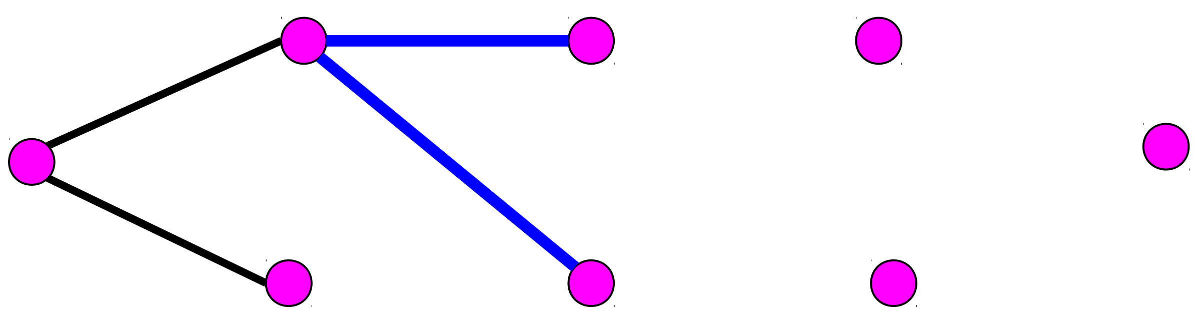


# Construction with cost perturbation

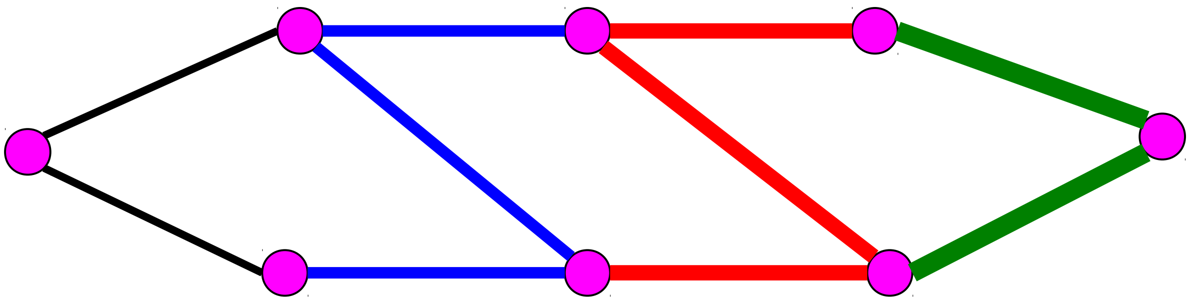


Perturb with costs increasing from top to bottom.

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

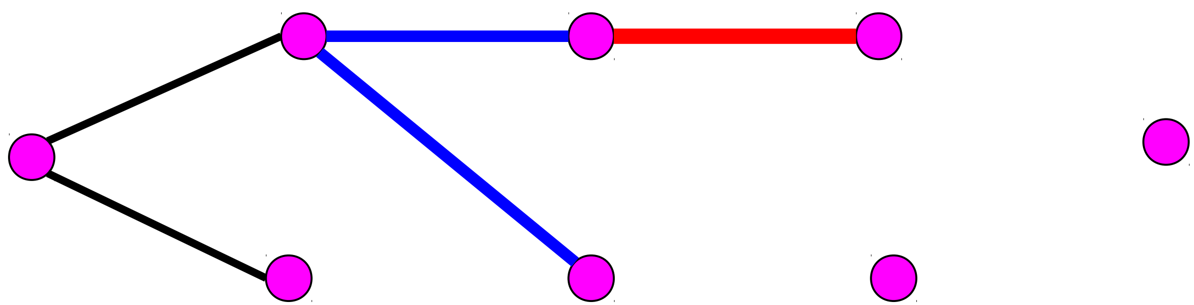


# Construction with cost perturbation

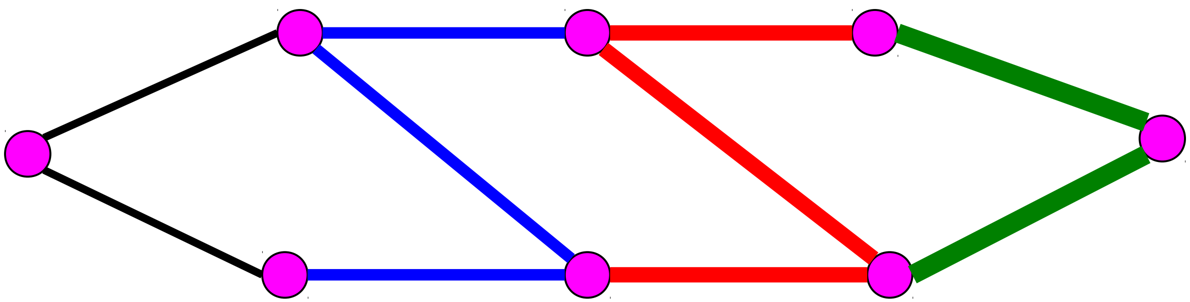


Perturb with costs increasing from top to bottom.

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

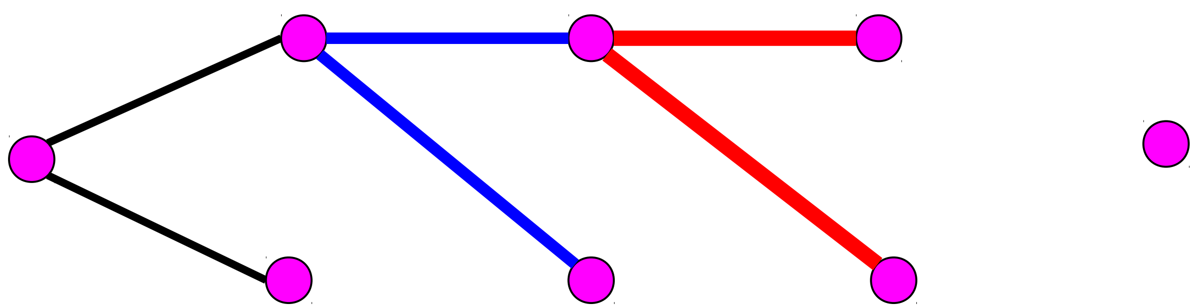


# Construction with cost perturbation

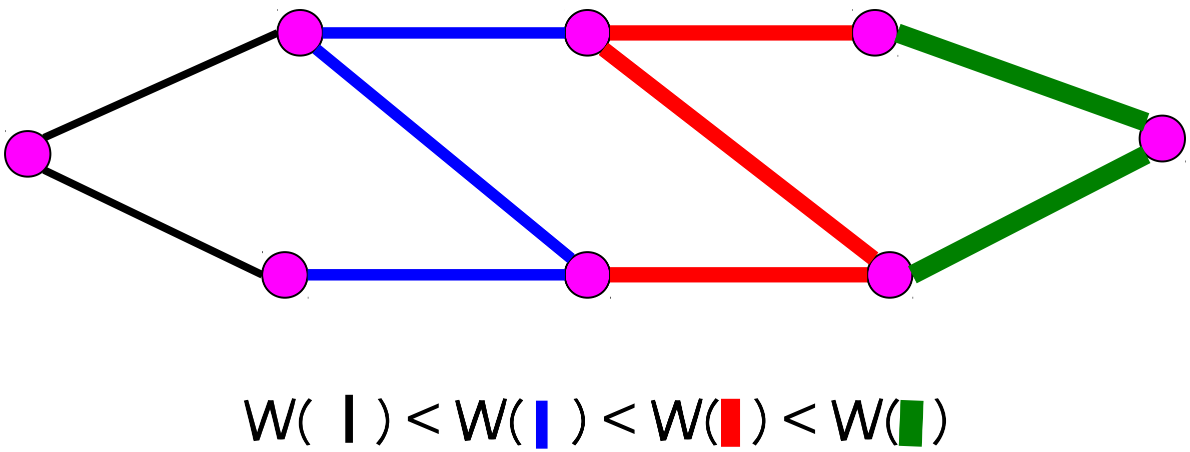


Perturb with costs increasing from top to bottom.

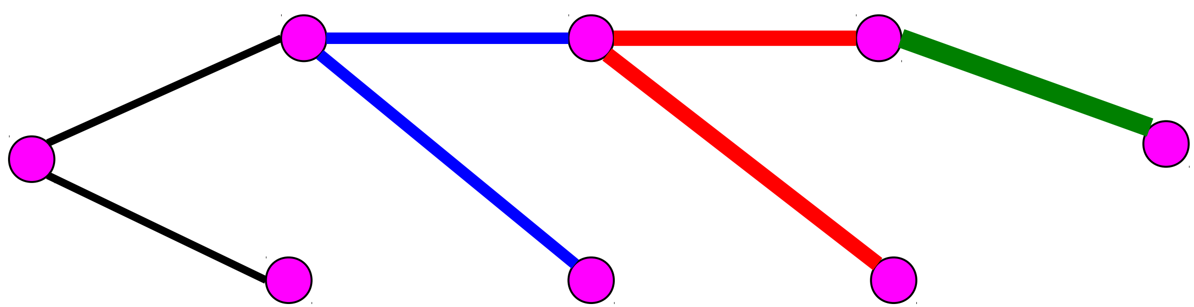
W( ) < W( ) < W( ) < W( )



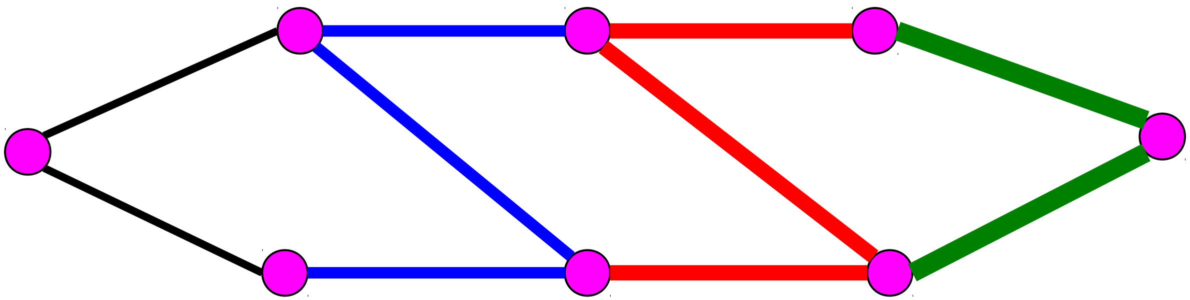
# Construction with cost perturbation




Perturb with costs increasing from top to bottom.



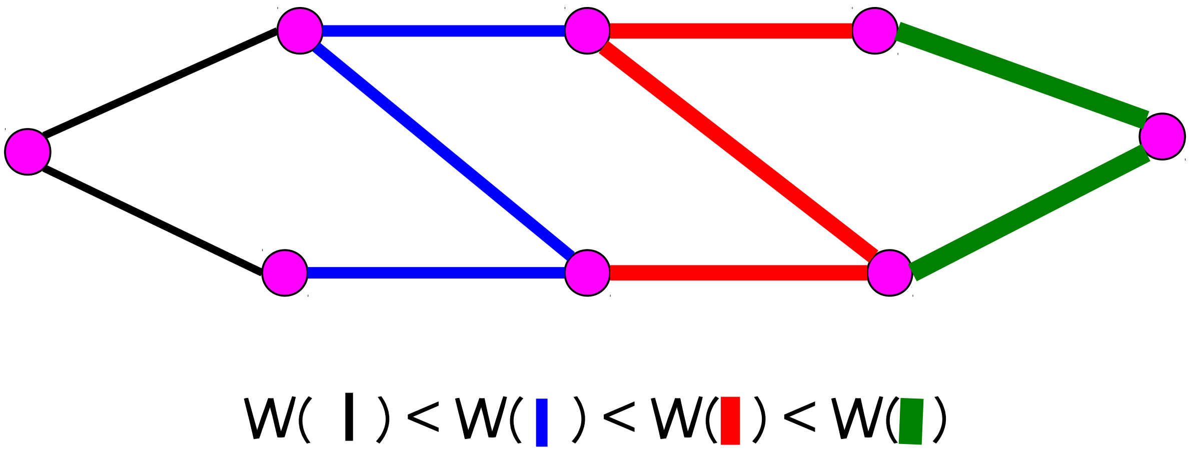
# Construction with cost perturbation



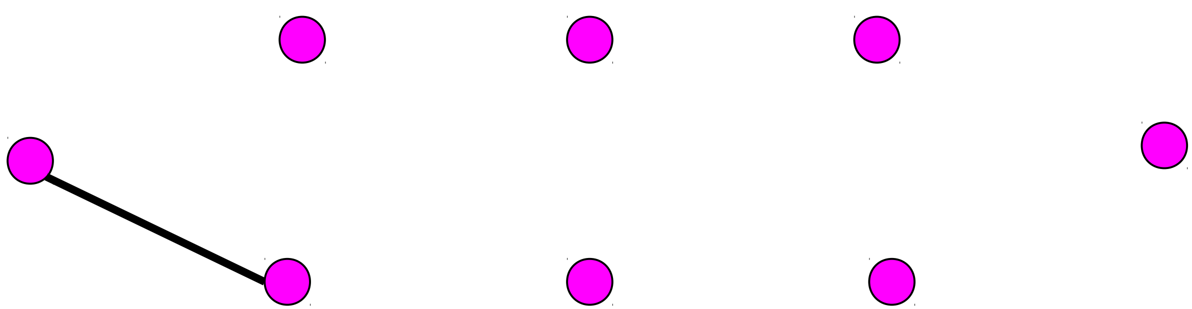
W( ) < W( ) < W( ) < W( )

Perturb with costs increasing from bottom to top.

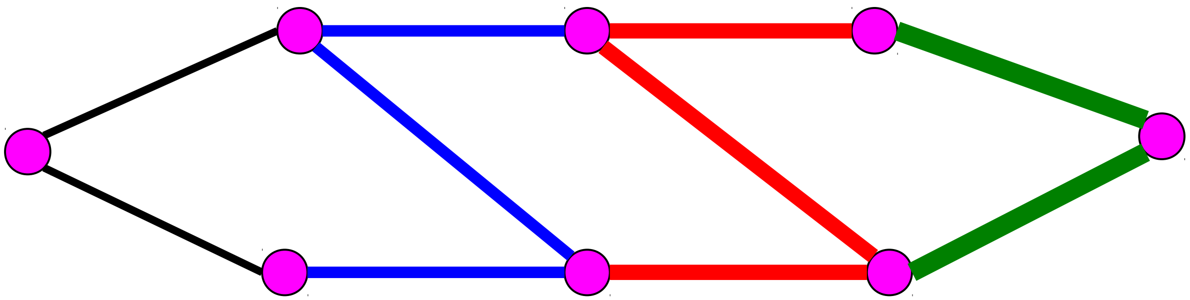
# Construction with cost perturbation




Perturb with costs increasing from bottom to top.

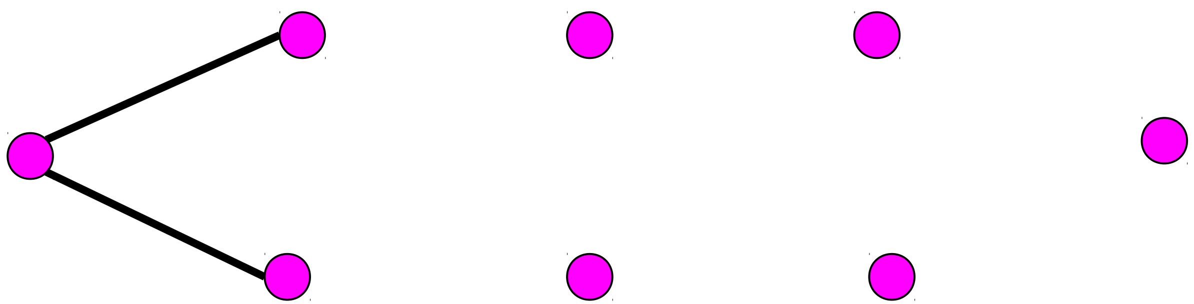


# Construction with cost perturbation

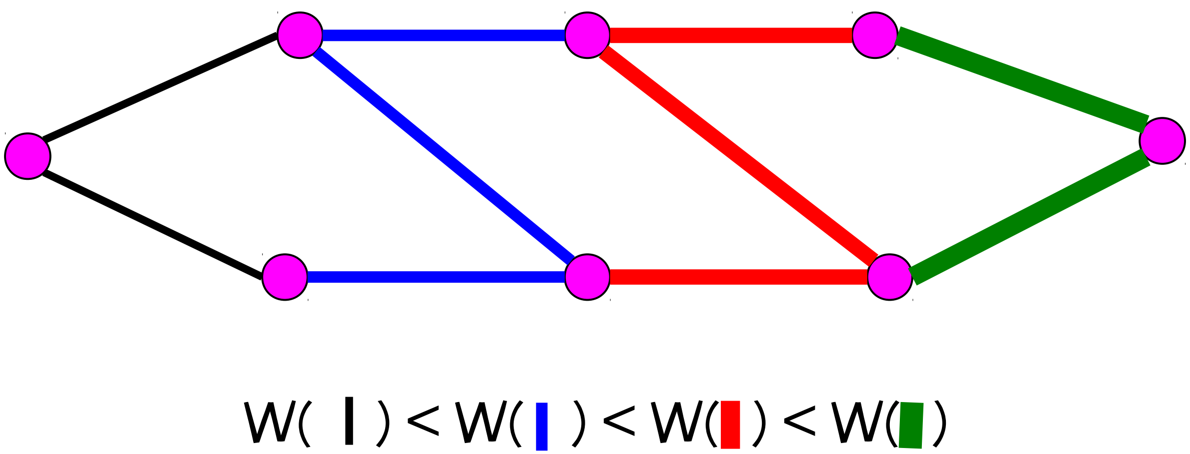


Perturb with costs increasing from bottom to top.

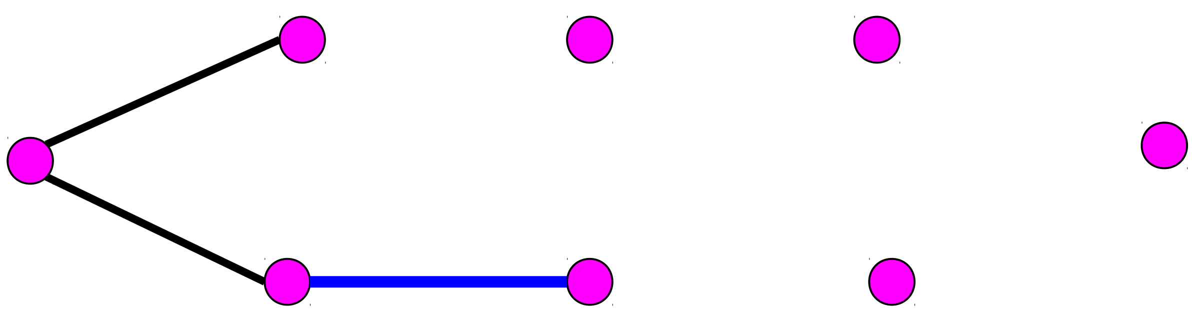
W( | ) < W( | ) < W( | ) < W( | )



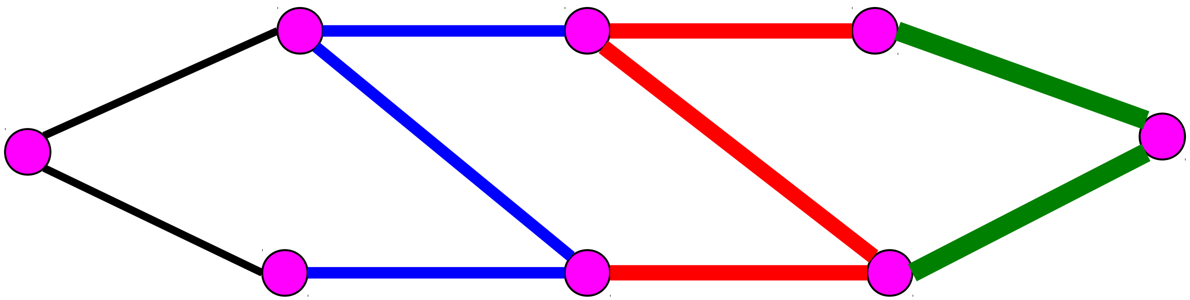
# Construction with cost perturbation




Perturb with costs increasing from bottom to top.

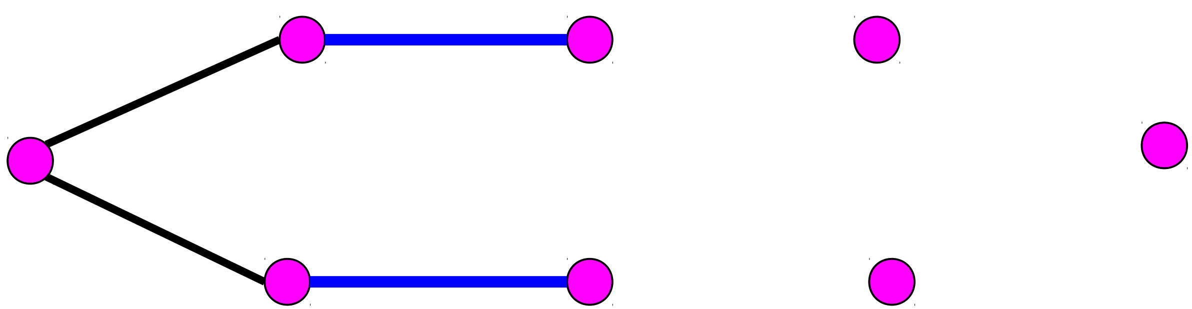


# Construction with cost perturbation

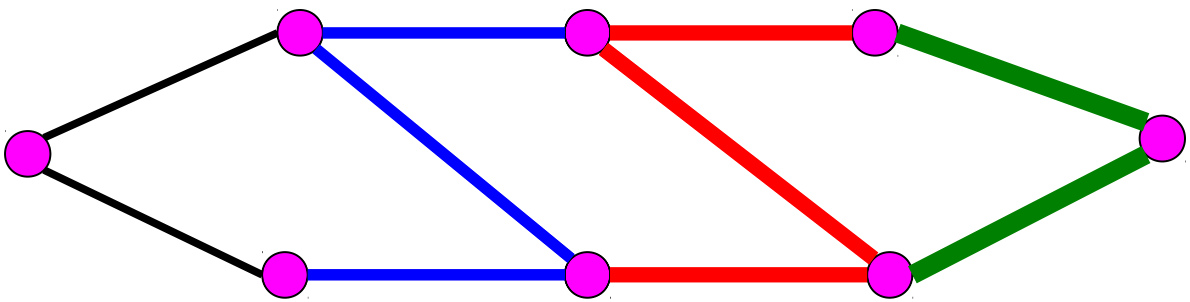


Perturb with costs increasing from bottom to top.

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

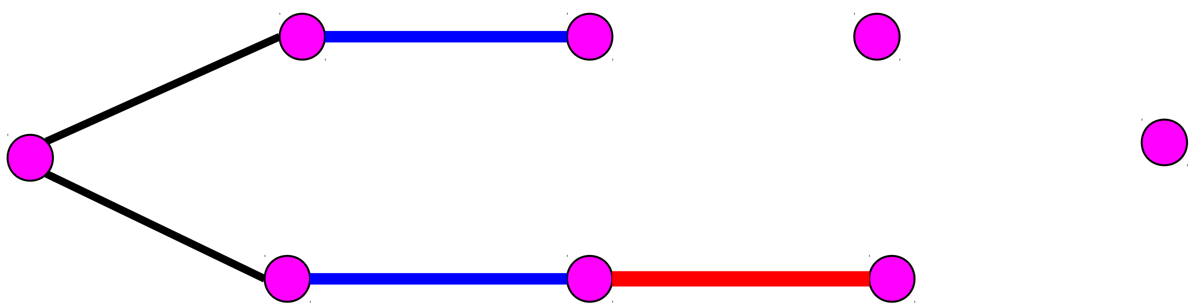


# Construction with cost perturbation

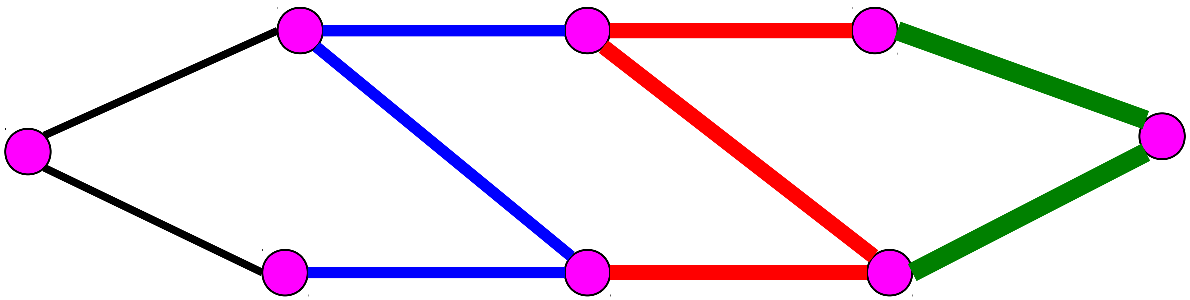


Perturb with costs increasing from bottom to top.

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

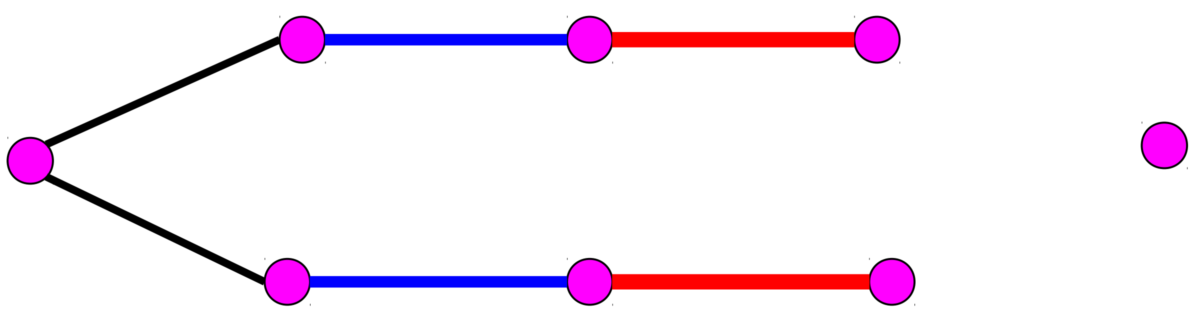


# Construction with cost perturbation

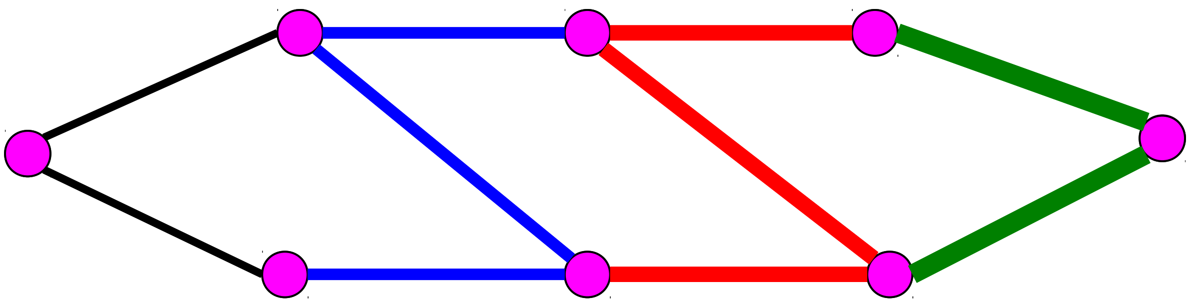


Perturb with costs increasing from bottom to top.

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

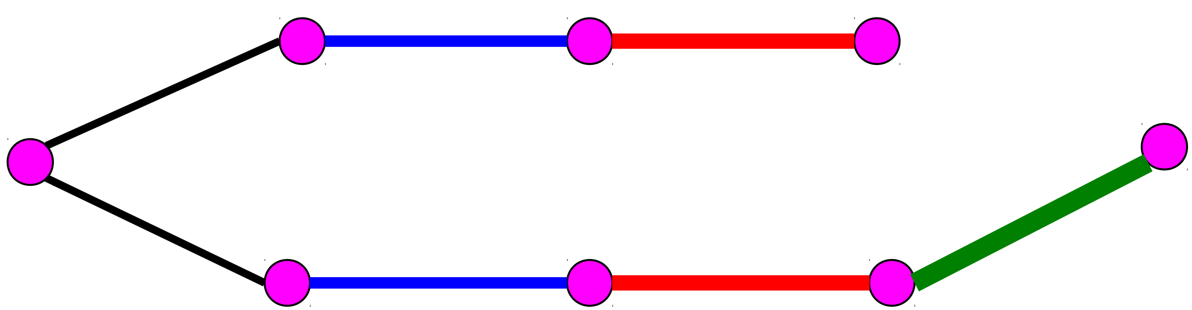


# Construction with cost perturbation

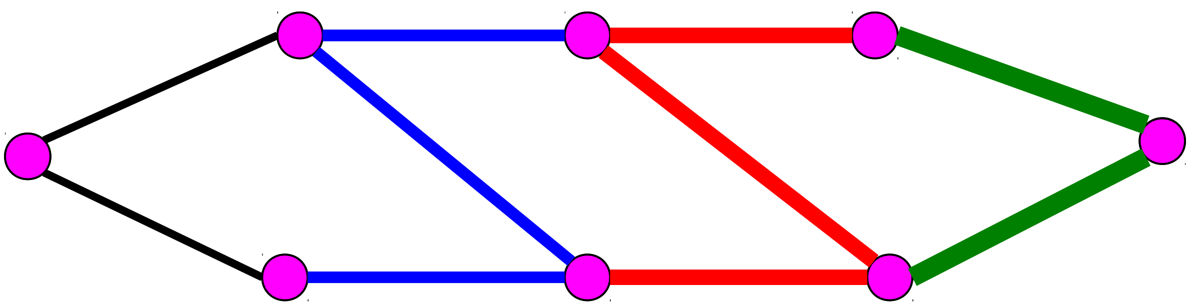


Perturb with costs increasing from bottom to top.

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

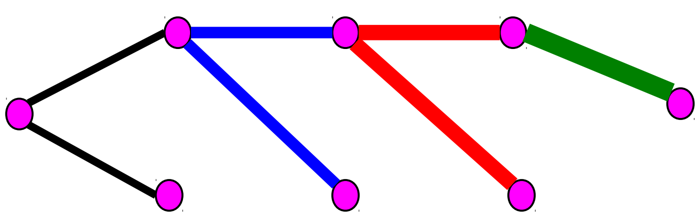


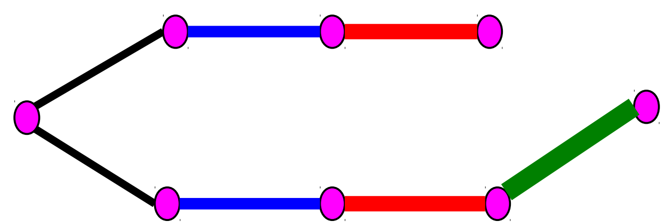
# Construction with cost perturbation



Greedy heuristic generates two different spanning trees.

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





# Reactive GRASP

Prais & Ribeiro (2000)

## When building RCL, what $\alpha$ to use?

Fix a some value $0 \le \alpha \le 1$

Choose $\alpha$ at random (uniformly) at each GRASP iteration.

## Another approach reacts to search ...

At each GRASP iteration, a value of the RCL parameter $\alpha$ is chosen from a discrete set of values $[\alpha_1, \alpha_2, \ldots, \alpha_m]$.

The probability that $\alpha_k$ is selected is $p_k$.

Reactive GRASP: adaptively changes the probabilities $[p_1, p_2, \ldots, p_m]$ to favor values of $\alpha$ that produce good solutions.

# Reactive GRASP

Prais & Ribeiro (2000)

Reactive GRASP for minimization ...

Initially $p_k = 1/m$, for $k = 1,\ldots,m$. ($\alpha$'s are selected uniformly at random)

Define

$F(S^*)$ be the best solution so far

$A_k$ be the average value of the solutions obtained with $\alpha_k$

Every $N_\alpha$ GRASP iterations, compute

$q_k = F(S^*) / A_k$, for $k = 1,\ldots,m$

$p_k = q_k / \text{sum}(q_i \mid i = 1,\ldots,m)$

# Reactive GRASP

Prais & Ribeiro (2000)

Reactive GRASP for minimization ...

Initially $p_k = 1/m$, for $k = 1,\ldots,m$. ($\alpha$'s are selected uniformly at random)

Define

$F(S^*)$ be the best solution so far

$A_k$ be the average value of the solutions obtained with $\alpha_k$

Every $N_\alpha$ GRASP iterations, compute

$q_k = F(S^*) / A_k$, for $k = 1,\ldots,m$

$p_k = q_k / \mathrm{sum}(q_i \mid i = 1,\ldots,m)$

> The more suitable is $\alpha_k$, the larger is $q_k$, and consequently $p_k$, making $\alpha_k$ more likely to chosen.

# Hybrid local search in GRASP

# Local search within GRASP

Local search is usually implemented in GRASP as:

$x = x^0$;

while ( there exists $y \in N(x)$ | $f(y) < f(x)$ ) do

$x = y$; // y is first improving solution found in $N(x)$

end while

return $x$;

# Local search within GRASP

Local search is usually implemented in GRASP as:

$x = x^0;$

while ( there exists $y \in N(x) \mid f(y) < f(x)$ ) do

  $x = y$; // y is first improving solution found in N(x)

end while

return x;

first improving



at&t Your world. Delivered.

# Local search within GRASP

Another way to implement local search in GRASP is:

$$x = x^0;$$

$$y = \text{argmin} \{ f(z) \mid z \in N(x) \};$$

while ( $f(y) < f(x)$ ) do

  $x = y;$

  $y = \text{argmin} \{ f(z) \mid z \in N(x) \};$

end while

return $x$;

# Local search within GRASP

Another way to implement local search in GRASP is:

$x = x^0;$

$y = \text{argmin} \{ f(z) \mid z \in N(x) \};$

while ( $f(y) < f(x)$ ) do

$x = y;$

$y = \text{argmin} \{ f(z) \mid z \in N(x) \};$

end while

return x;

best improving

$x = x^0;$

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

    $x = y;$

end while

return $x;$

first improving

$x = x^0;$

$y = \text{argmin} \{ f(z) \mid z \in N(x) \};$

while ( $f(y) < f(x)$ ) do

    $x = y;$

    $y = \text{argmin} \{ f(z) \mid z \in N(x) \};$

end while

return $x;$

best improving

First improving is usually faster.

Premature convergence to low-quality local optimum is more likely to occur with best improving.

$x = x^0;$

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

$\quad x = y;$

end while

return $x$;

**first improving**

$x = x^0;$

$y = \operatorname{argmin} \{ f(z) \mid z \in N(x) \};$

while ( $f(y) < f(x)$ ) do

$\quad x = y;$

$\quad y = \operatorname{argmin} \{ f(z) \mid z \in N(x) \};$

end while

return $x$;

**best improving**

If search of $N(x)$ is done deterministically, then repeated applications of local search starting from same $x^0$ lead to same local minimum

$x = x^0;$

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

    $x = y;$

end while

return $x$;

**first improving**

$x = x^0;$

$y = \text{argmin}\,\{ f(z) \mid z \in N(x) \};$

while ( $f(y) < f(x)$ ) do

    $x = y;$

    $y = \text{argmin}\,\{ f(z) \mid z \in N(x) \};$

end while

return $x$;

**best improving**

If search of N(x) is done deterministically, then repeated applications of local search starting from same $x^0$ lead to same local minimum

Hashing can avoid repeating local search from previous $x^0$

{ Woodruff & Zemel (1993), Ribeiro et. al (1997), Martins et al. (2000) }

if ( $f(x^0)$ < CUTOFF) then

$x = x^0$;

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

$x = y$;

end while

return x;

end if

**first improving**

if ( $f(x^0)$ < CUTOFF) then

$x = x^0$;

$y = \text{argmin}\ \{ f(z) \mid z \in N(x) \}$;

while ( $f(y) < f(x)$ ) do

$x = y$;

$y = \text{argmin}\ \{ f(z) \mid z \in N(x) \}$;

end while

return x;

end if

**best improving**

Filtering to avoid application of local search to low quality solutions, only promising solutions are investigated: { Feo, Resende, & Smith (1994), Prais & Ribeiro (2000), Martins et. al (2000) }

# Local search within GRASP

As the name implies, local search, is confined to a localized region of the solution space.

To escape from local minima and broaden the search several alternatives have been proposed to replace local search in GRASP:

variable neighborhood descent (VND)

variable neighborhood search (VNS)

short-term memory tabu search (TS)

simulated annealing (SA)

iterated local search (ILS)

very large-scale neighborhood search (VLSNS)

# Local search within GRASP

As the name implies, local search, is confined to a localized region of the solution space.

To escape from local minima and broaden the search several alternatives have been proposed to replace local search in GRASP:

**variable neighborhood descent (VND)**

variable neighborhood search (VNS)

short-term memory tabu search (TS)

simulated annealing (SA)

iterated local search (ILS)

very large-scale neighborhood search (VLSNS)

# GRASP VND local search

Instead of using a single neighborhood, VND uses K not necessarily related neighborhoods $N_1$, $N_2$, ..., $N_K$.

# GRASP VND local search

$x = x_0$; $k = 1$;

while ($k \leq K$) do

    if ( $\exists\, s \in N_k(x)$ such that $f(s) < f(x)$ ) then

        $x = s$; $k = 1$; break;

    endif

    $k = k + 1$;

endwhile

return $x$;

Instead of using a single neighborhood, VND uses $K$ not necessarily related neighborhoods $N_1, N_2, \ldots, N_K$.

# GRASP VND local search

$x = x_0$; $k = 1$;

while ($k \leq K$) do

scan all K neighborhoods

if ( $\exists\, s \in N_k(x)$ such that $f(s) < f(x)$ ) then

$x = s$; $k = 1$; break;

endif

$k = k + 1$;

endwhile

return x;

# GRASP VND local search

$x = x_0$; $k = 1$;

while ($k \leq K$) do — scan all K neighborhoods

   if ( $\exists\, s \in N_k(x)$ such that $f(s) < f(x)$ ) then

      $x = s$; $k = 1$; break; — found improving solution

   endif

   $k = k + 1$;

endwhile

return $x$;

# GRASP VND local search

$x = x_0$; $k = 1$;

while ($k \leq K$) do      scan all K neighborhoods

    if ( $\exists\, s \in N_k(x)$ such that $f(s) < f(x)$ ) then

        $x = s$; $k = 1$; break;      found improving solution in $N_k$

    endif

    $k = k + 1$;      x is a local mimimum of $N_k$

endwhile

return x;

# GRASP VND local search

$x = x_0$; $k = 1$;

while ($k \leq K$) do — scan all K neighborhoods

if ( $\exists\, s \in N_k(x)$ such that $f(s) < f(x)$ ) then

$x = s$; $k = 1$; break; — found improving solution in $N_k$

endif

$k = k + 1$; — x is a local mimimum of $N_k$

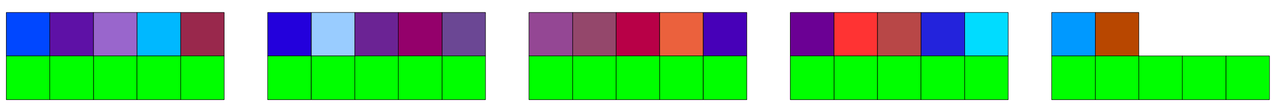
endwhile

return x; — x is a local mimimum of $N_k$, for $k = 1,\ldots,K$

# GRASP VND local search

example: scheduling of multi-grouped units

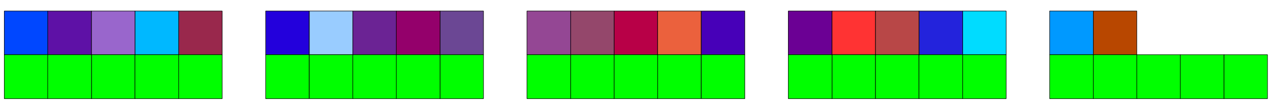
Input: Assignment of units to periods:



# GRASP VND local search

example: scheduling of multi-grouped units

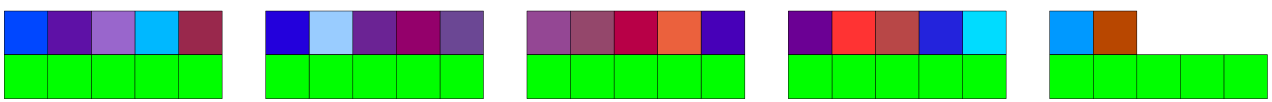
Local search: Examine neighborhood of current solution. If better solution found, make it current solution.



# GRASP VND local search

example: scheduling of multi-grouped units

Three neighborhoods: Swap units, move unit, swap periods.

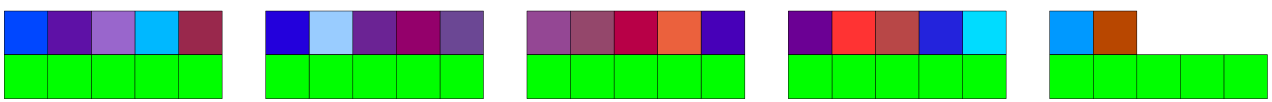


# GRASP VND local search

example: scheduling of multi-grouped units

Swap units neighborhood: Swaps places of two units assigned to distinct periods.

solution

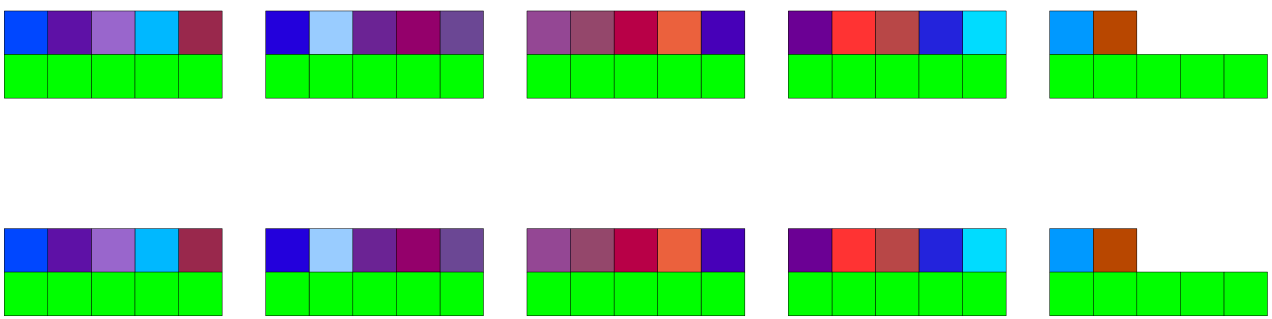


# GRASP VND local search

example: scheduling of multi-grouped units

Swap units neighborhood: Swaps places of two units assigned to distinct periods.

solution

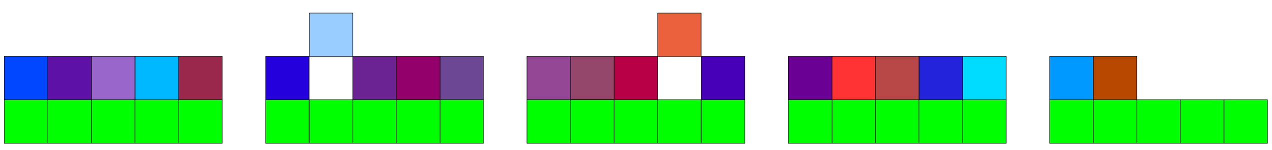


# GRASP VND local search

example: scheduling of multi-grouped units

Swap units neighborhood: Swaps places of two units assigned to distinct periods.

solution

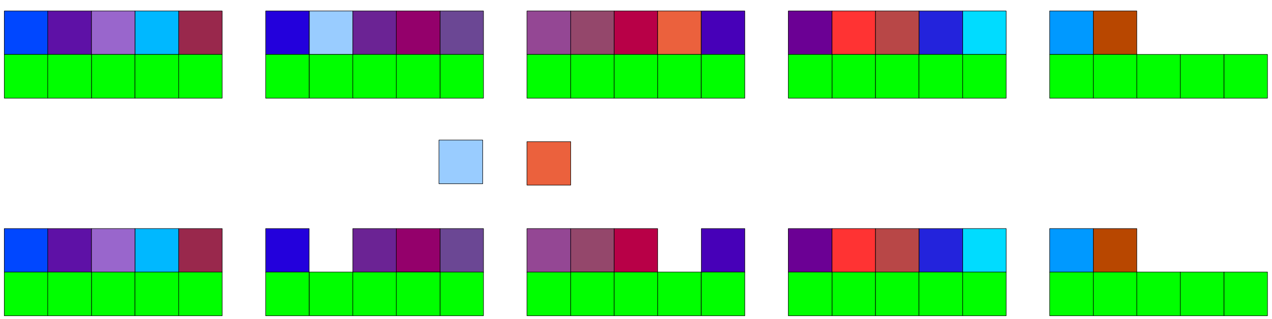


# GRASP VND local search

example: scheduling of multi-grouped units

Swap units neighborhood: Swaps places of two units assigned to distinct periods.

solution

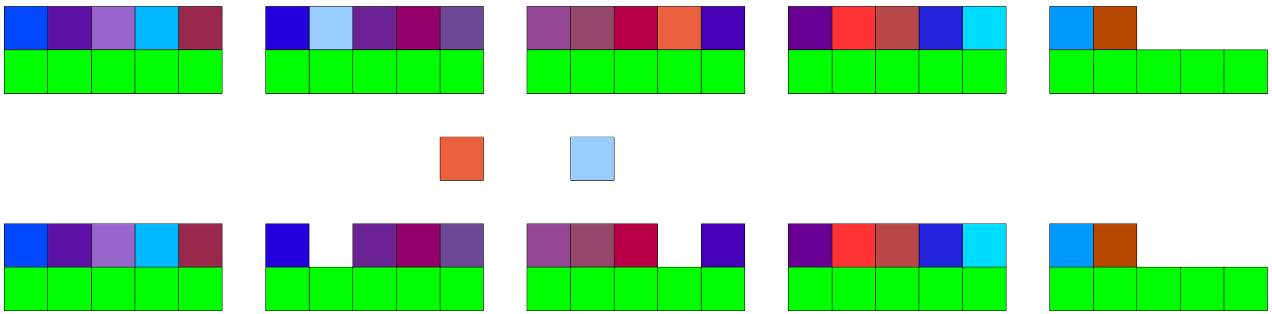


# GRASP VND local search

example: scheduling of multi-grouped units

Swap units neighborhood: Swaps places of two units assigned to distinct periods.

solution

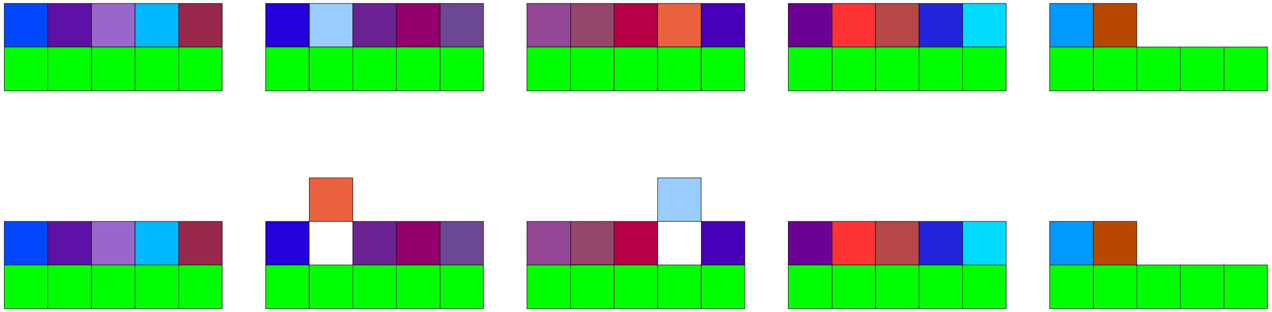


# GRASP VND local search

example: scheduling of multi-grouped units

Swap units neighborhood: Swaps places of two units assigned to distinct periods.

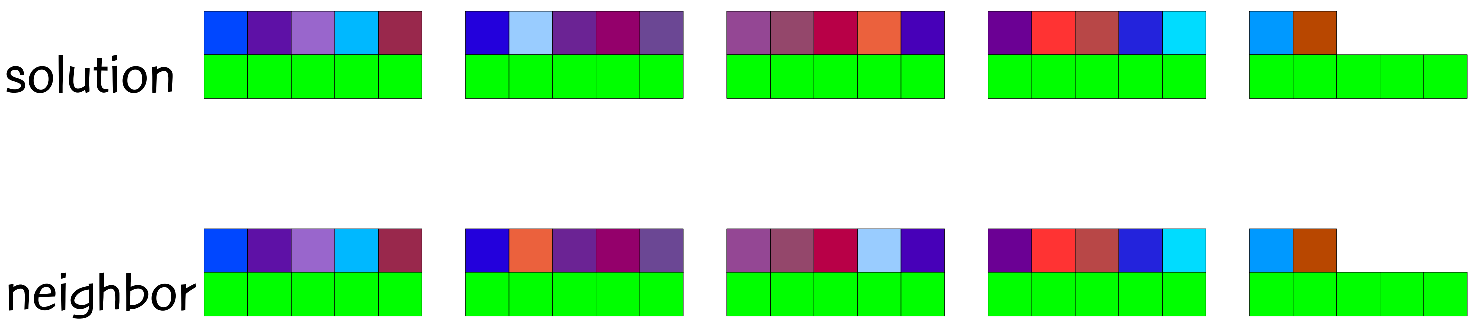
solution



# GRASP VND local search

example: scheduling of multi-grouped units

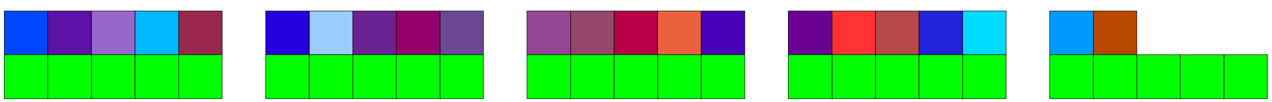
Swap units neighborhood: Swaps places of two units assigned to distinct periods.



# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

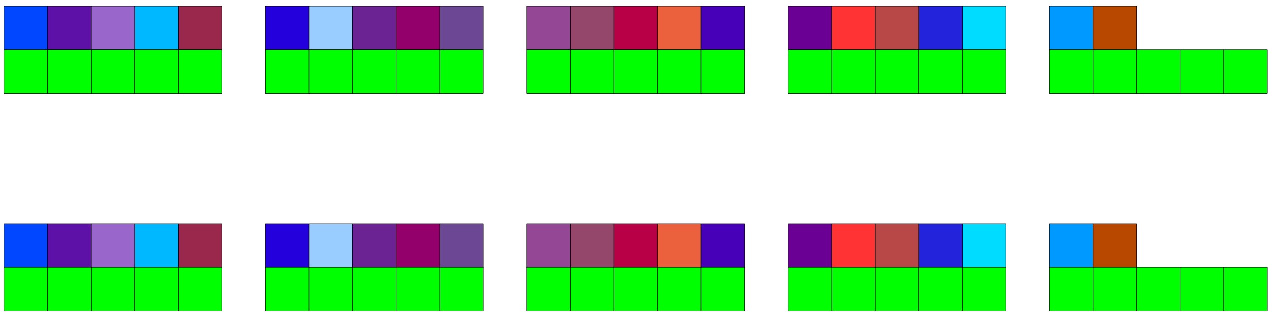
solution 

# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

solution

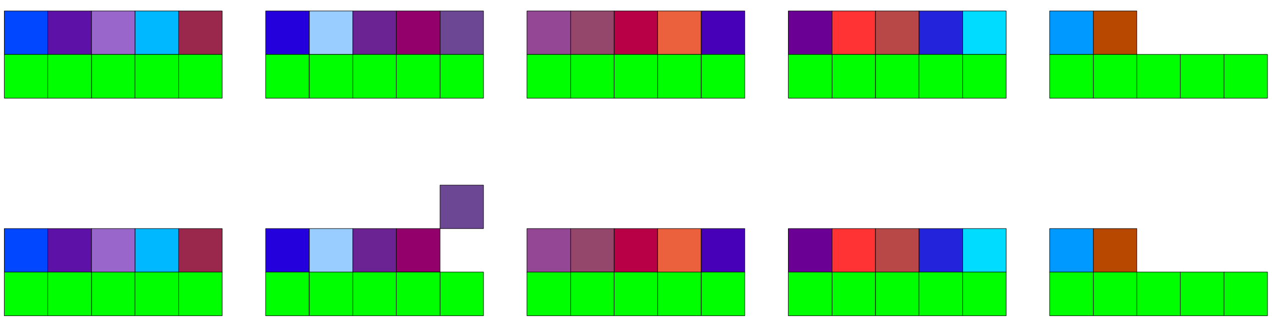


# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

solution

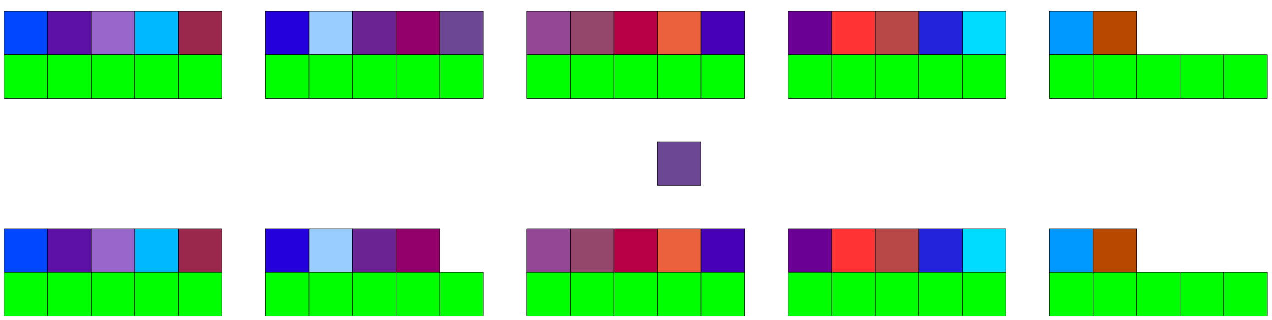


# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

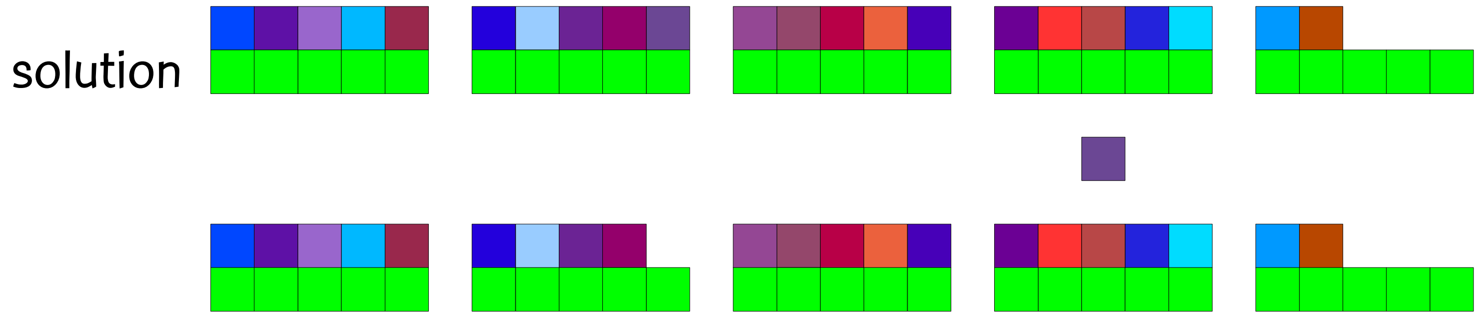
solution



# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

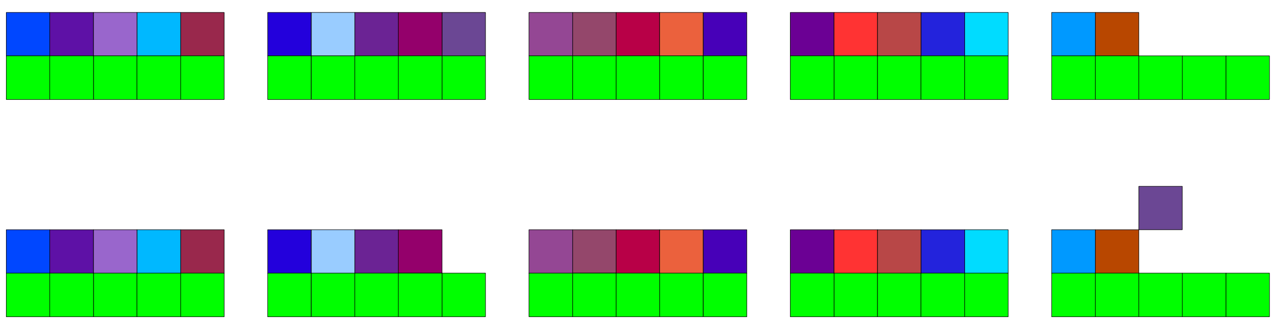


# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

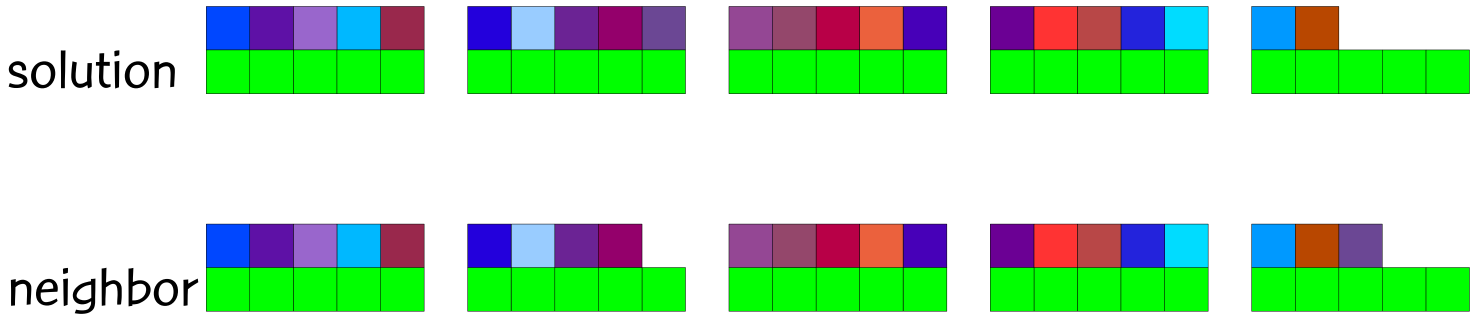
solution



# GRASP VND local search

example: scheduling of multi-grouped units

Move unit neighborhood: Moves unit from current period to available period.

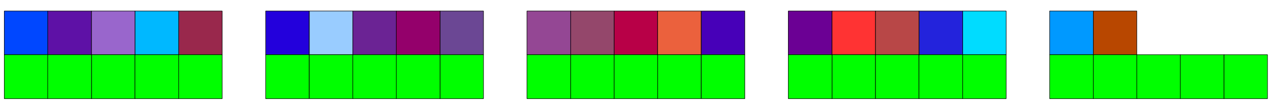


# GRASP VND local search

example: scheduling of multi-grouped units

Swap periods neighborhood: Swap all units in period i with all units in period j.

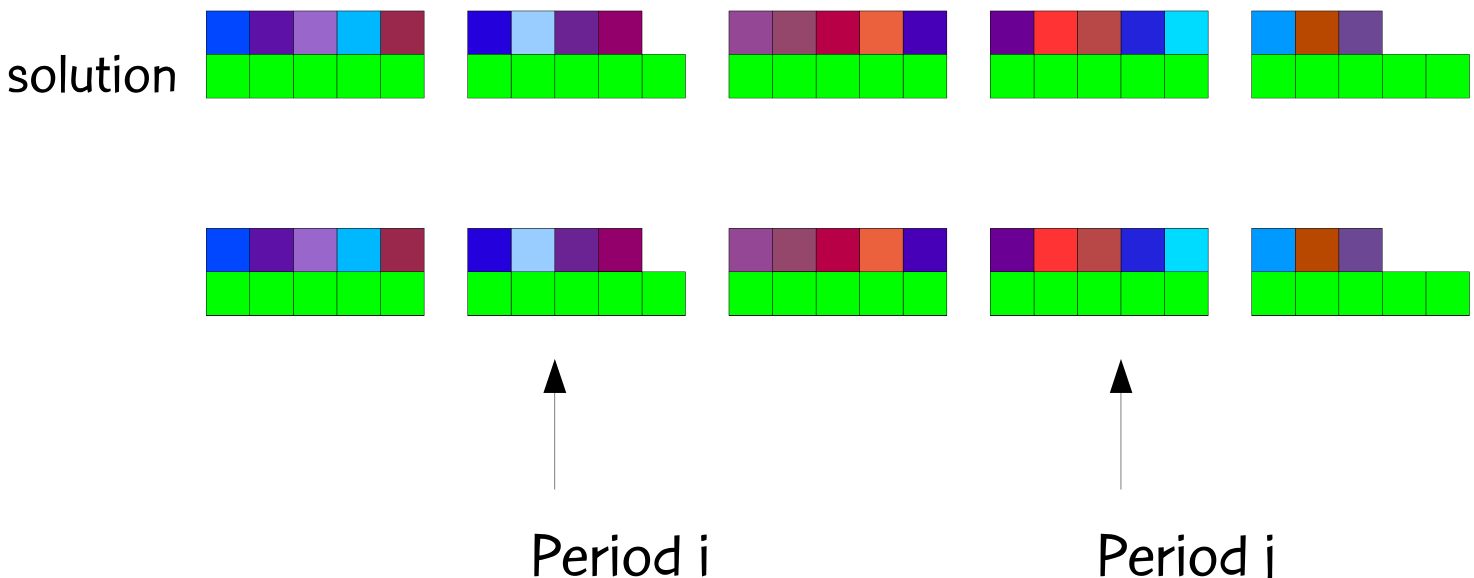
solution



# GRASP VND local search

example: scheduling of multi-grouped units

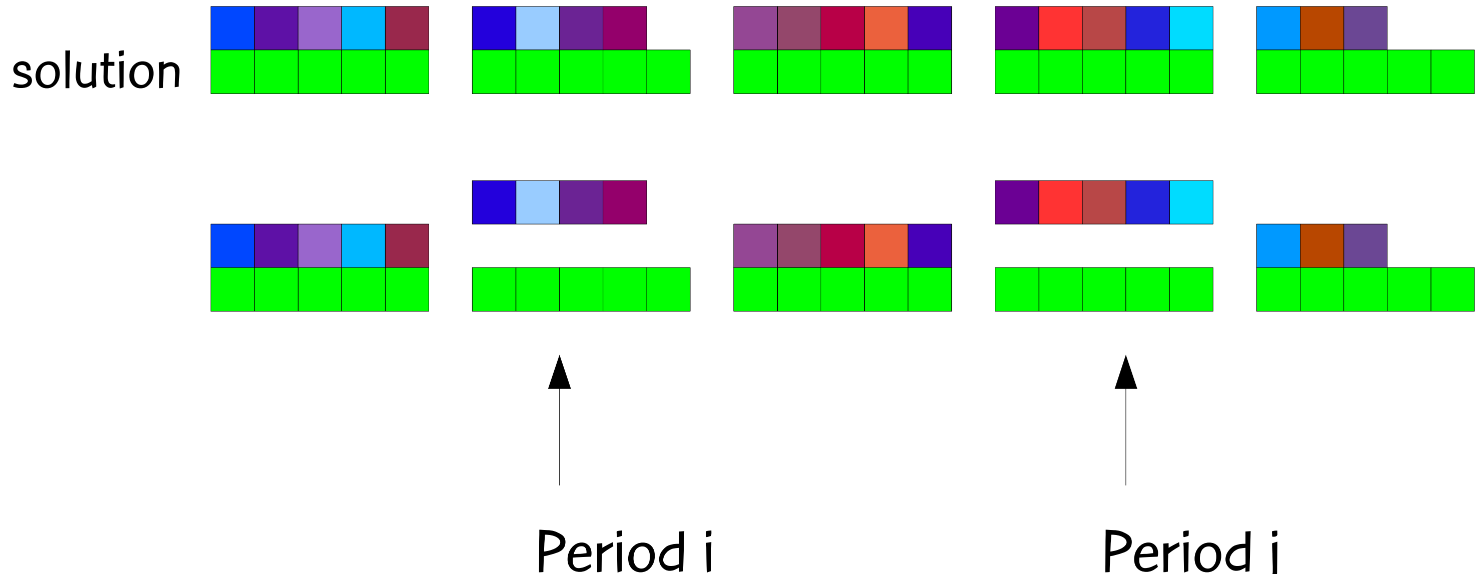
Swap periods neighborhood: Swap all units in period i with all units in period j.



# GRASP VND local search

example: scheduling of multi-grouped units

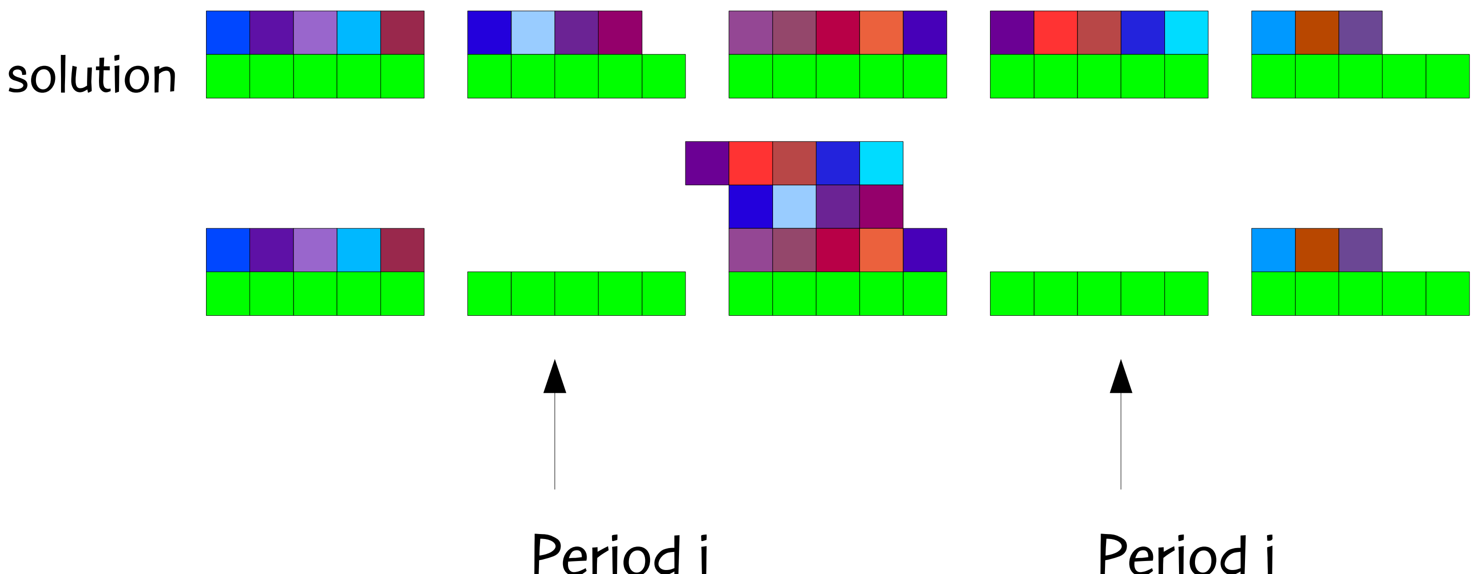
Swap periods neighborhood: Swap all units in period i with all units in period j.



# GRASP VND local search

example: scheduling of multi-grouped units

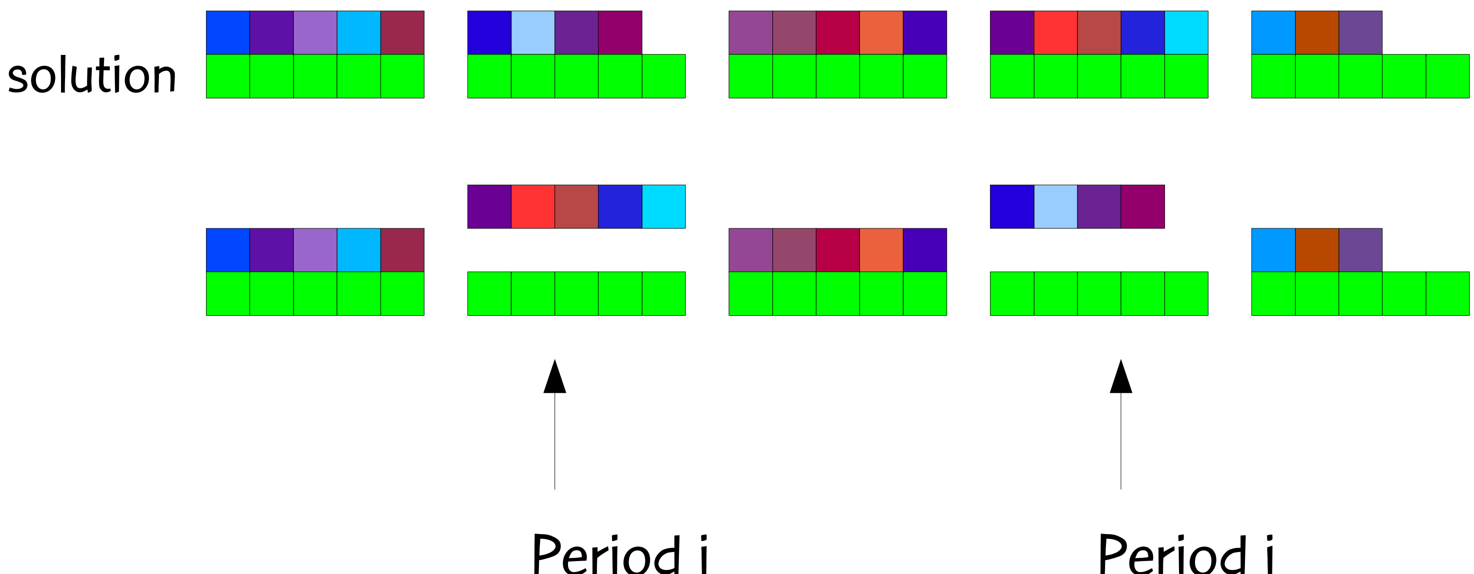
Swap periods neighborhood: Swap all units in period i with all units in period j.



# GRASP VND local search

example: scheduling of multi-grouped units

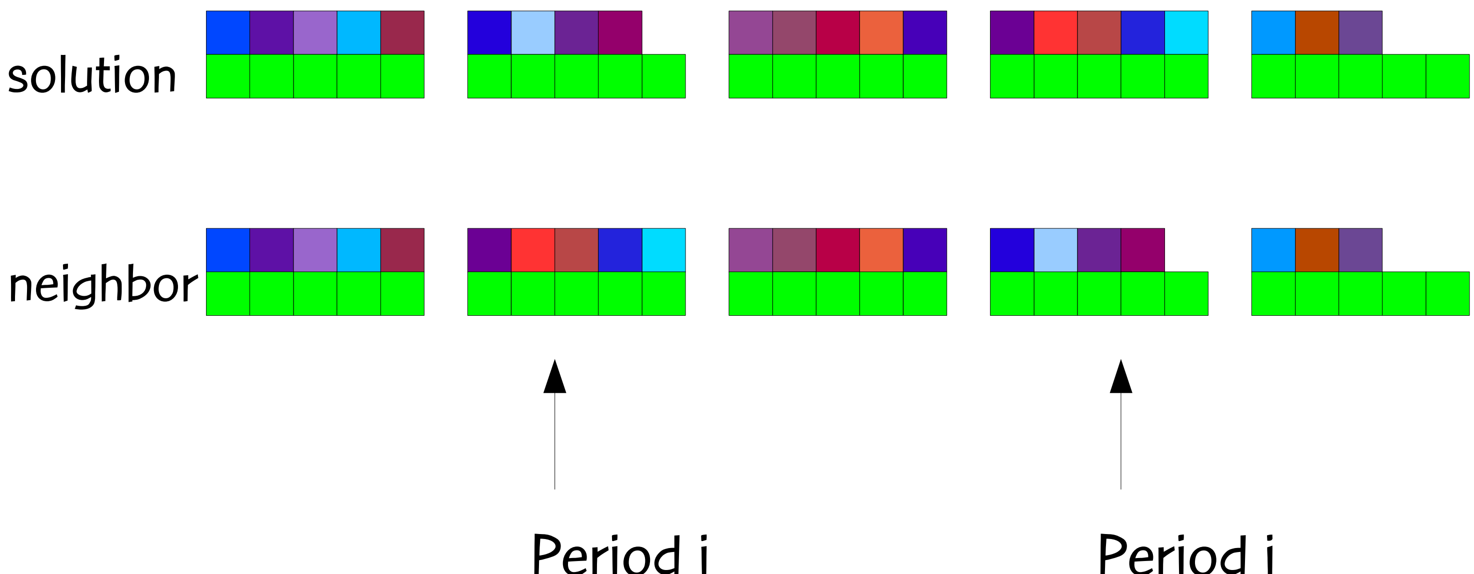
Swap periods neighborhood: Swap all units in period i with all units in period j.



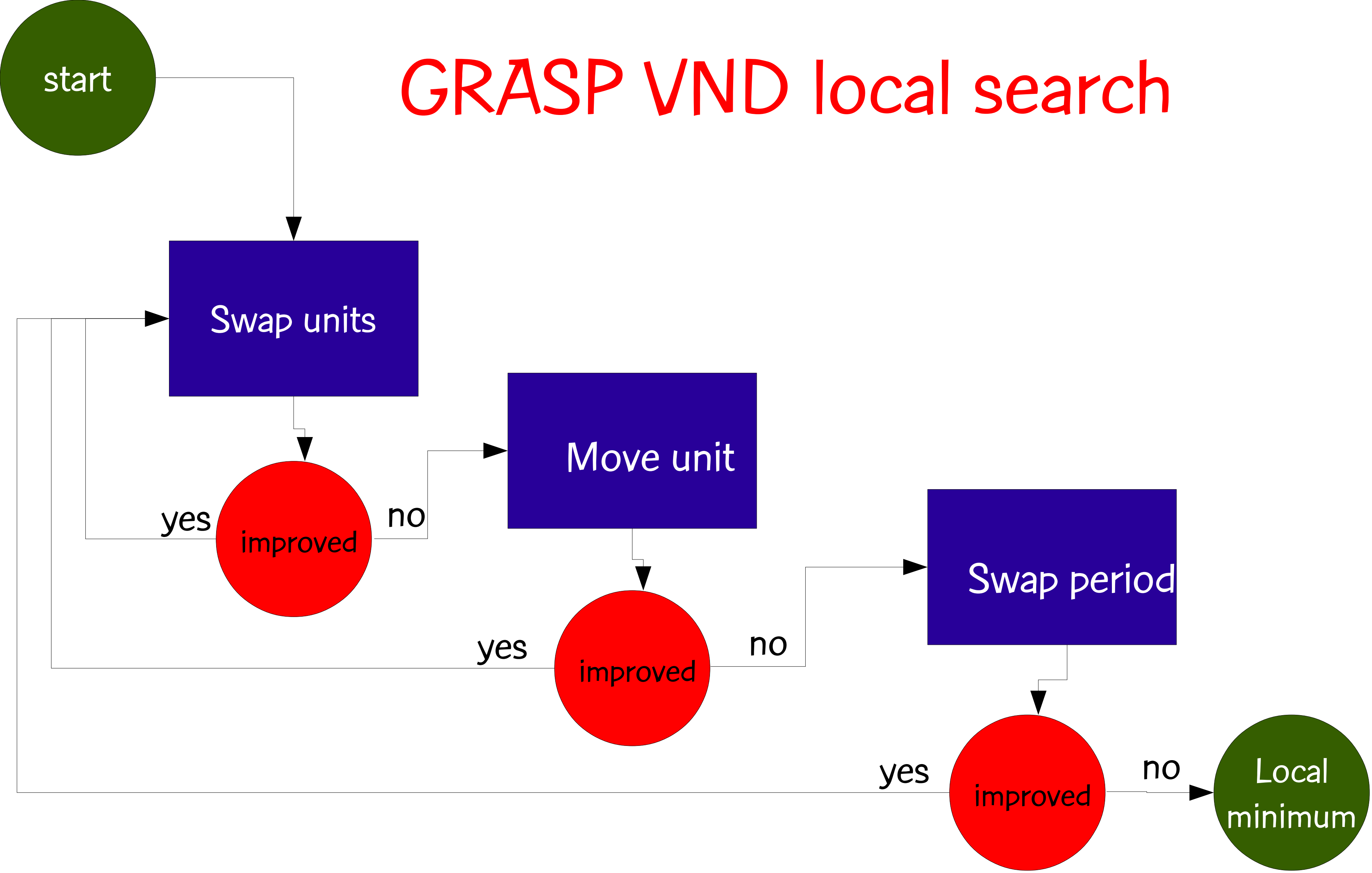
# GRASP VND local search

example: scheduling of multi-grouped units

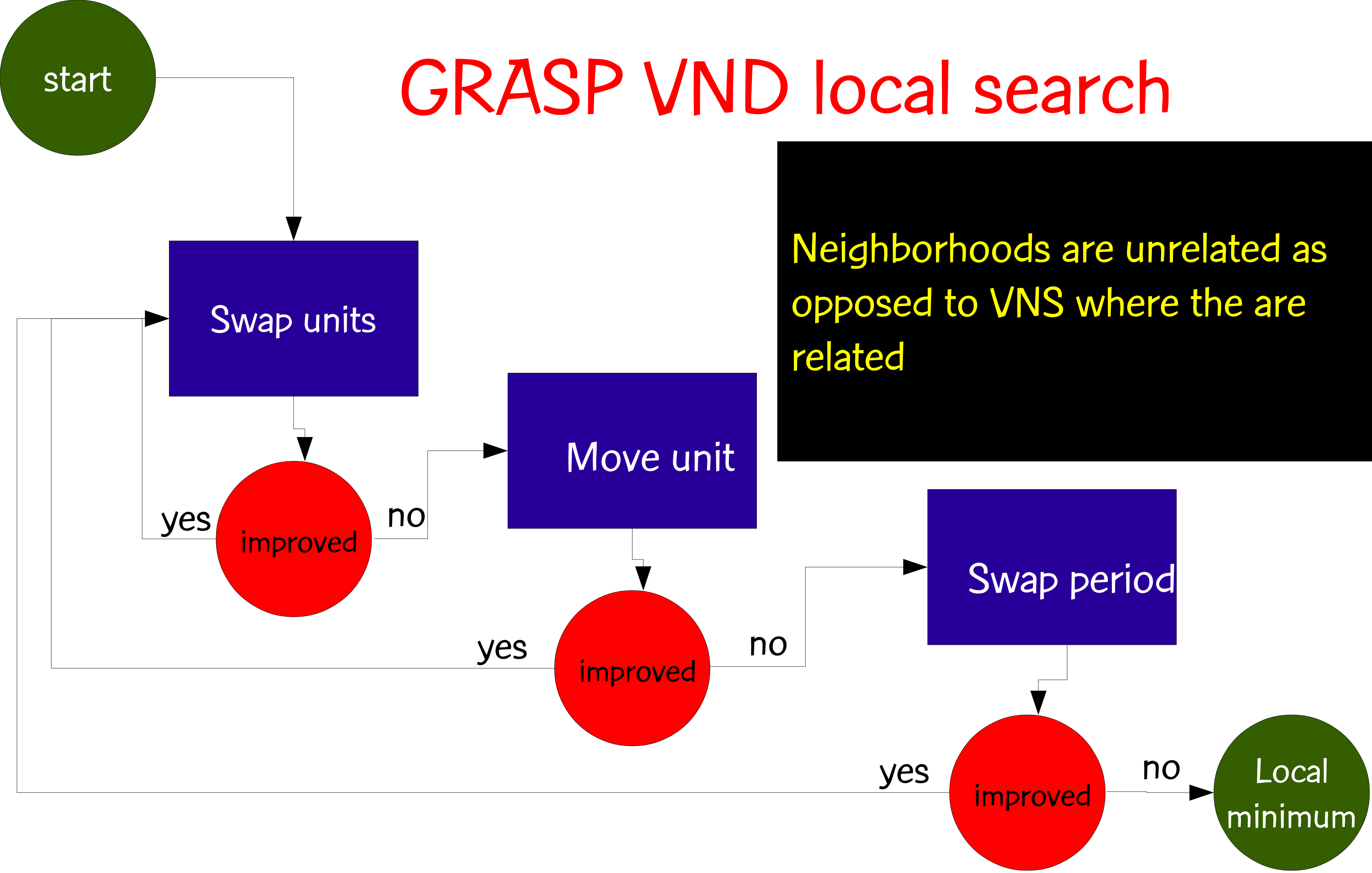
Swap periods neighborhood: Swap all units in period i with all units in period j.



# GRASP VND local search



# GRASP VND local search

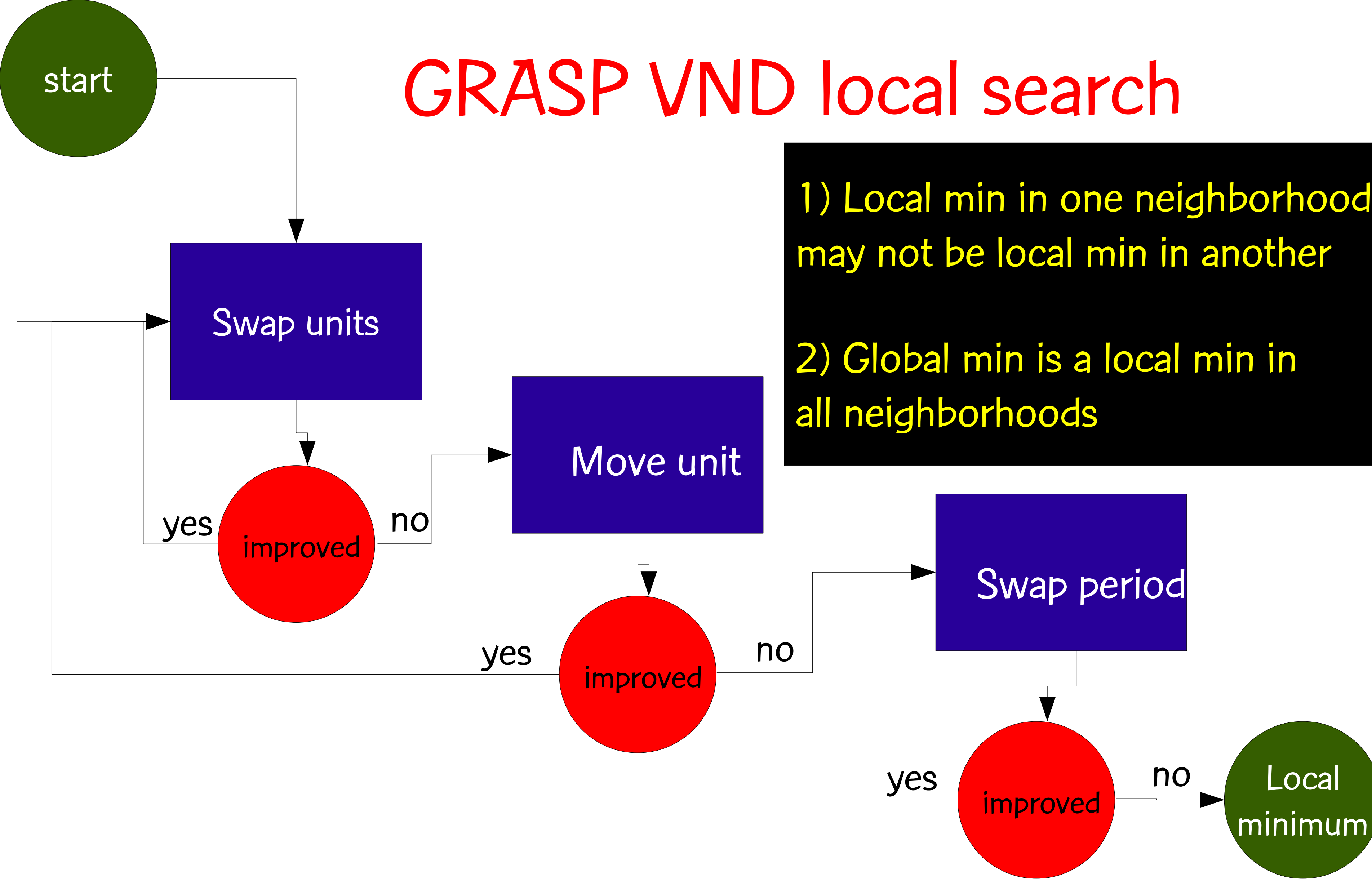



Neighborhoods are unrelated as opposed to VNS where the are related

# GRASP VND local search

1) Local min in one neighborhood may not be local min in another

2) Global min is a local min in all neighborhoods



# Examples of VND within GRASP

Martins et al. (1999): Steiner problem in graphs

Ribeiro and Souza (2002): degree constrained minimum spanning tree

Ribeiro et al. (2002): Steiner problem in graphs

Ribeiro and Vianna (2005): Phylogeny problem

Andrade and Resende (2006): PBX phone migration

# Path-relinking (PR)

# Path-relinking

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

Originally proposed in the context of tabu search and scatter search.

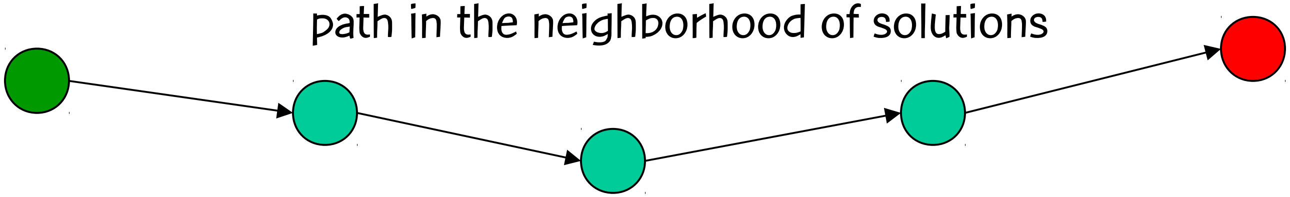
Paths in the solution space leading to other elite solutions are explored in the search for better solutions.

# Path-relinking

Exploration of trajectories that connect high quality (elite) solutions:

path in the neighborhood of solutions

initial solution



guiding solution

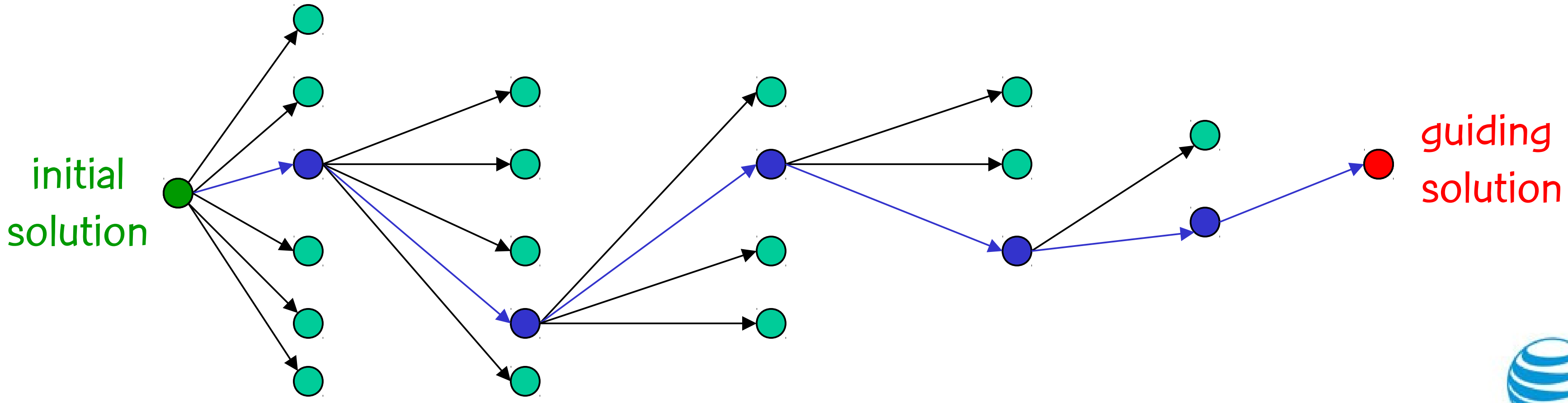


Your world. Delivered.

# 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:

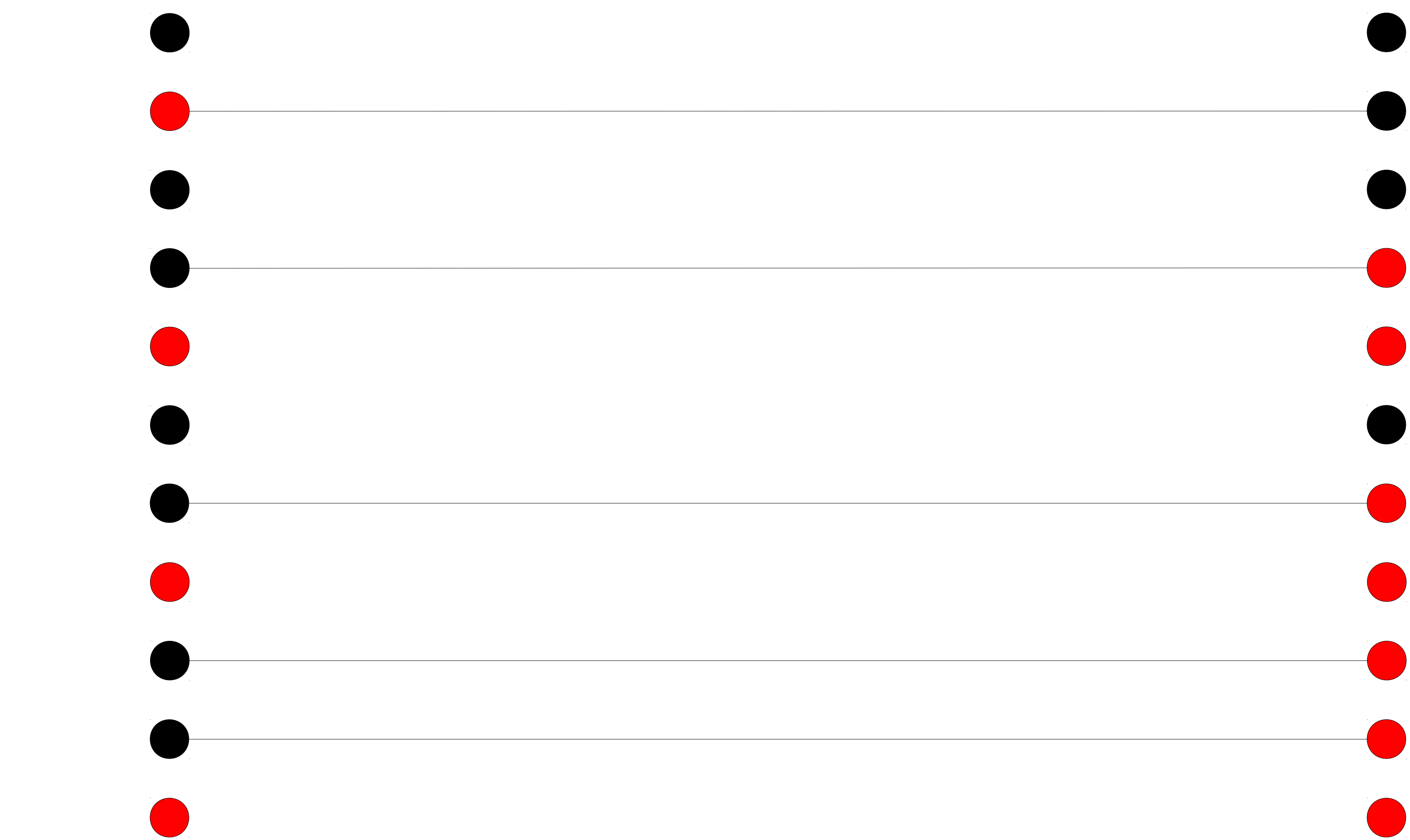


# PR example

starting solution

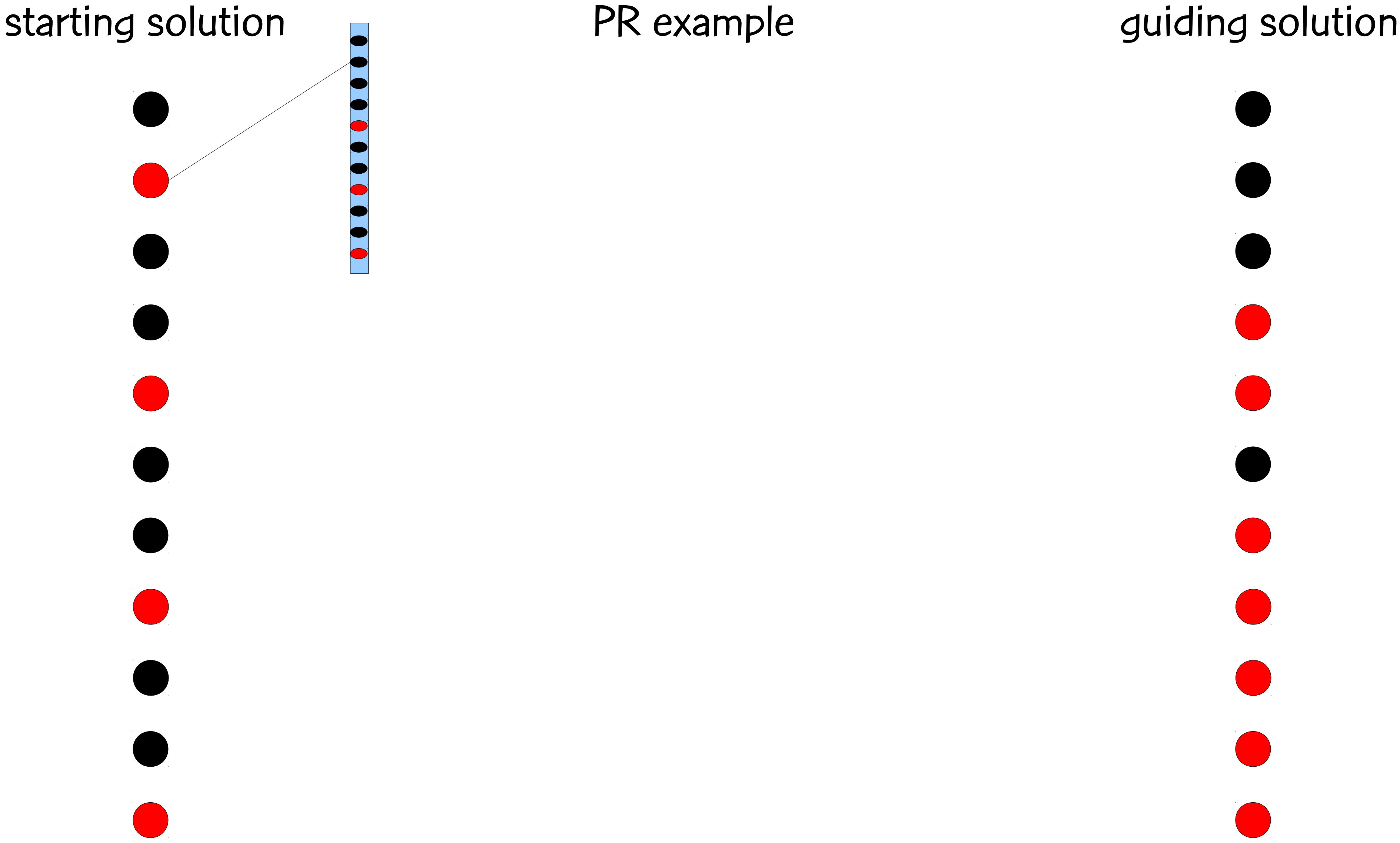
guiding solution

# PR example

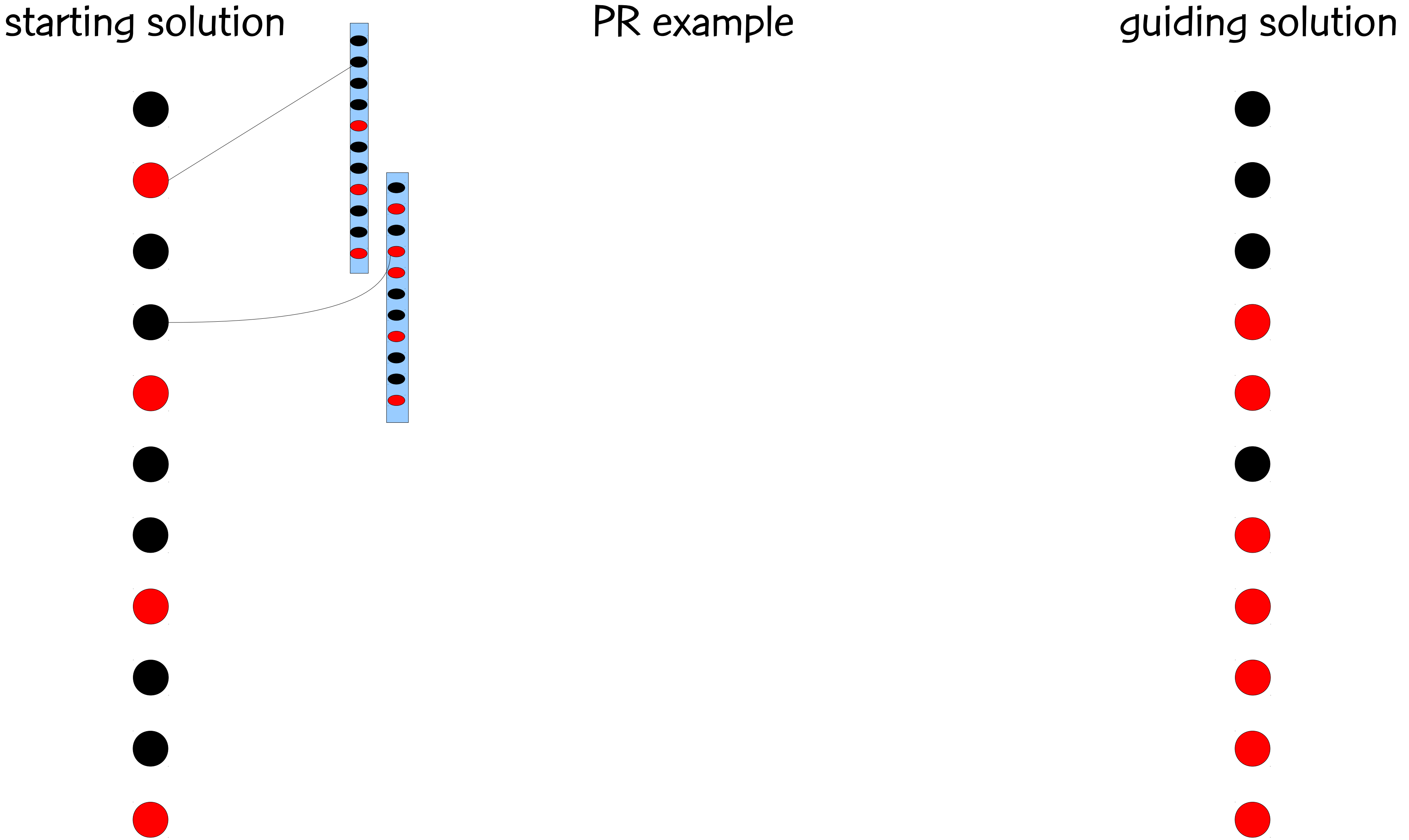



$$|\Delta(x,y)| = 5$$

# PR example



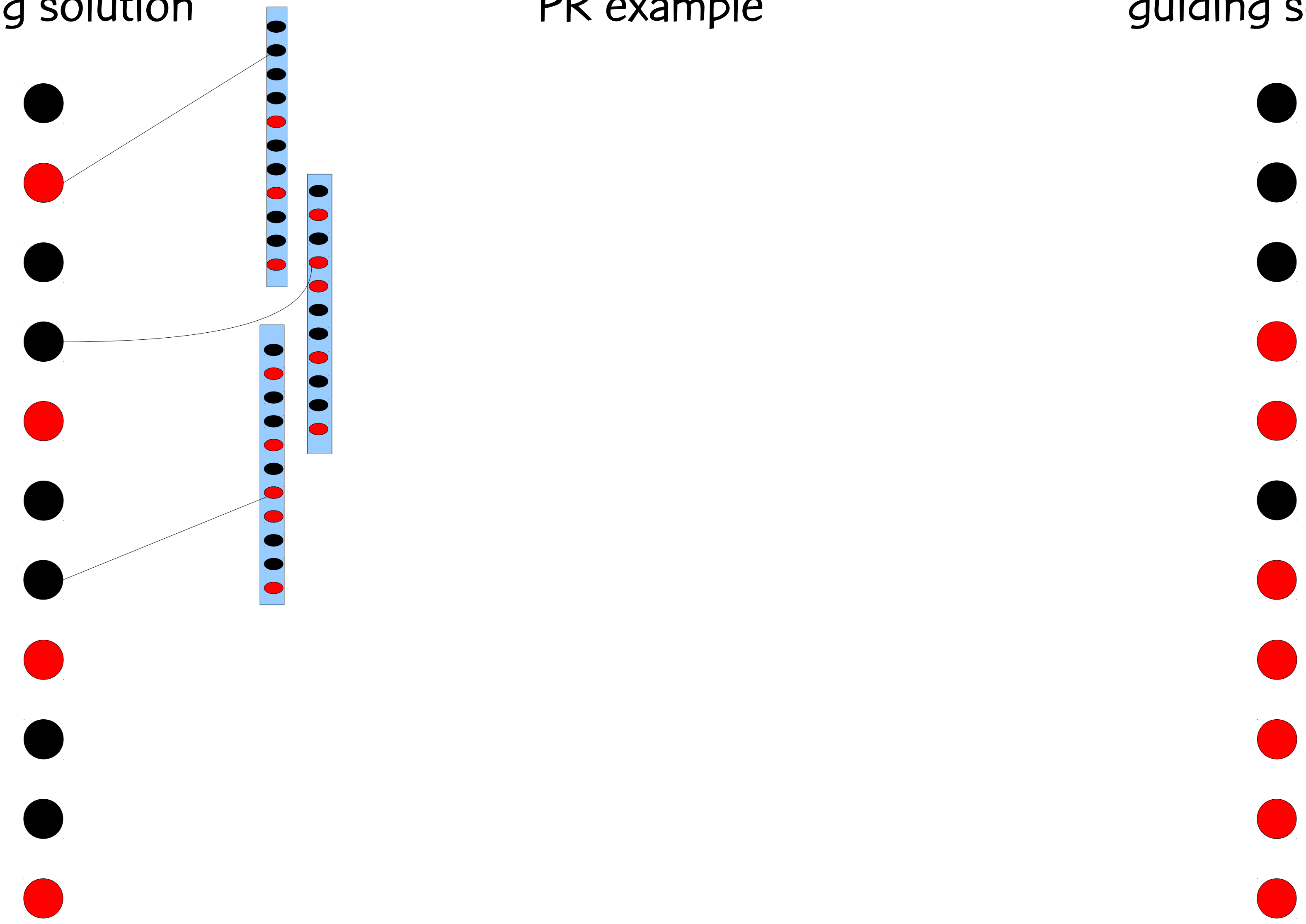
# PR example



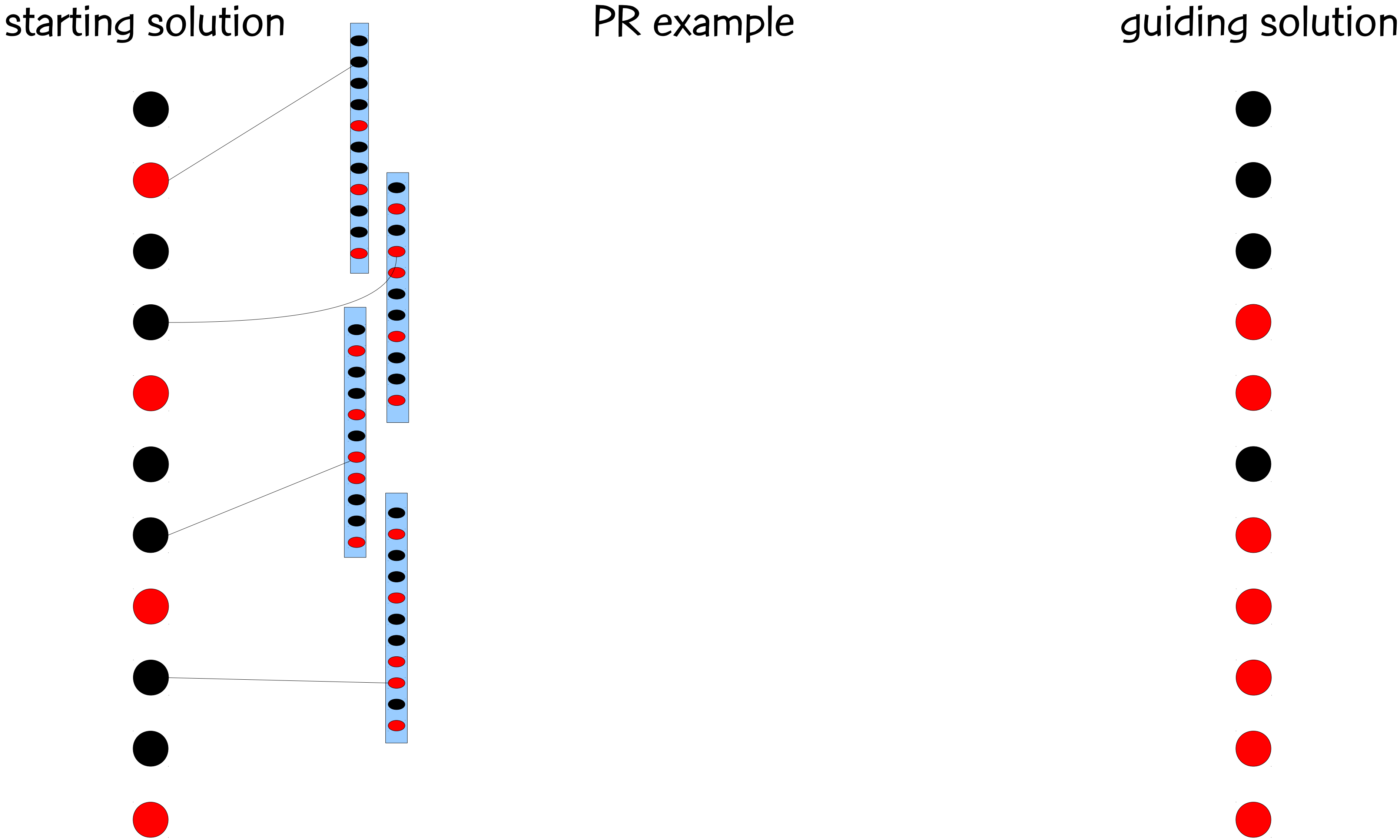
# PR example

starting solution

guiding solution



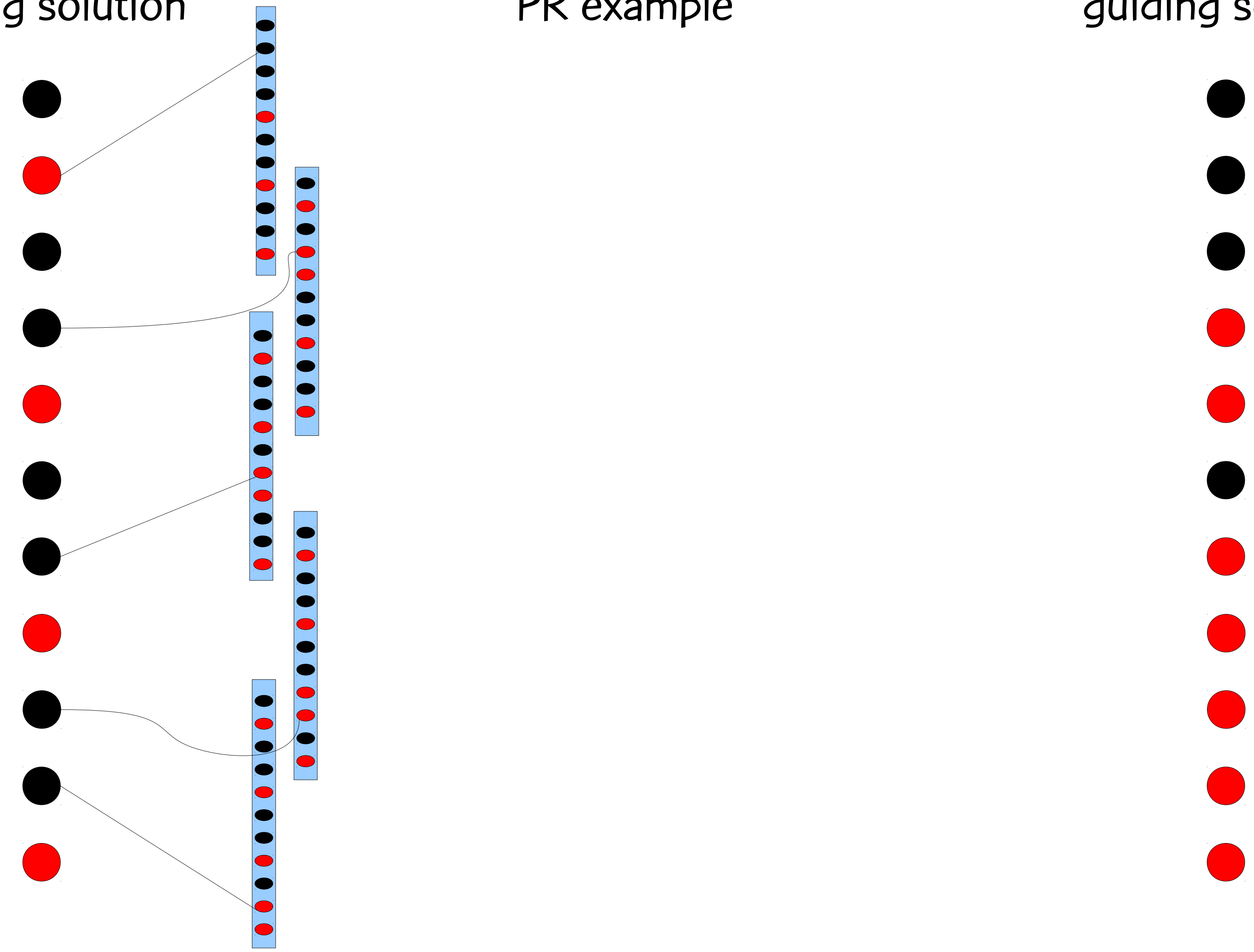
# PR example



starting solution

# PR example

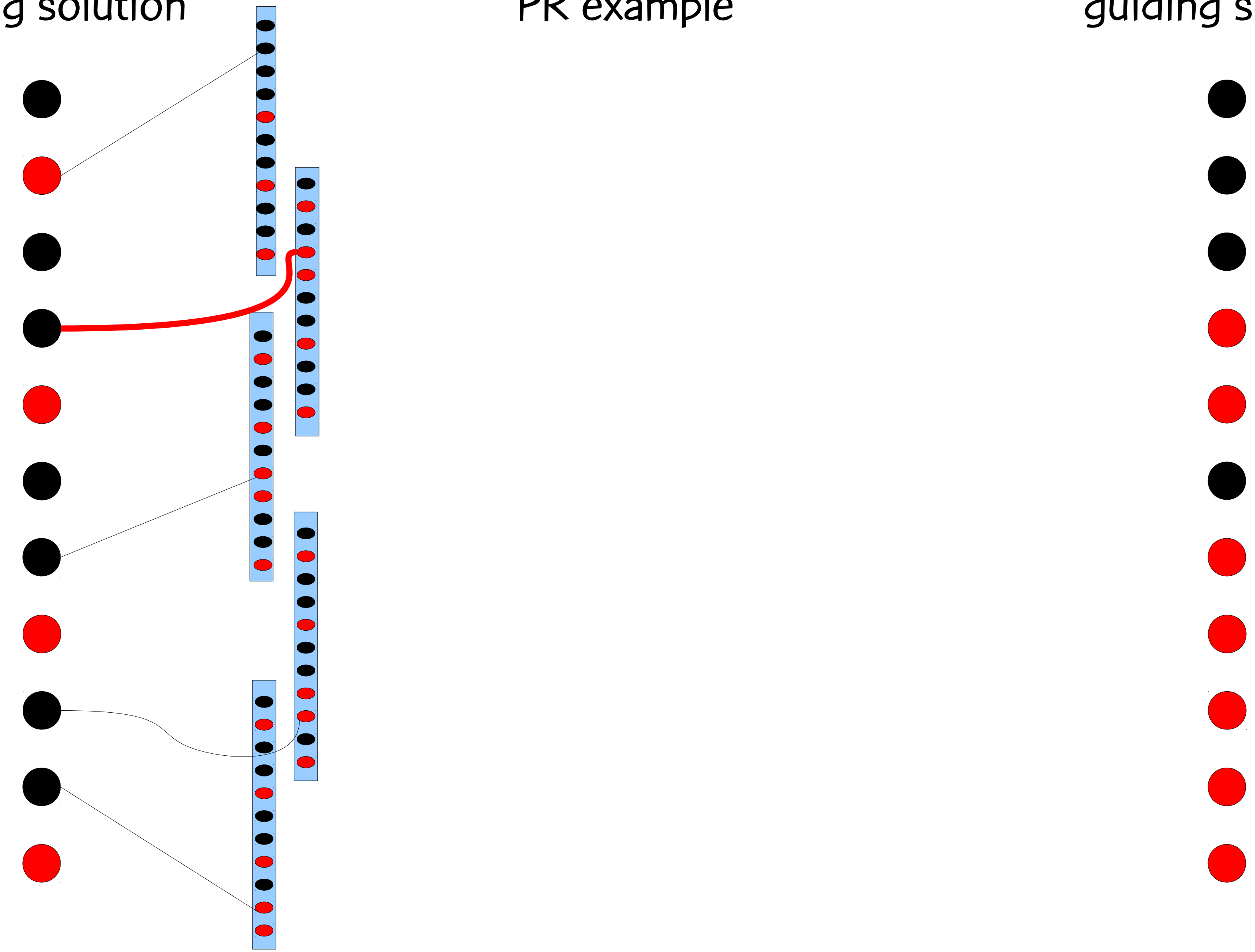
guiding solution



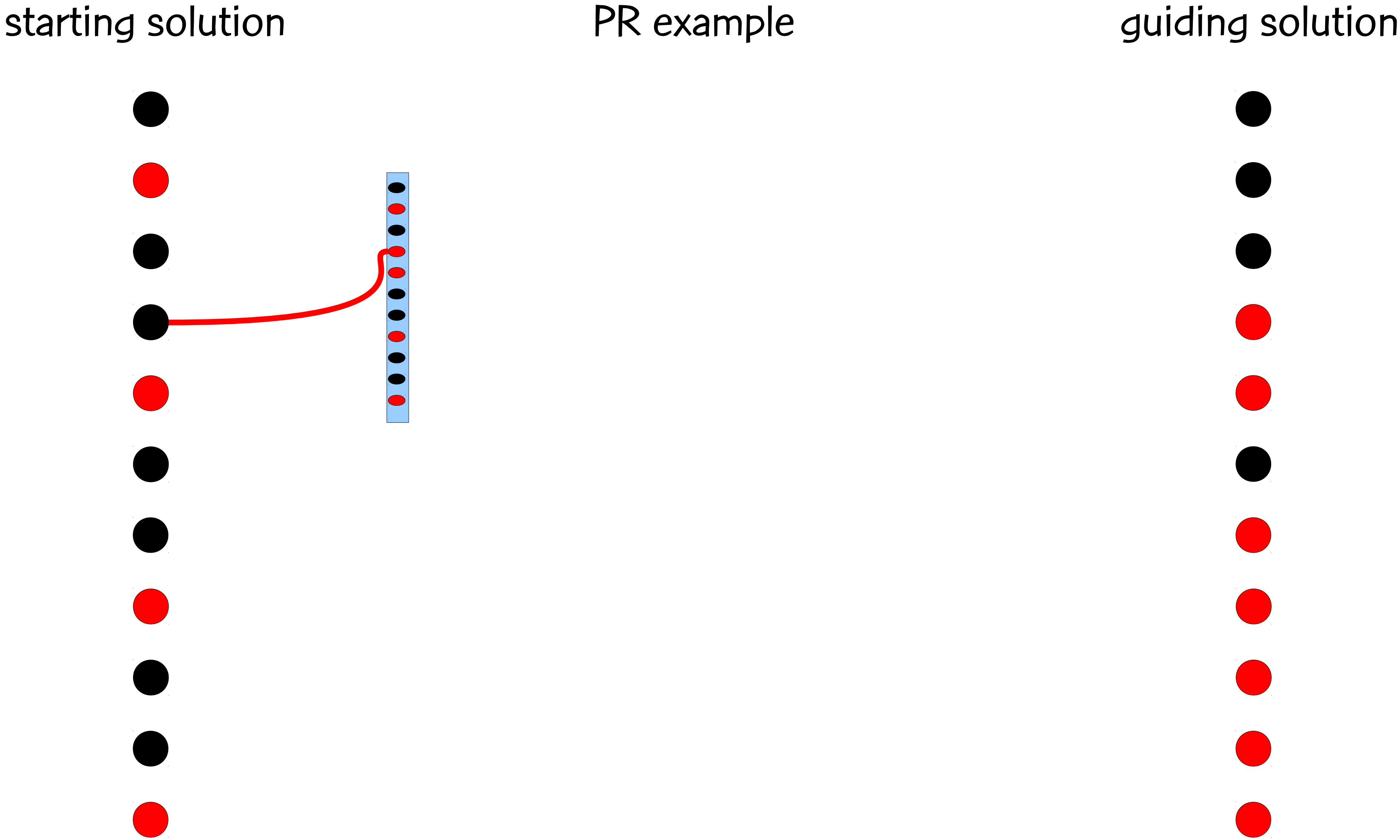
starting solution

# PR example

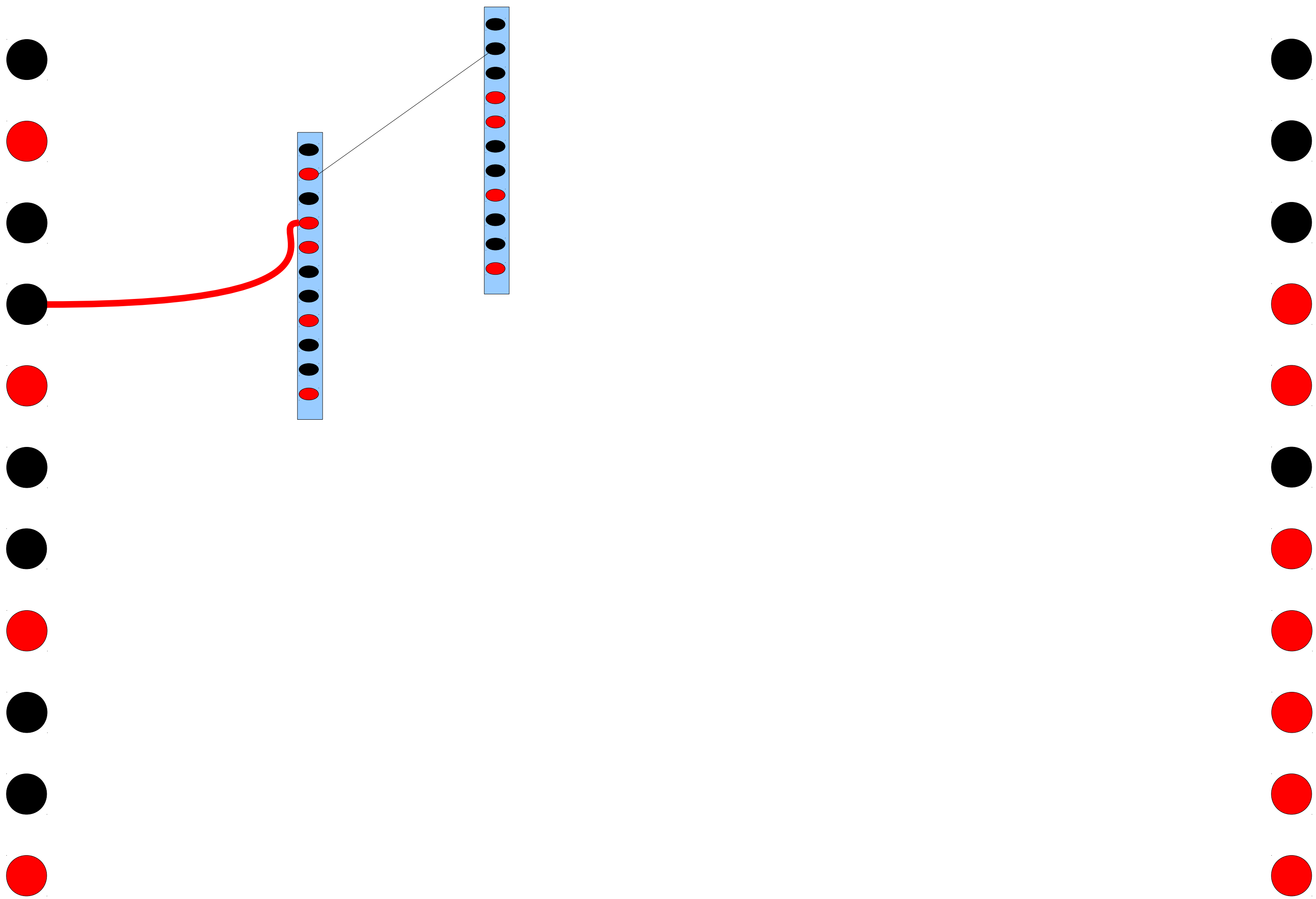
guiding solution



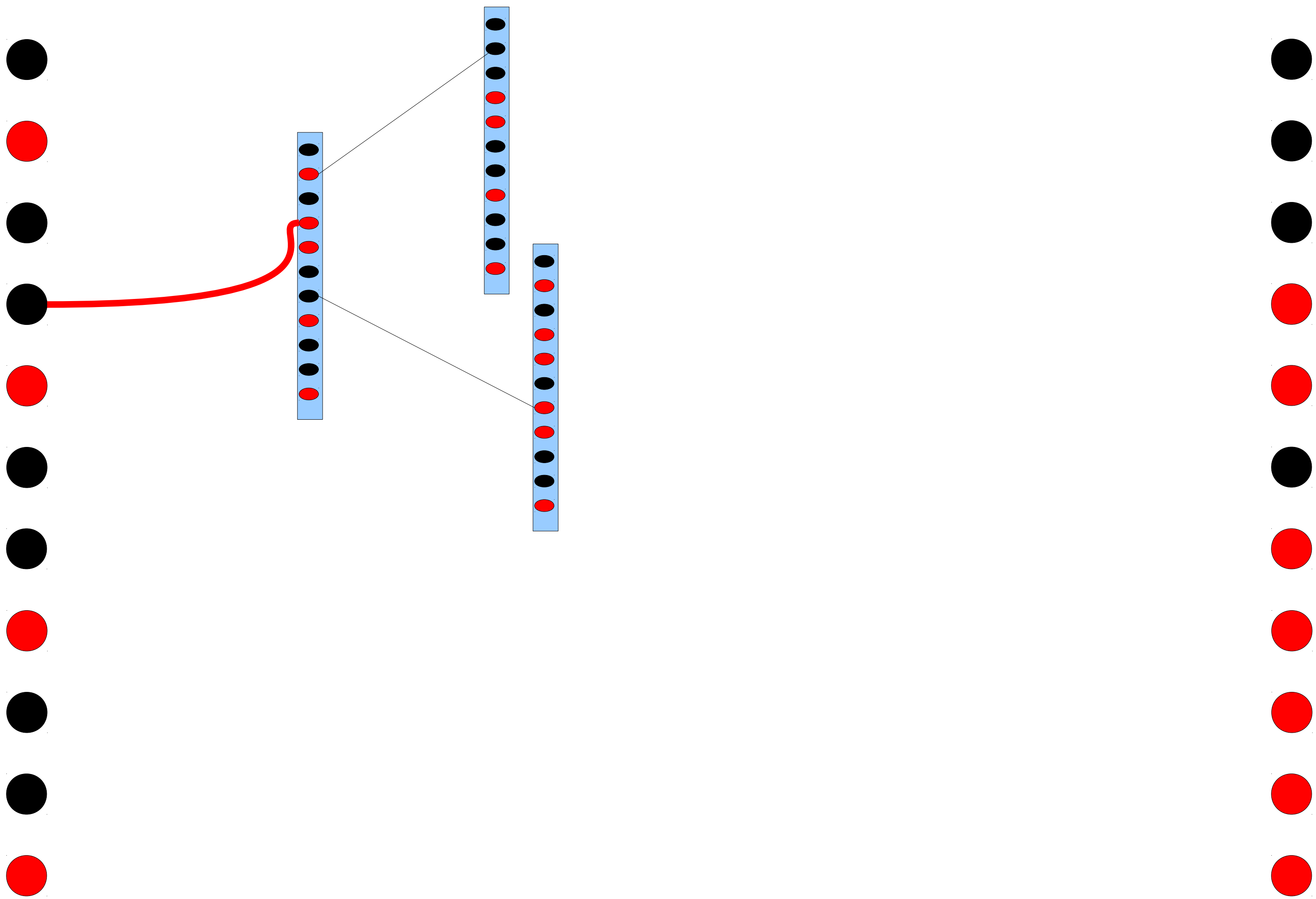
# PR example



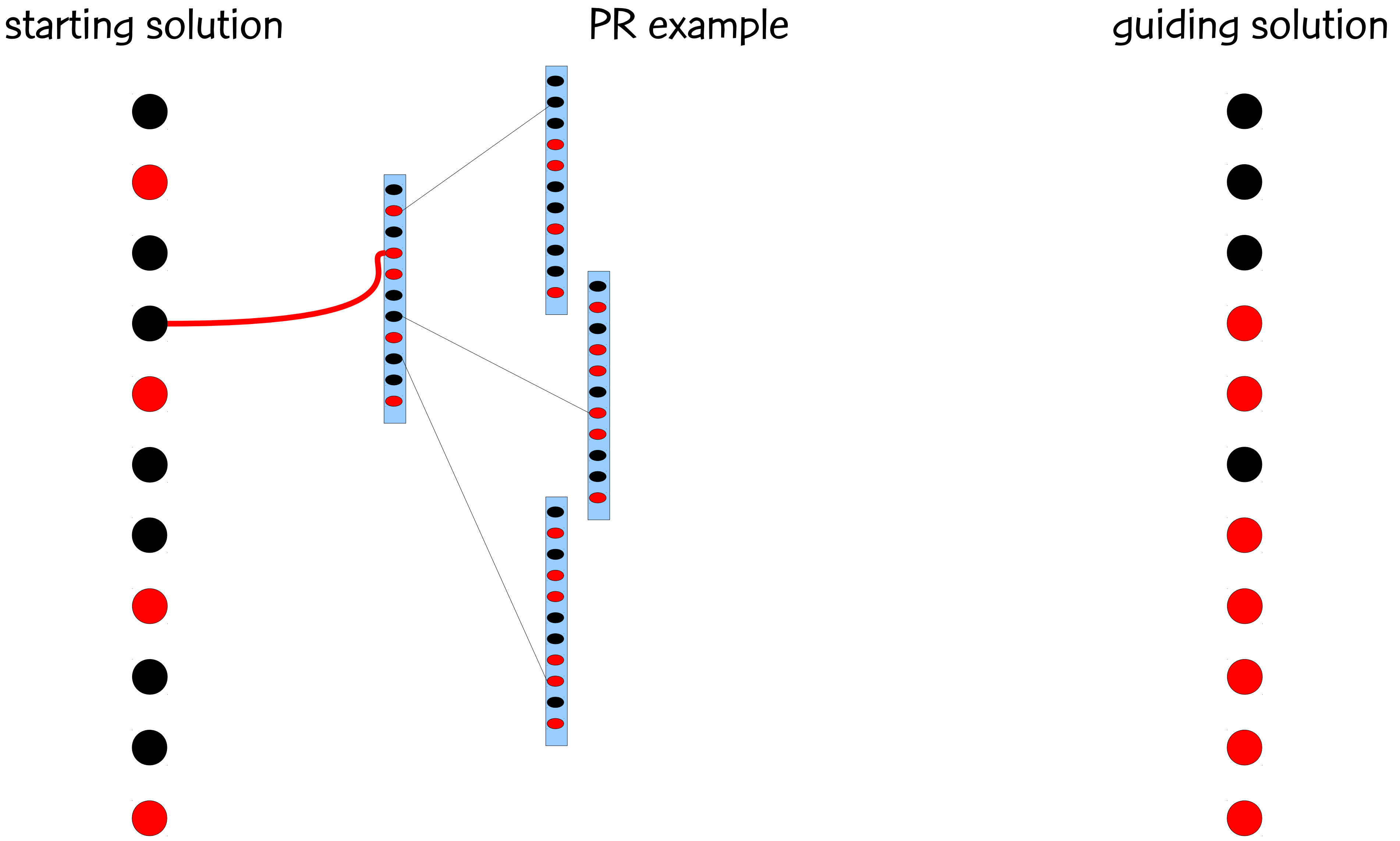
starting solution PR example guiding solution



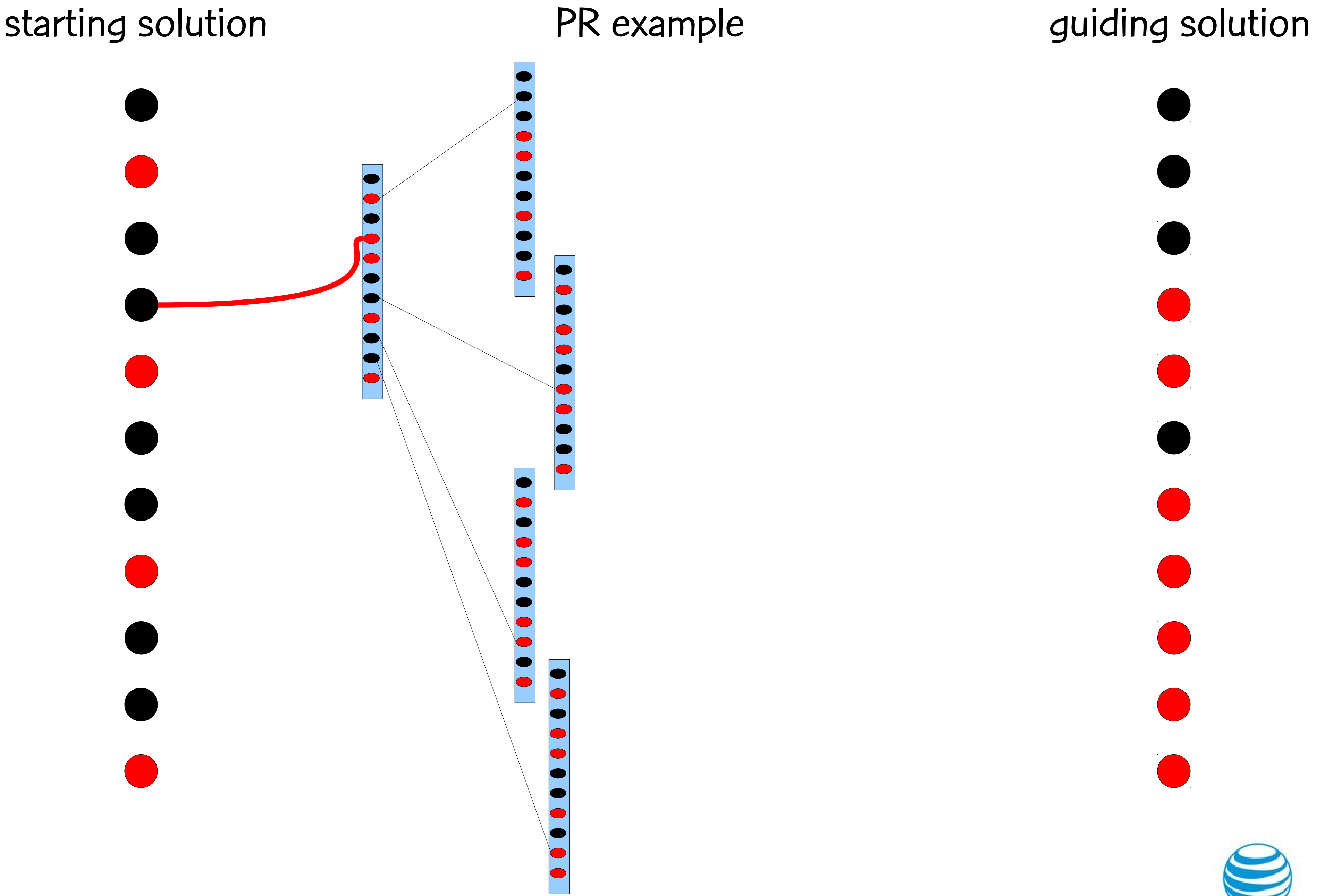
# PR example



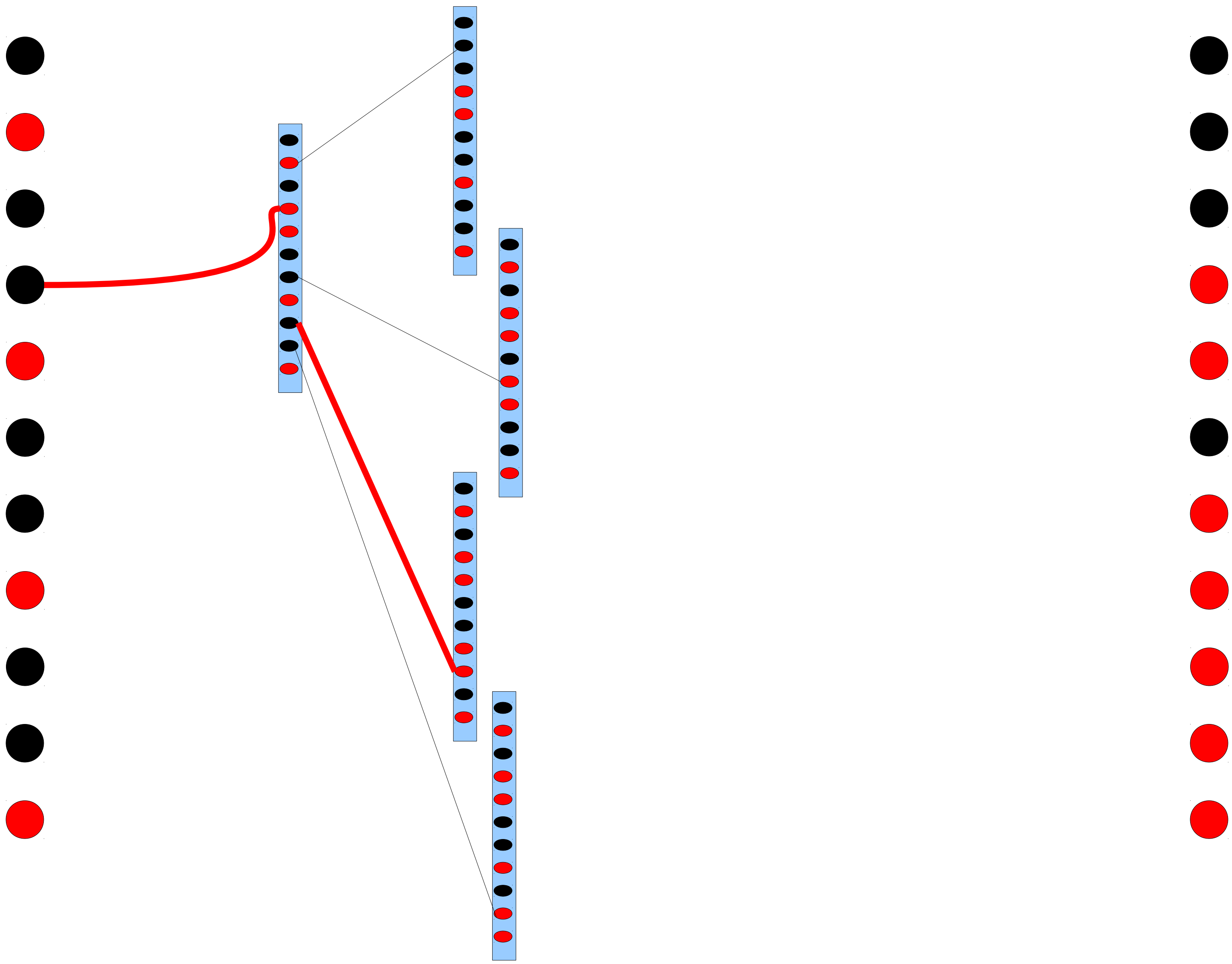
# PR example



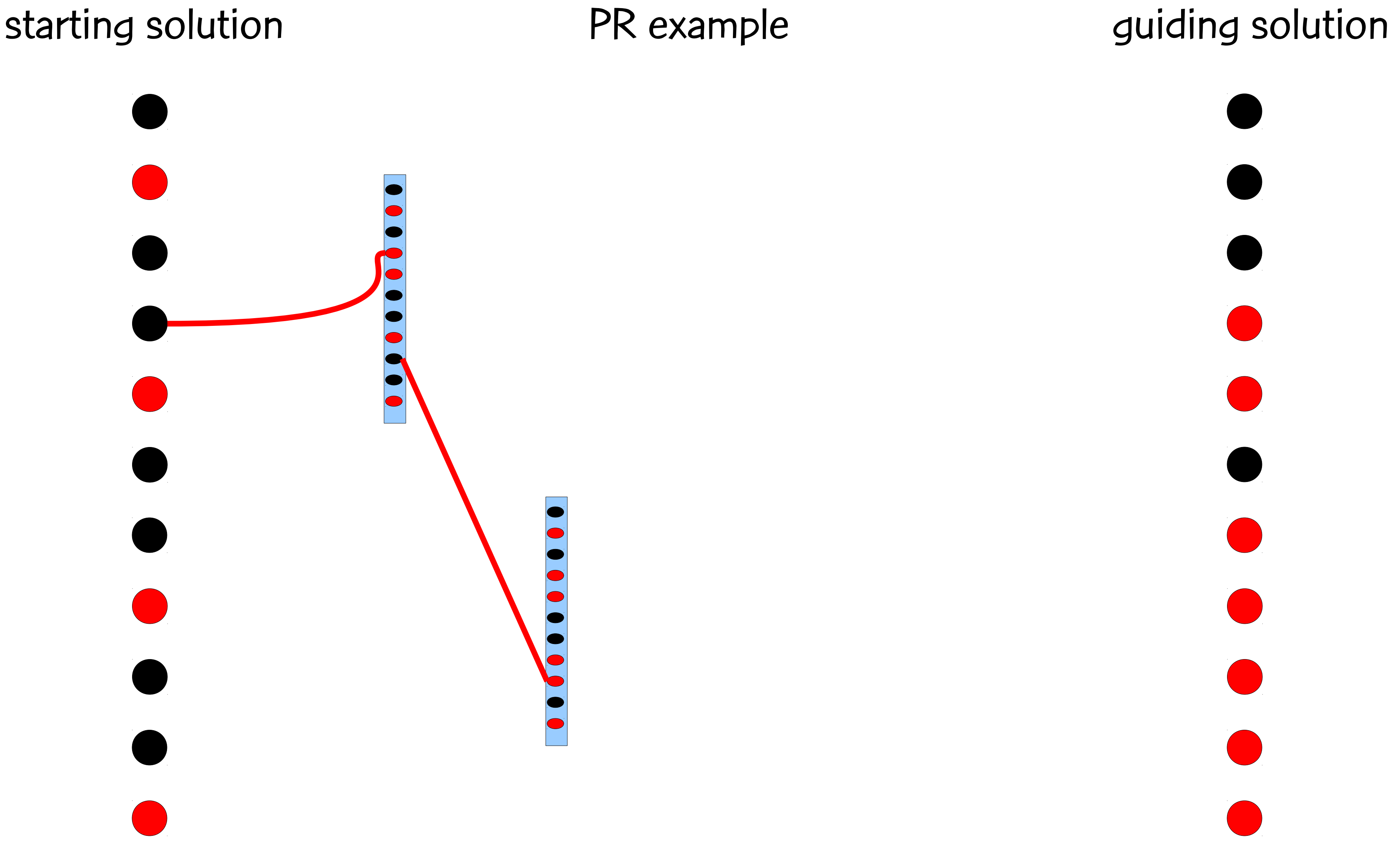
## PR example



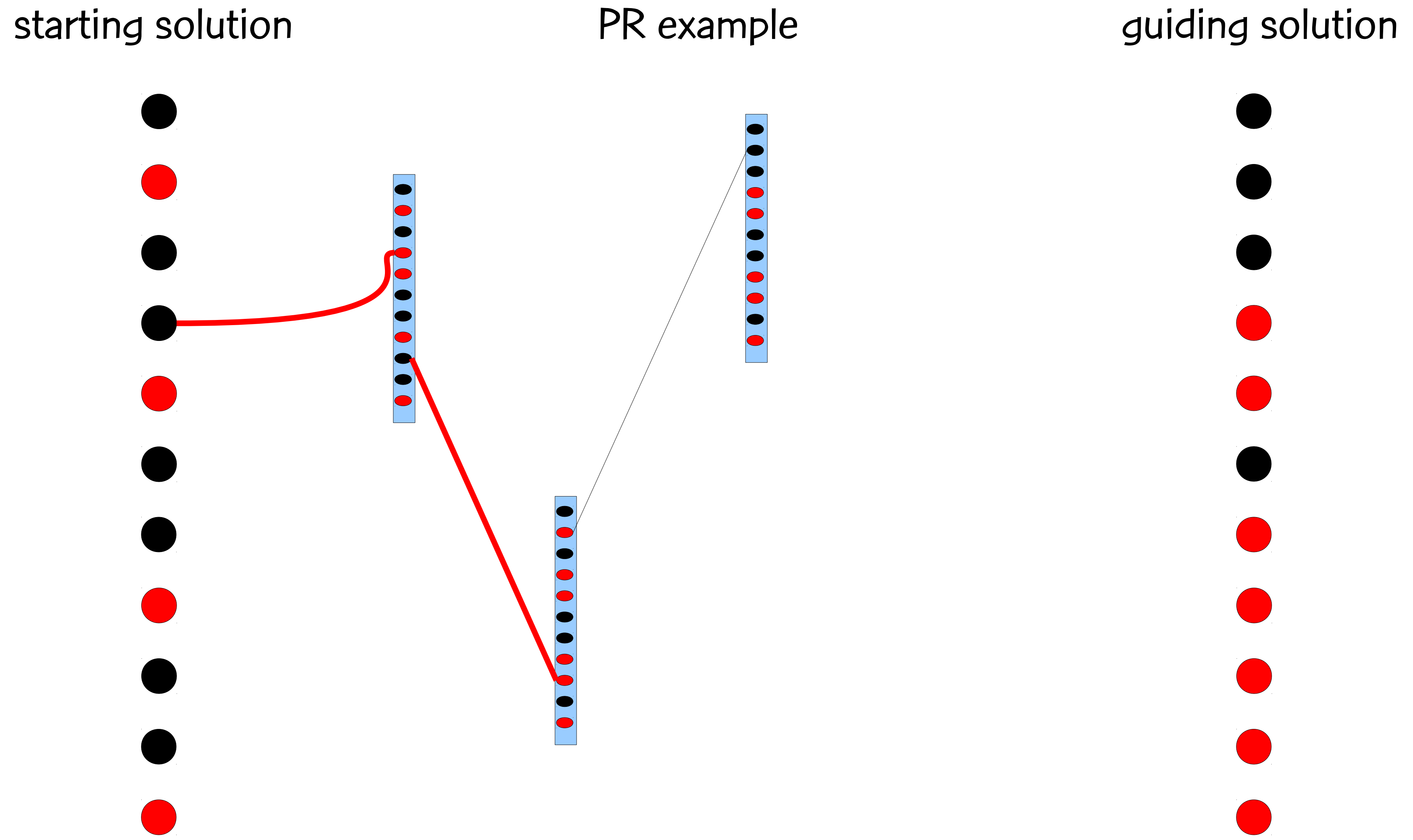
# starting solution PR example guiding solution



# PR example



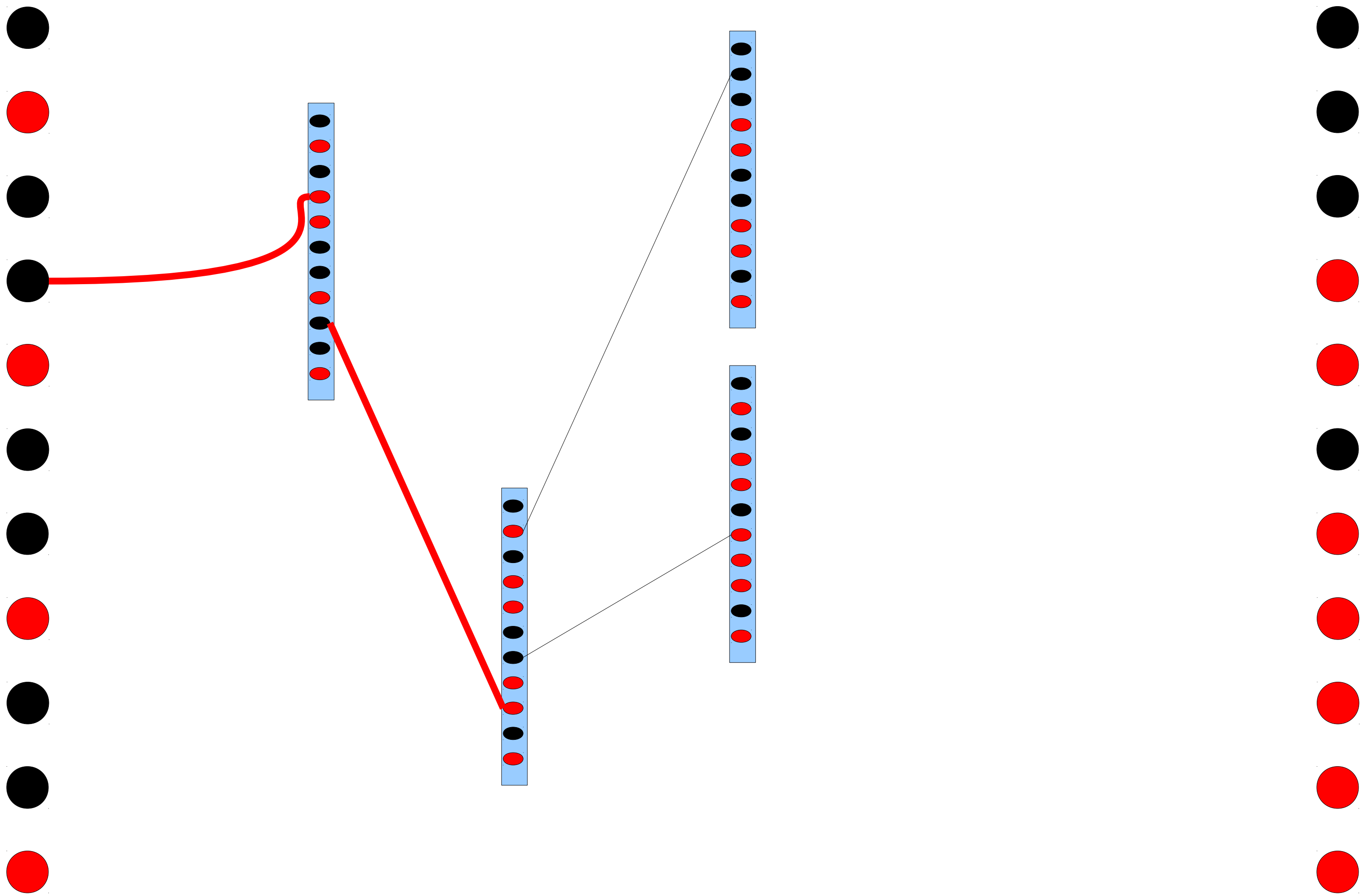
# PR example



# PR example

starting solution

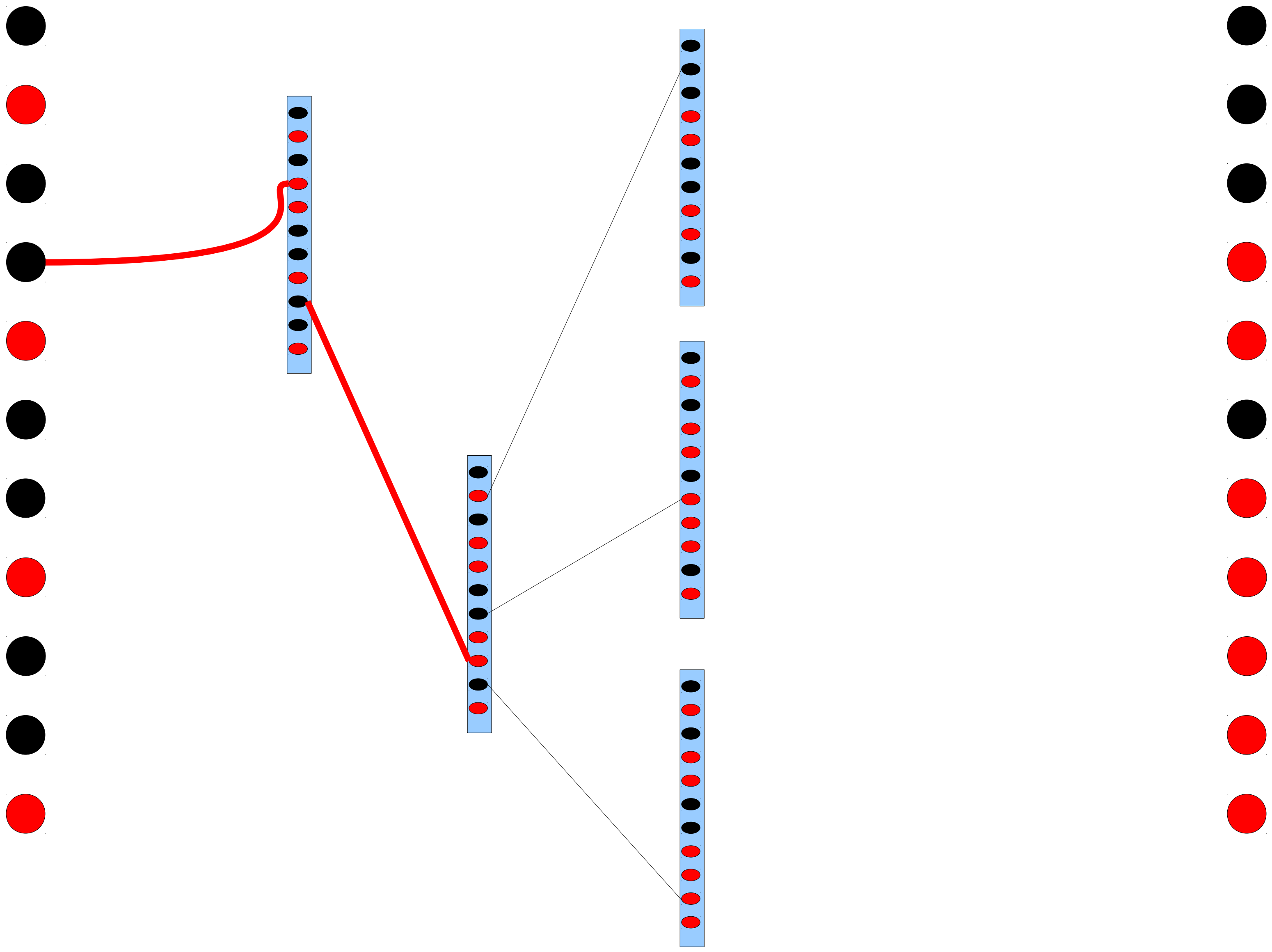
guiding solution



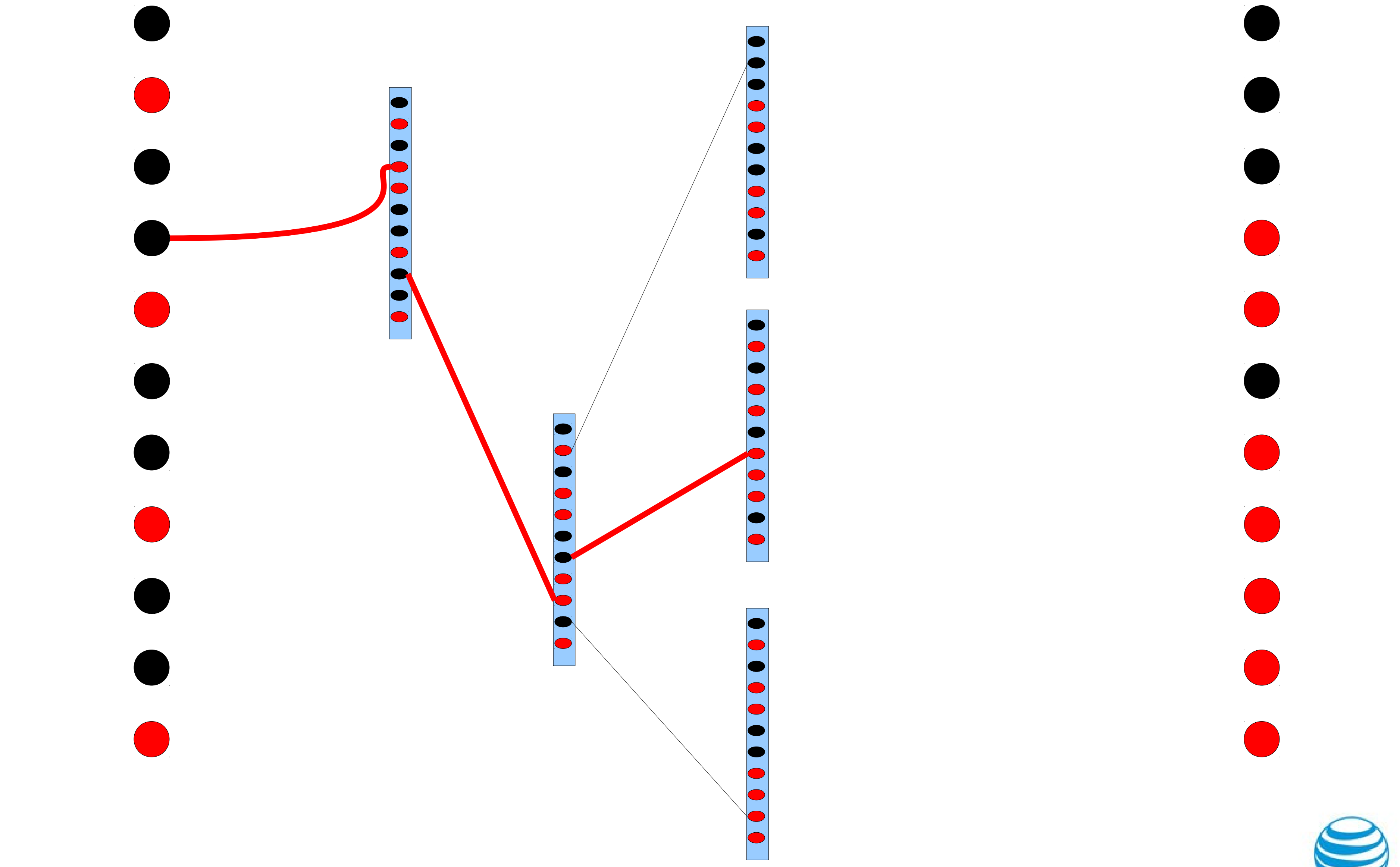
# PR example

starting solution

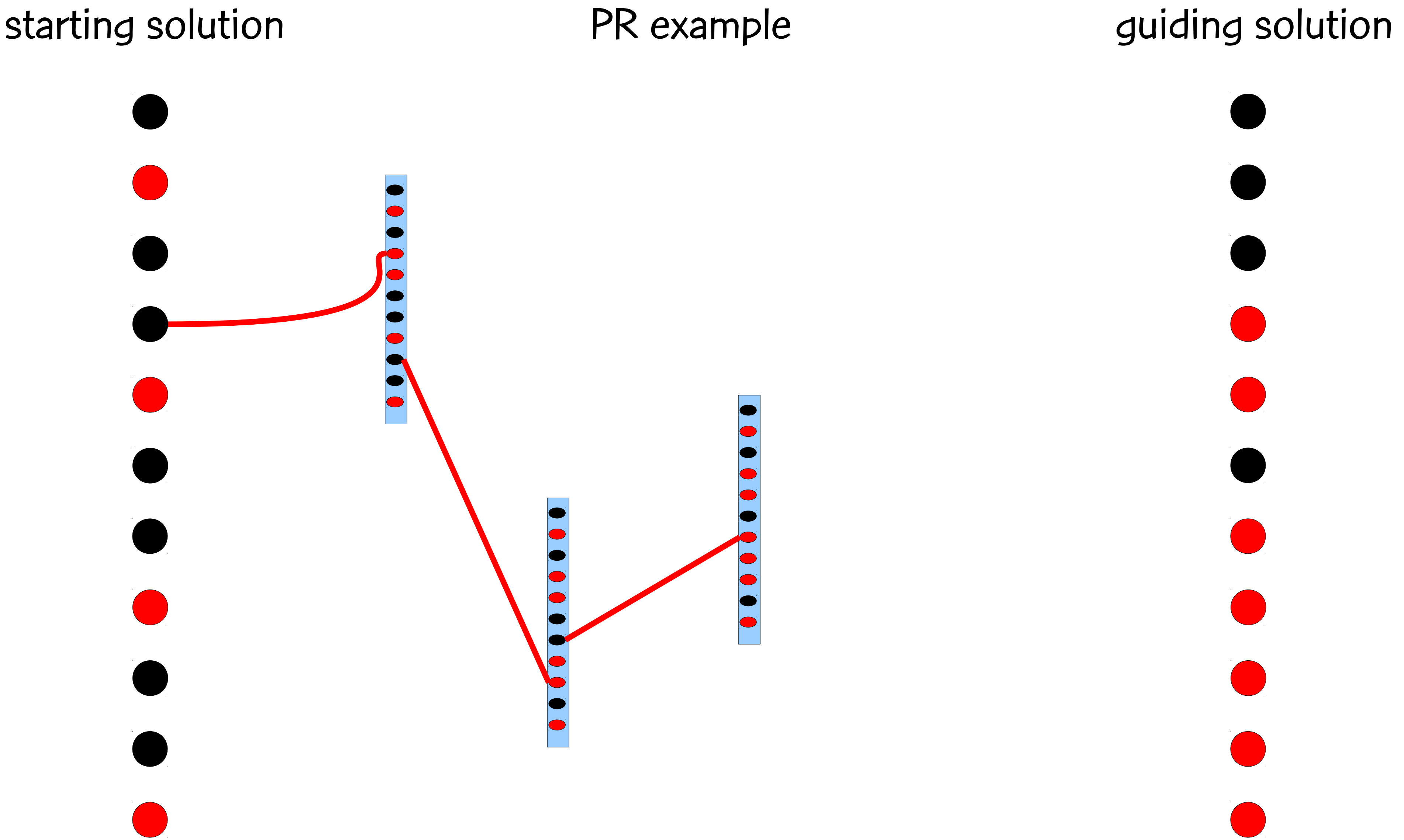
guiding solution



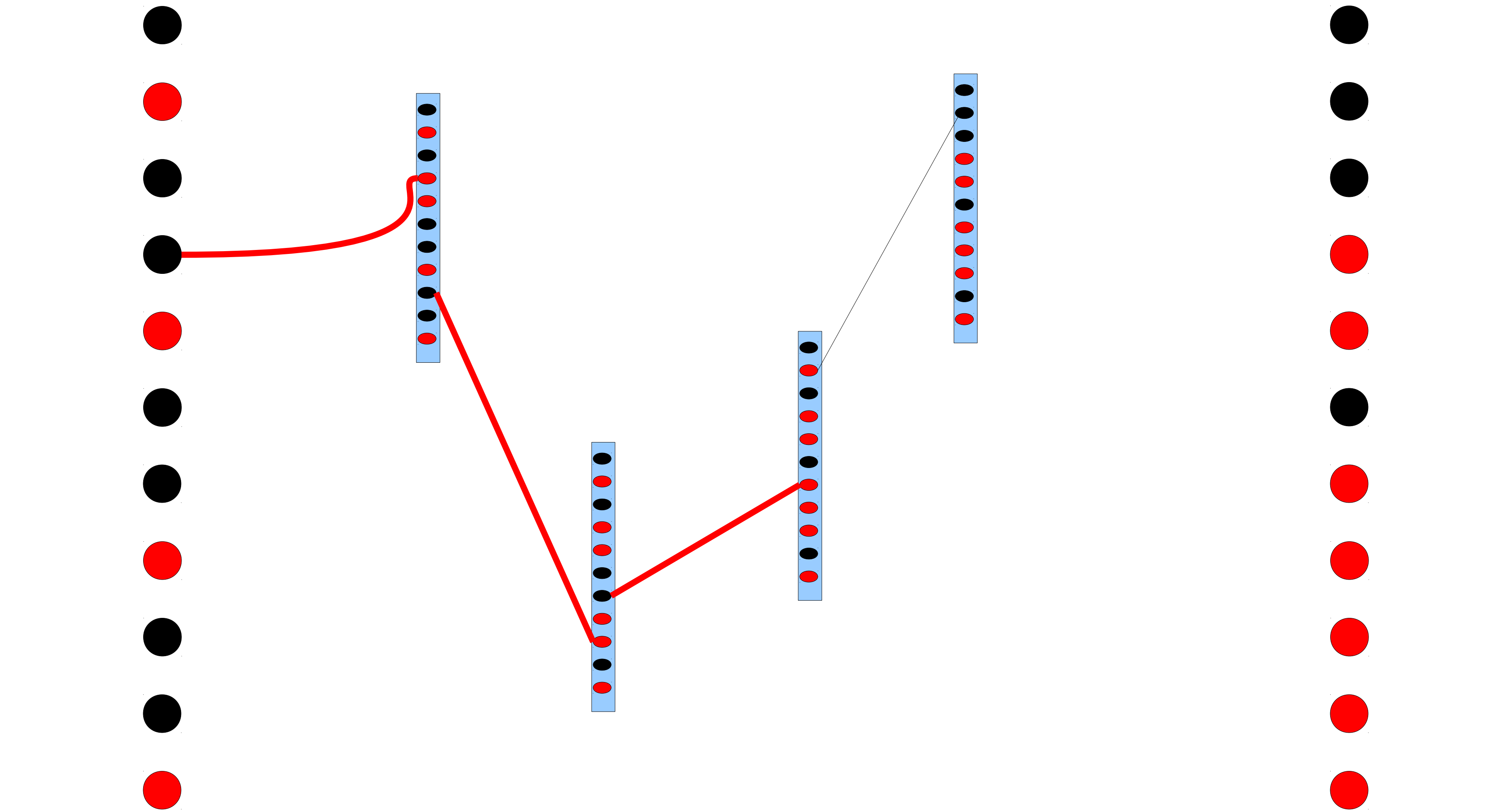
# PR example



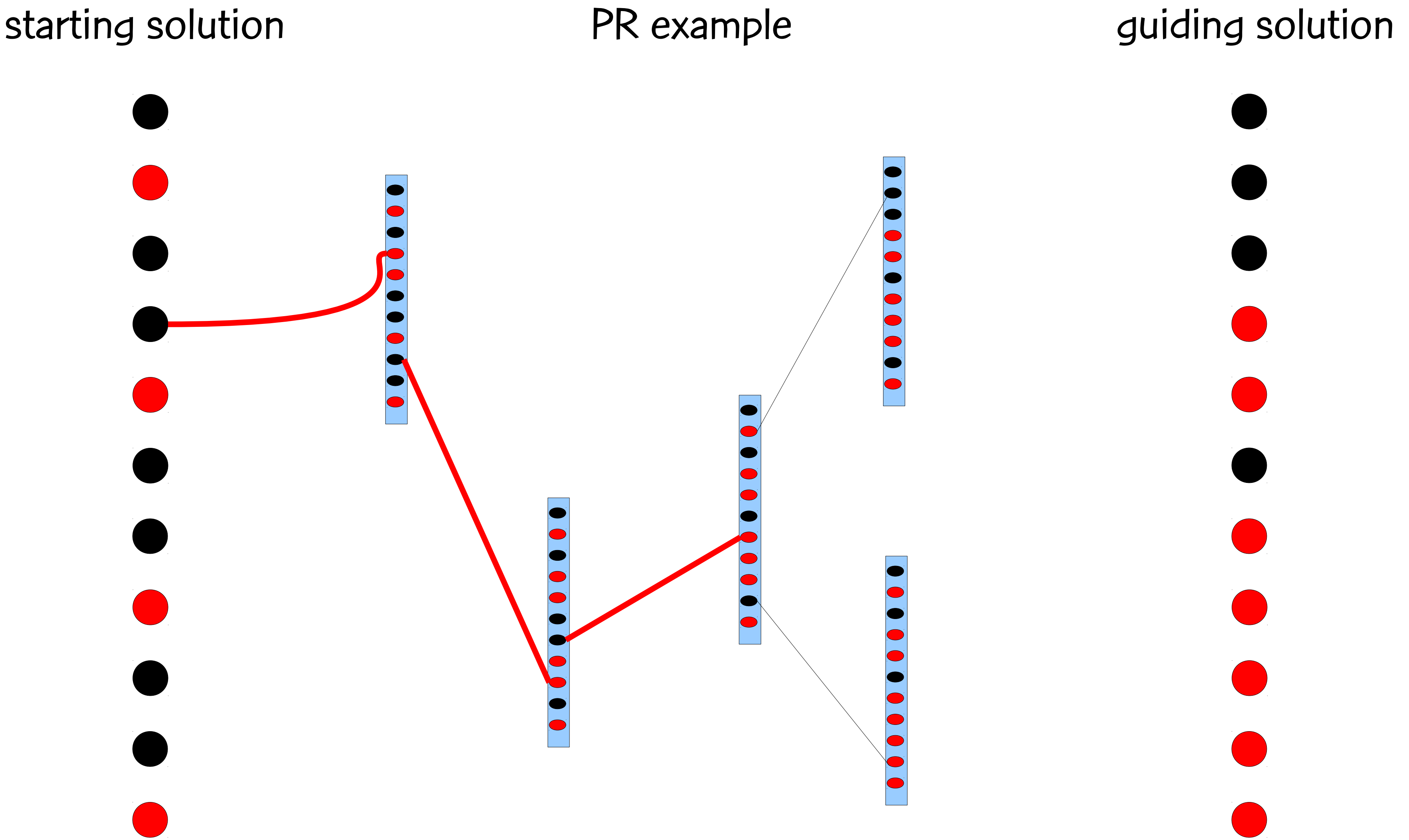
# PR example



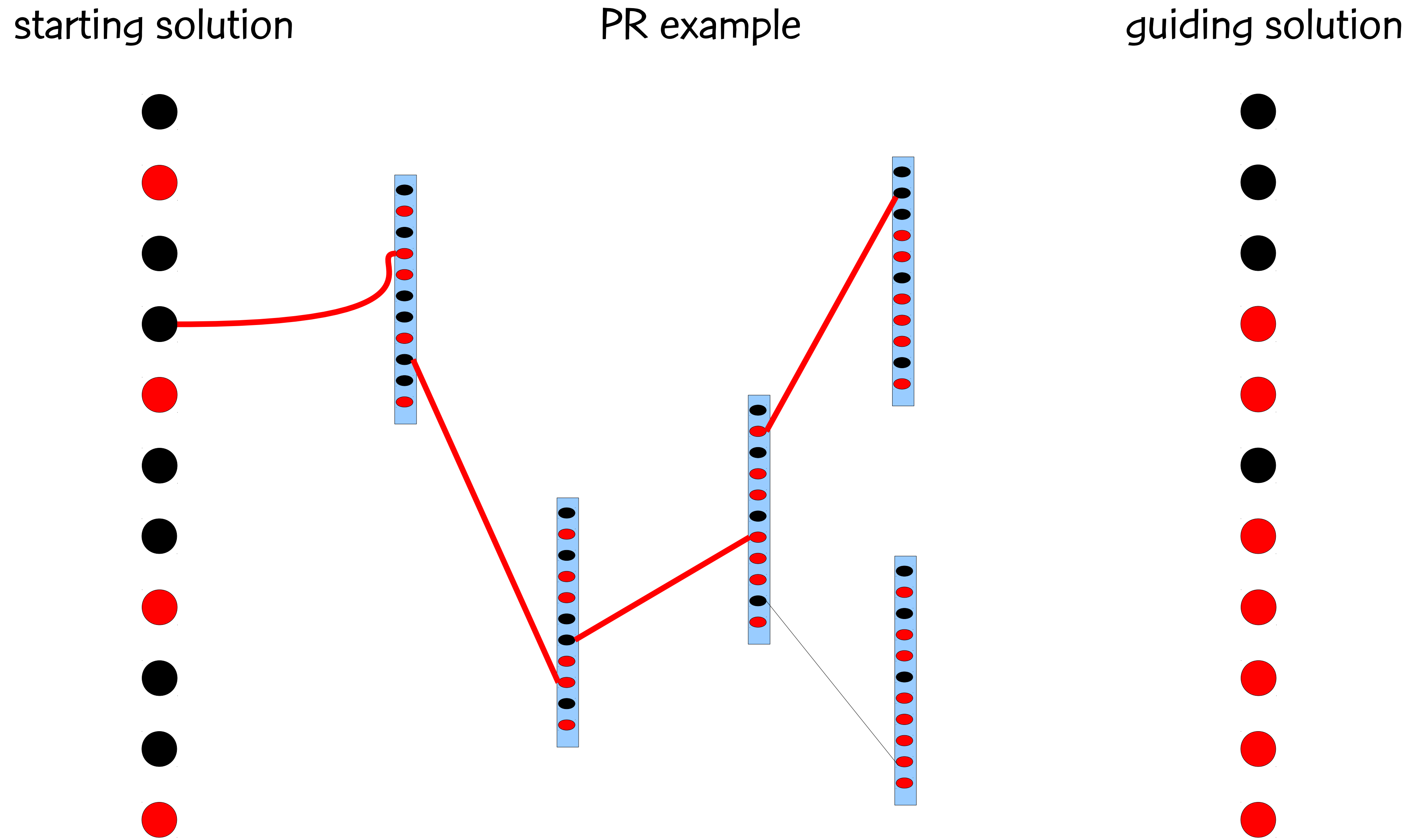
# PR example



# PR example



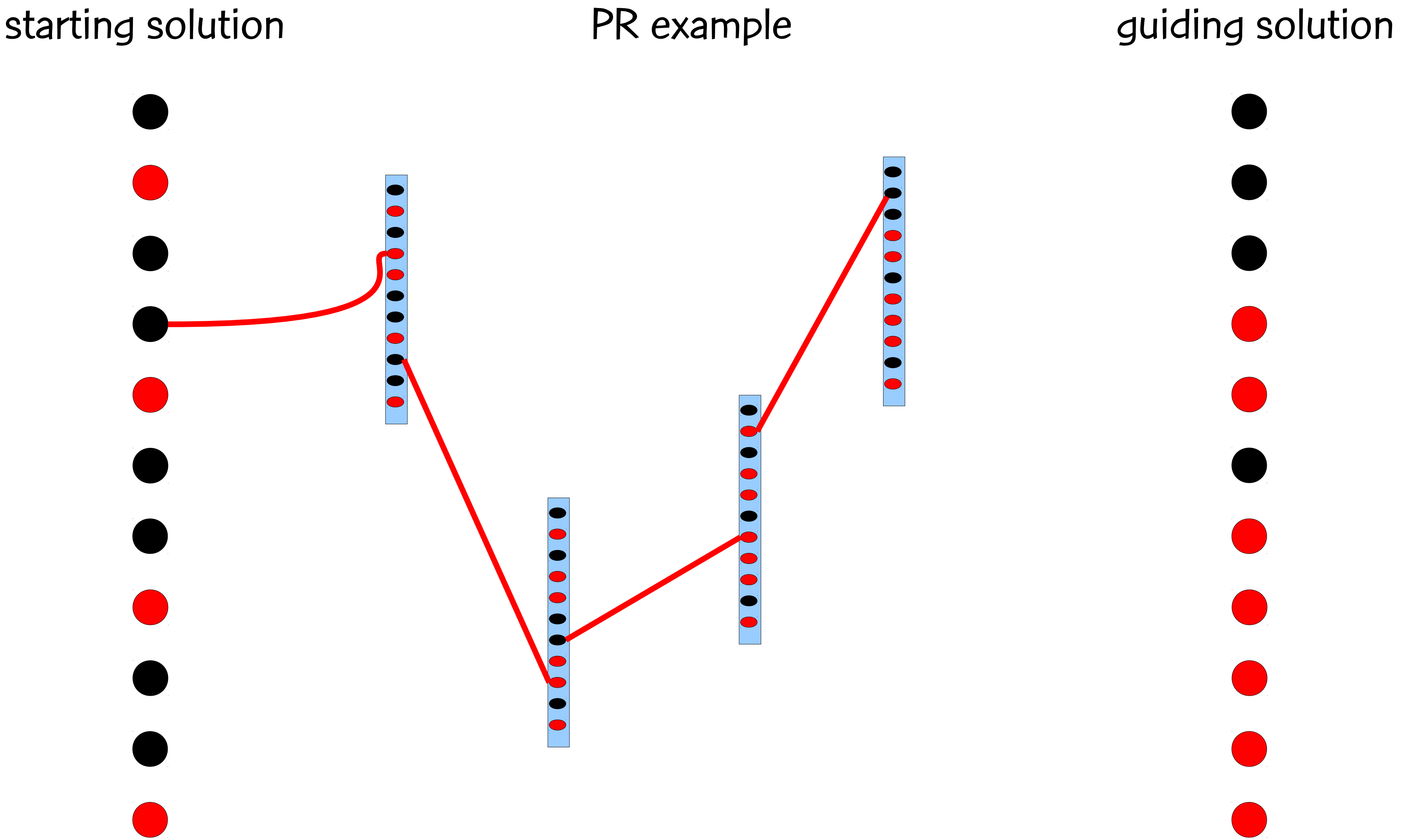
# PR example






at&t
Your world. Delivered.

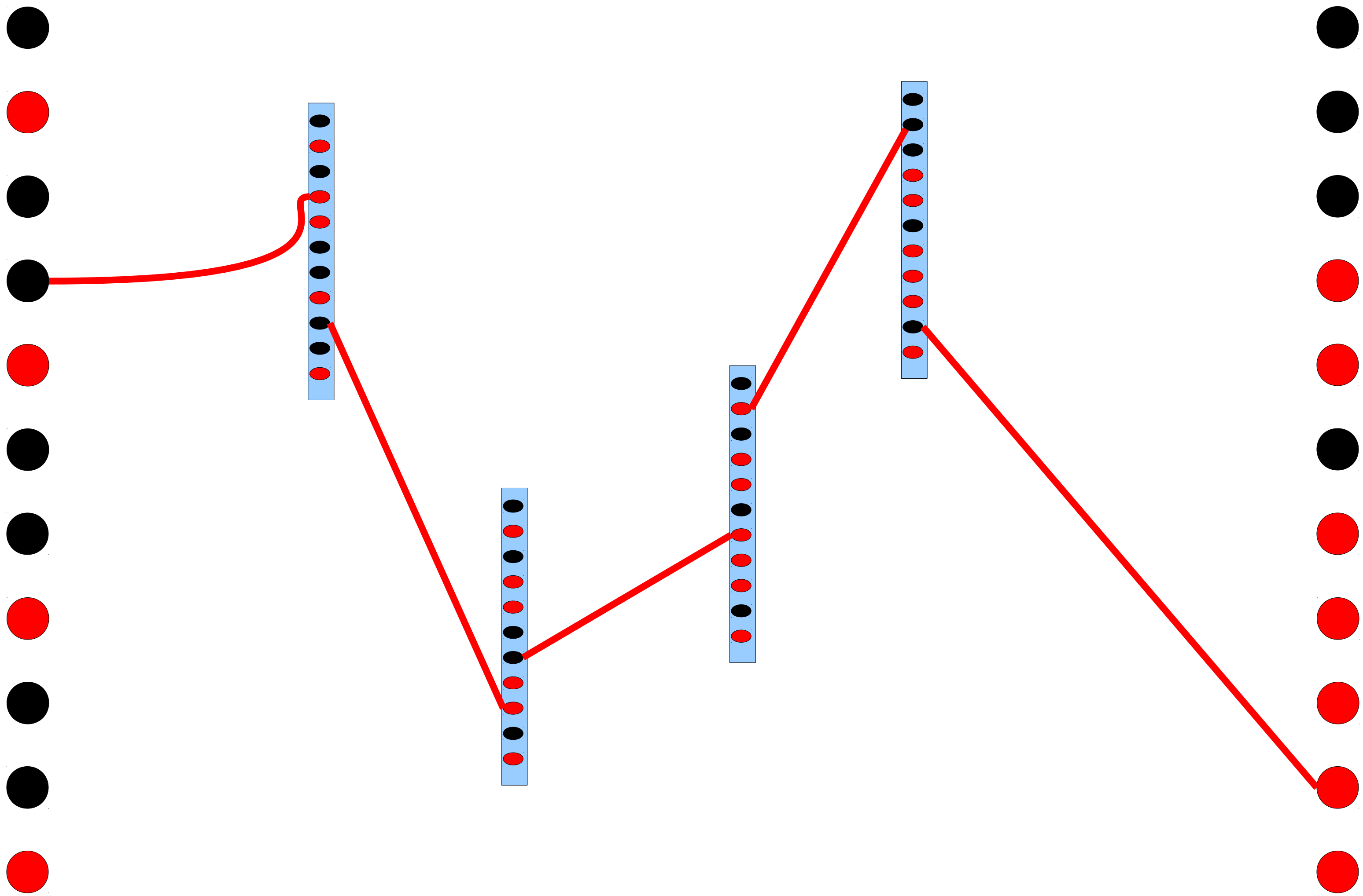
# PR example



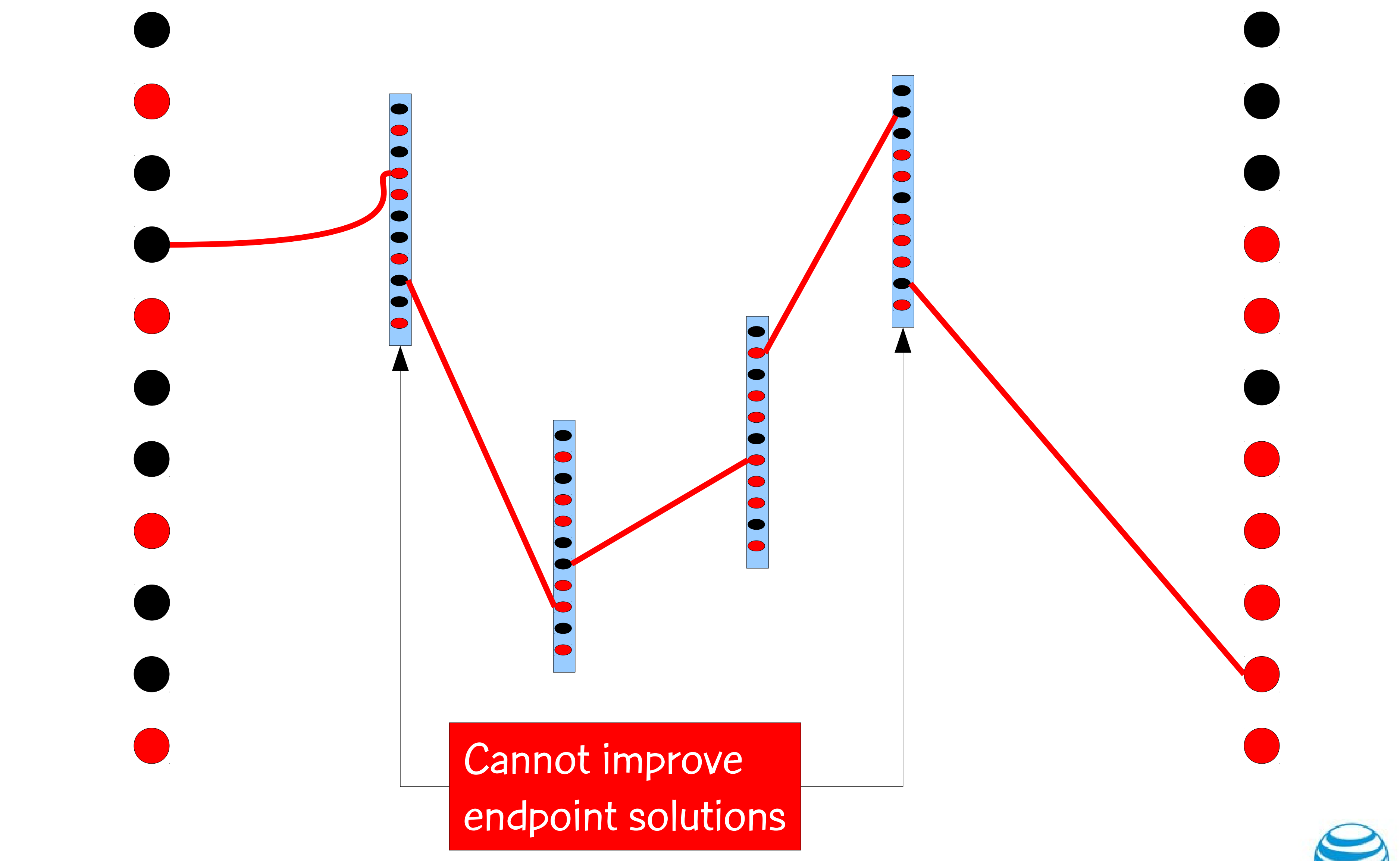
# PR example

starting solution

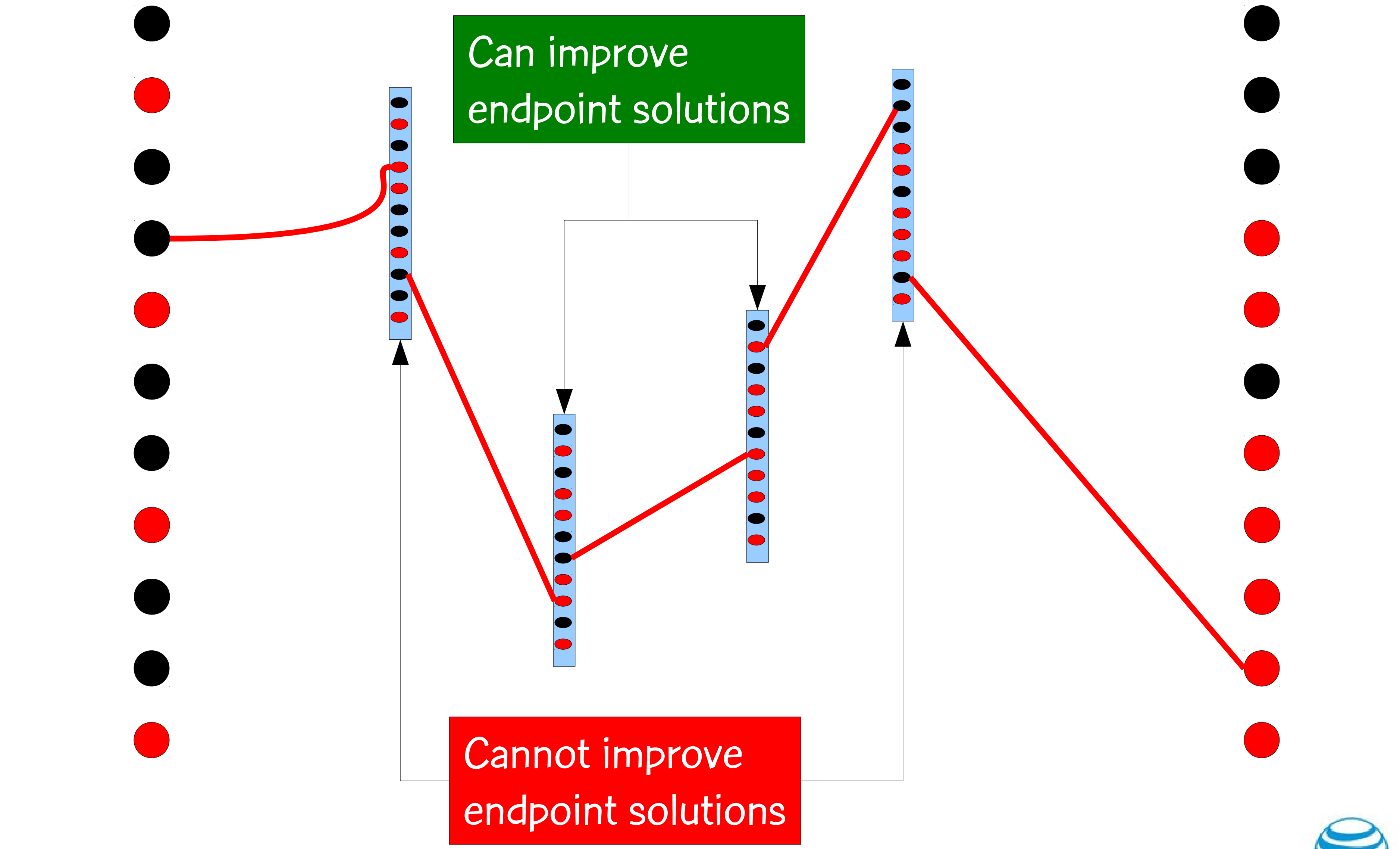
guiding solution



# PR example


starting solution
guiding solution
Cannot improve endpoint solutions

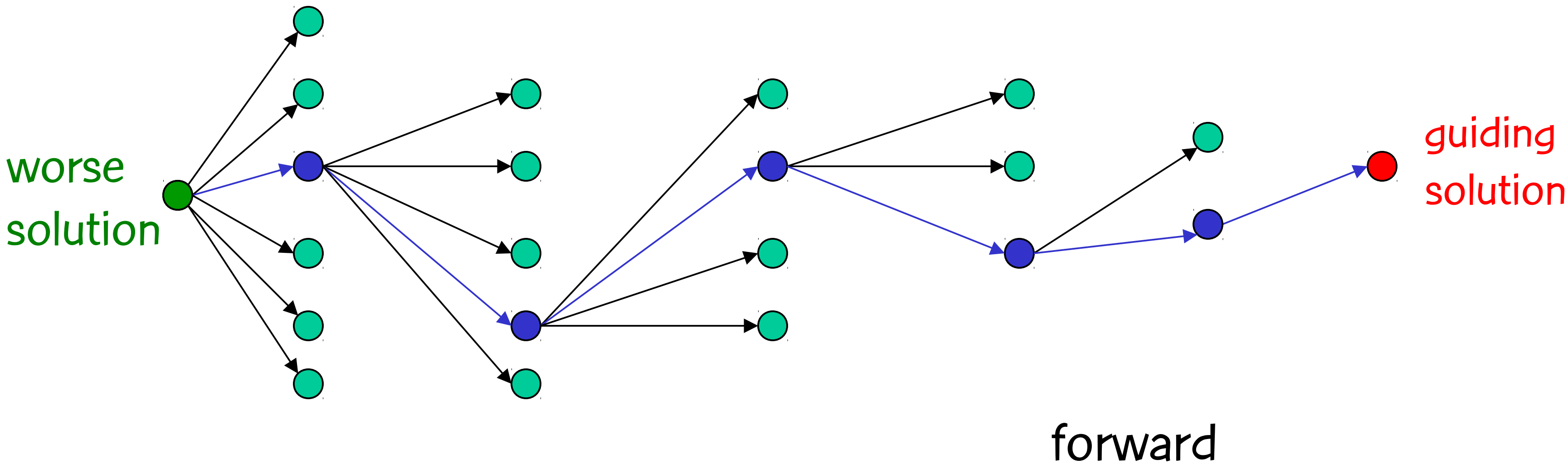
# PR example


starting solution
guiding solution
Can improve endpoint solutions
Cannot improve endpoint solutions

# Forward path-relinking

## Variants: trade-offs between computation time and solution quality

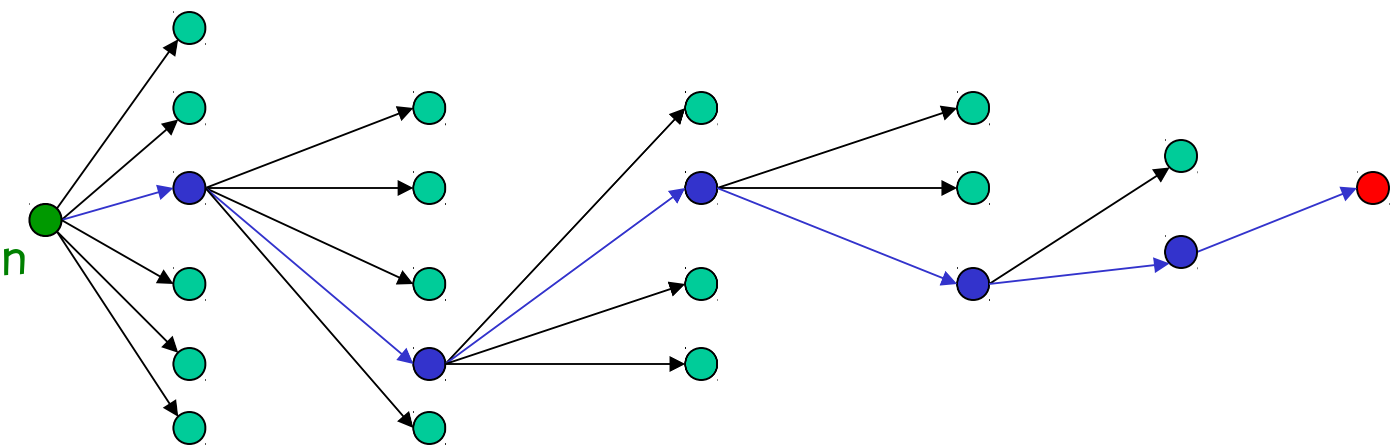
Forward PR adopts as initial solution the worse of the two input solutions and uses the better solution as the guide.



# Backward path-relinking

Variants: trade-offs between computation time and solution quality

Backward PR usually does better: Better to start from the best of the two input solutions, neighborhood of the initial solution is explored more than of the guide!

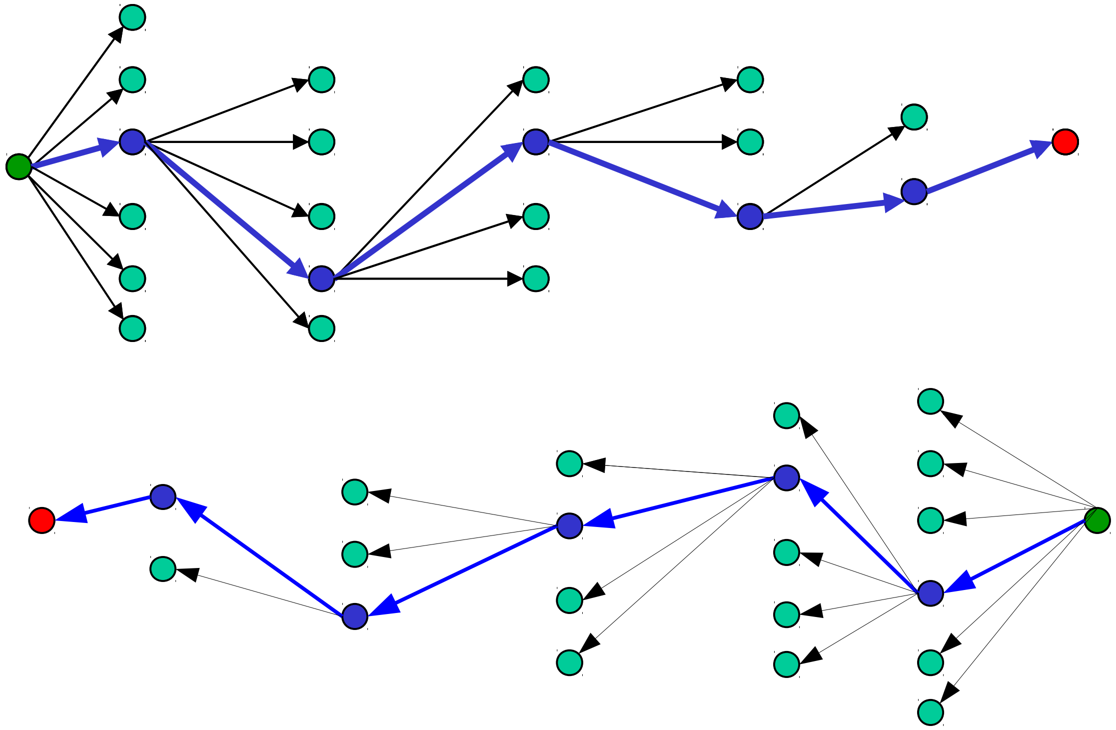





at&t
Your world. Delivered.

# Back and forth path-relinking

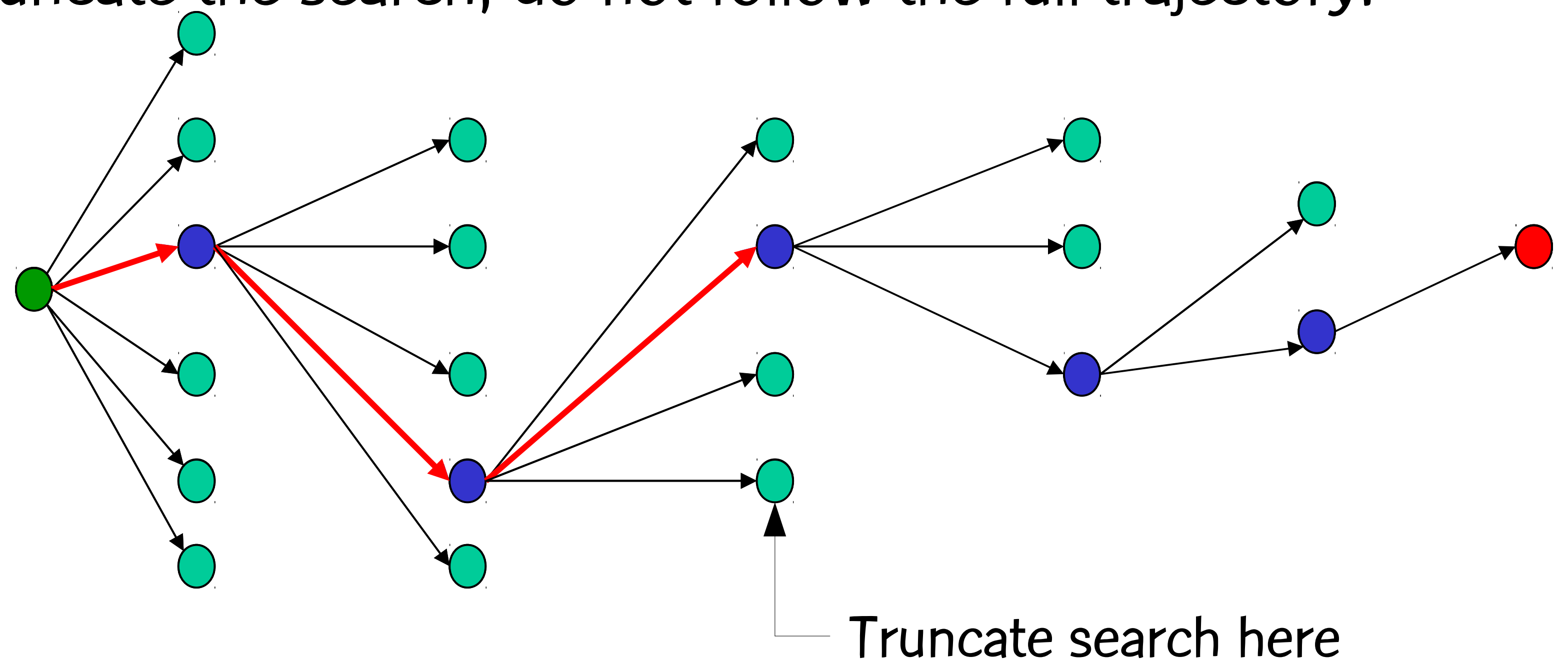
- Variants: trade-offs between computation time and solution quality
  - Explore both trajectories: twice as much time, often with only marginal improvements!



# Truncated path-relinking

## Variants: trade-offs between computation time and solution quality

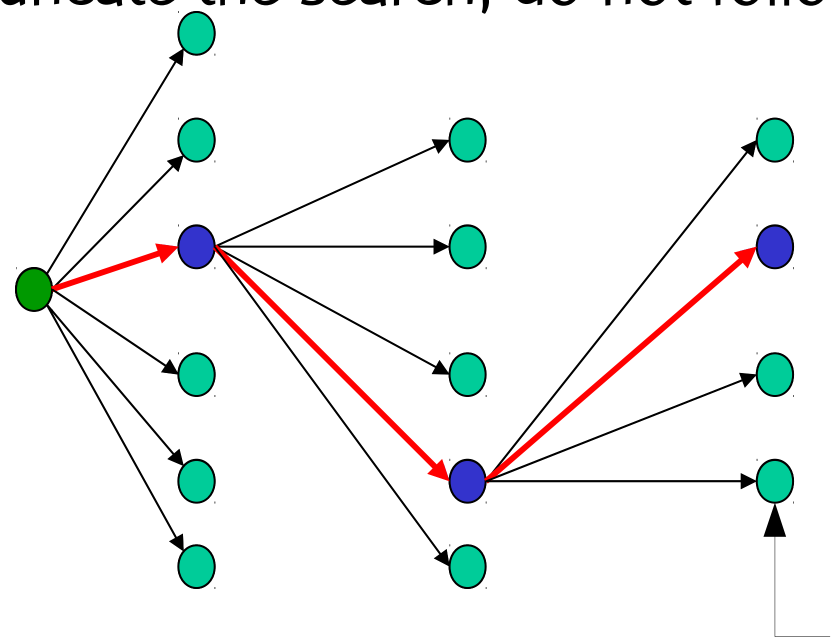
Truncate the search, do not follow the full trajectory.



# Truncated path-relinking

## Variants: trade-offs between computation time and solution quality

Truncate the search, do not follow the full trajectory.



Truncate search here

# Mixed path-relinking

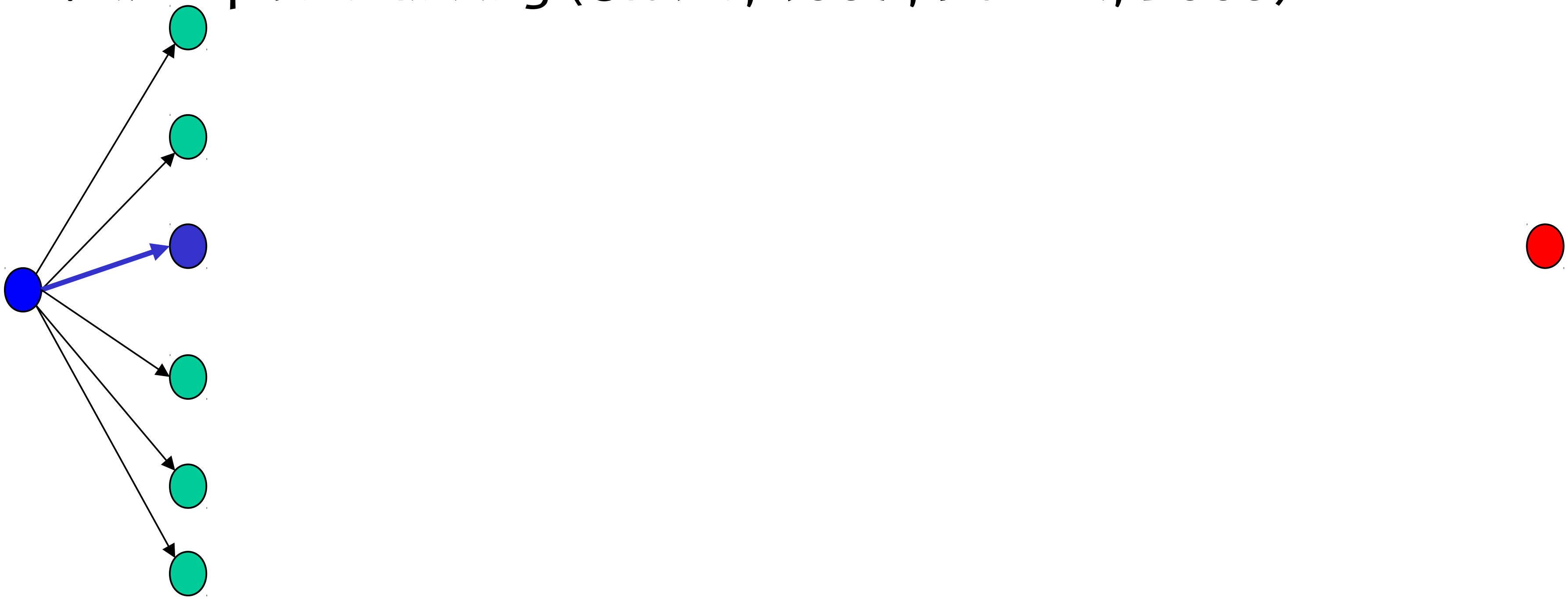
Variants: trade-offs between computation time and solution quality

Mixed path-relinking (Glover, 1997; Rosseti, 2003)

# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

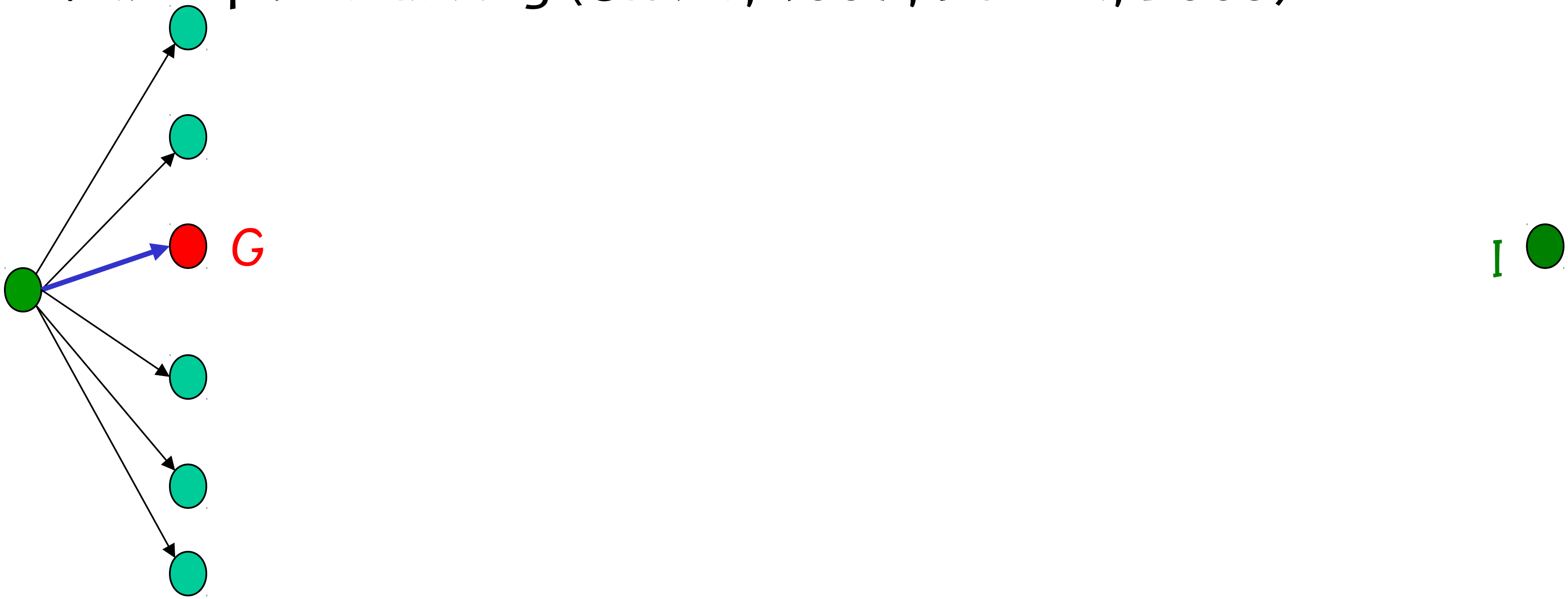
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

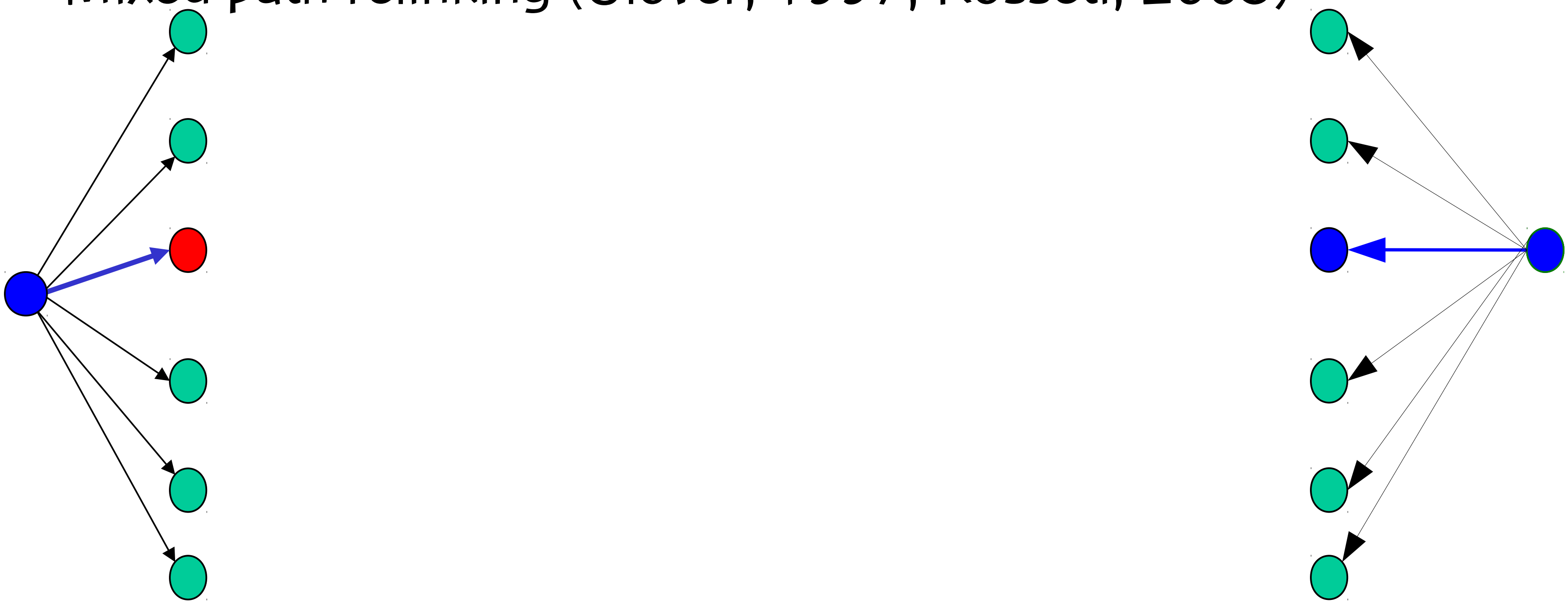
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

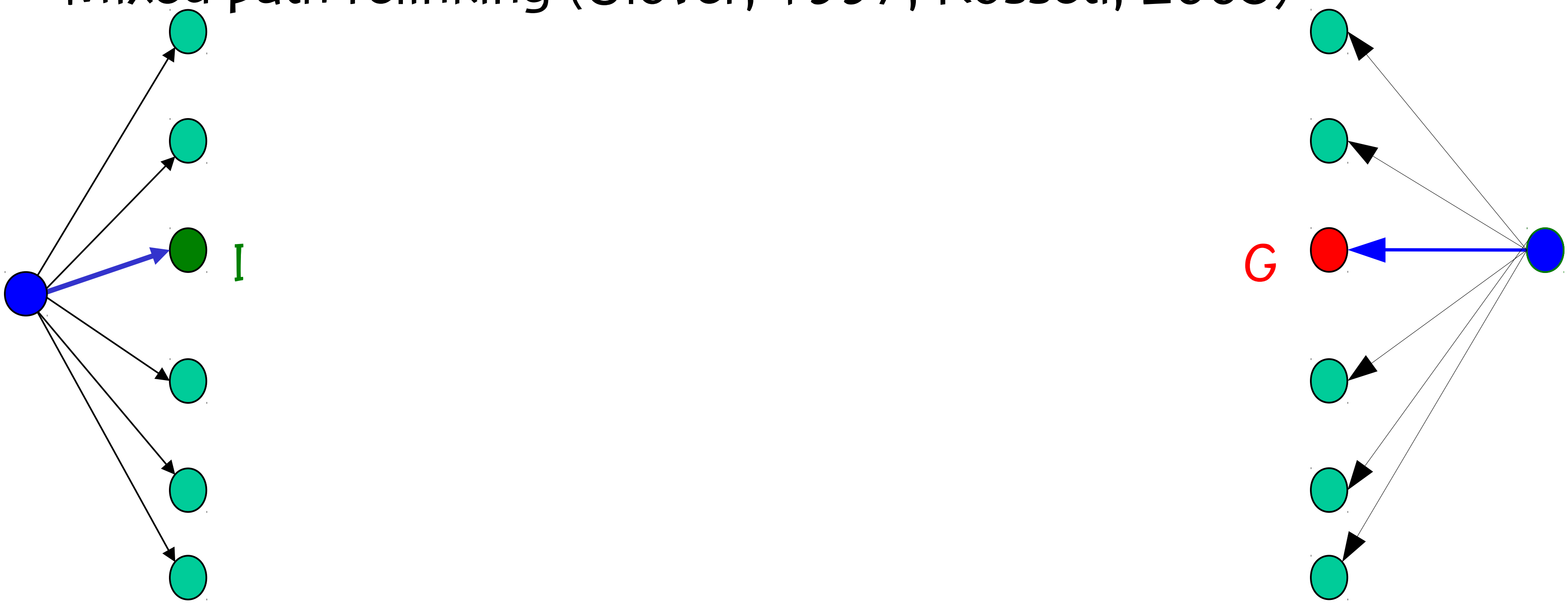
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

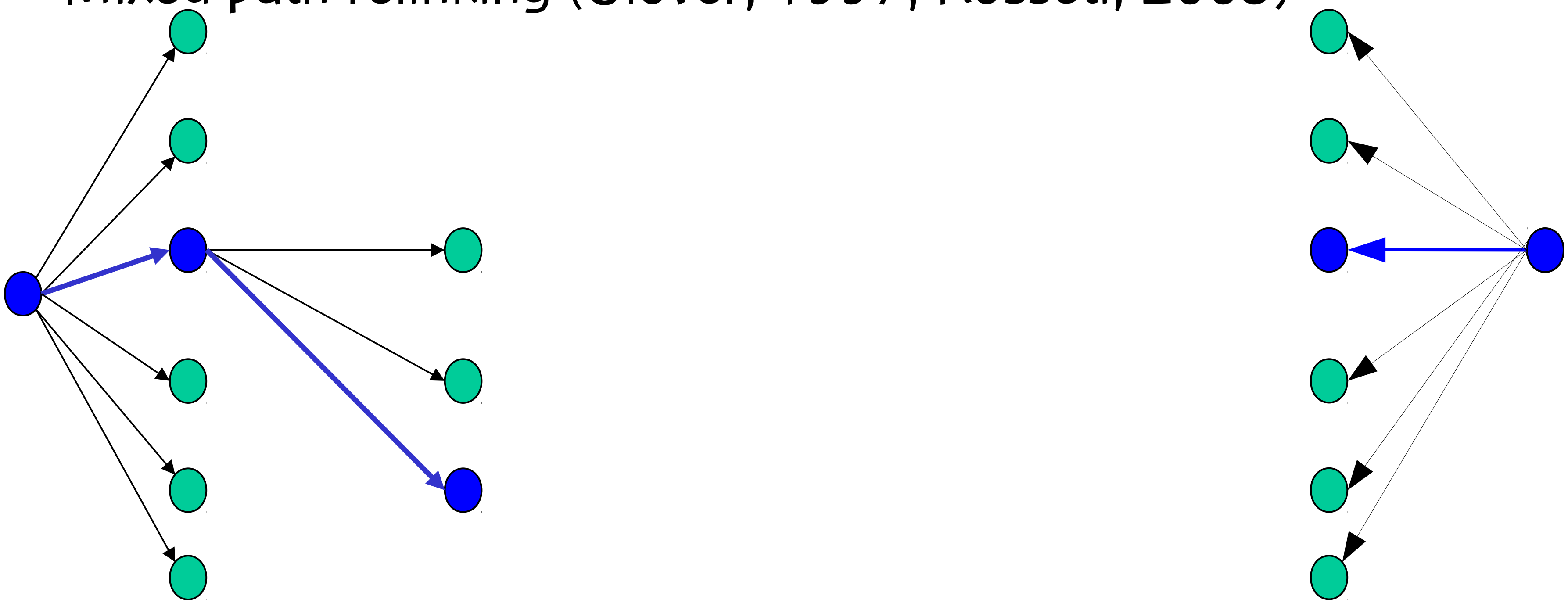
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

## Variants: trade-offs between computation time and solution quality

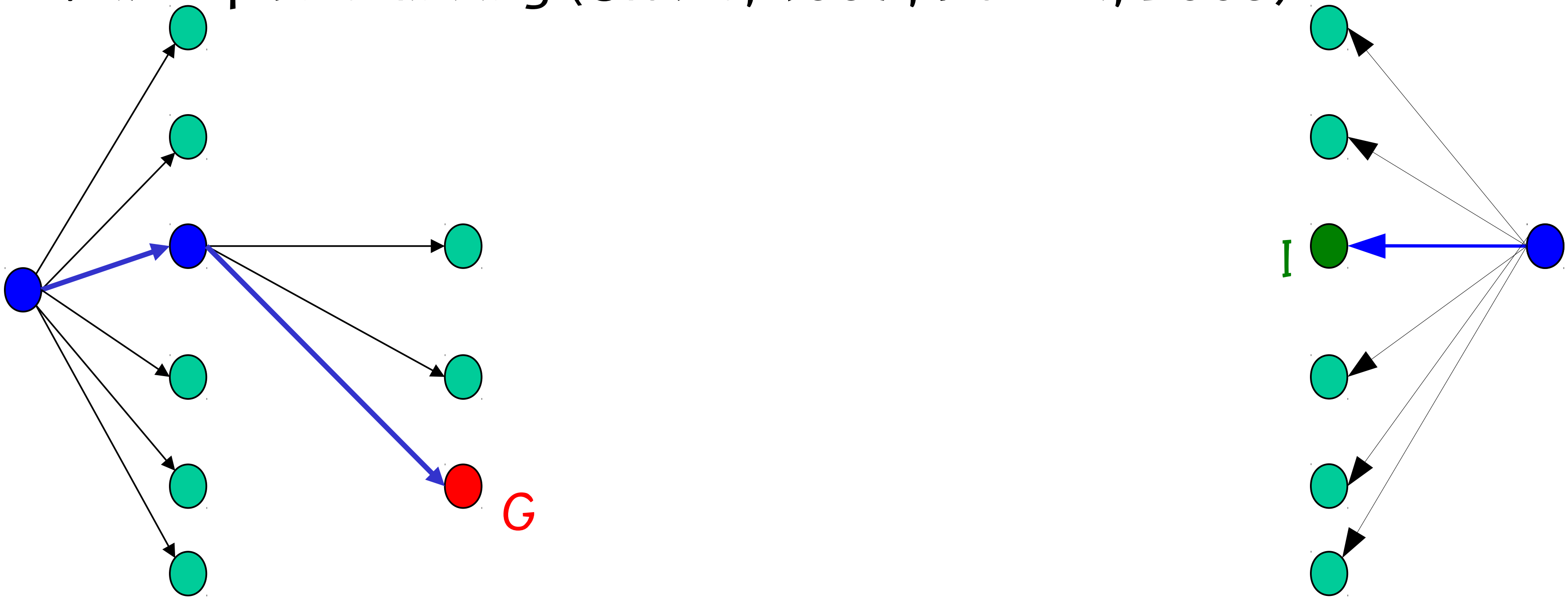
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

## Variants: trade-offs between computation time and solution quality

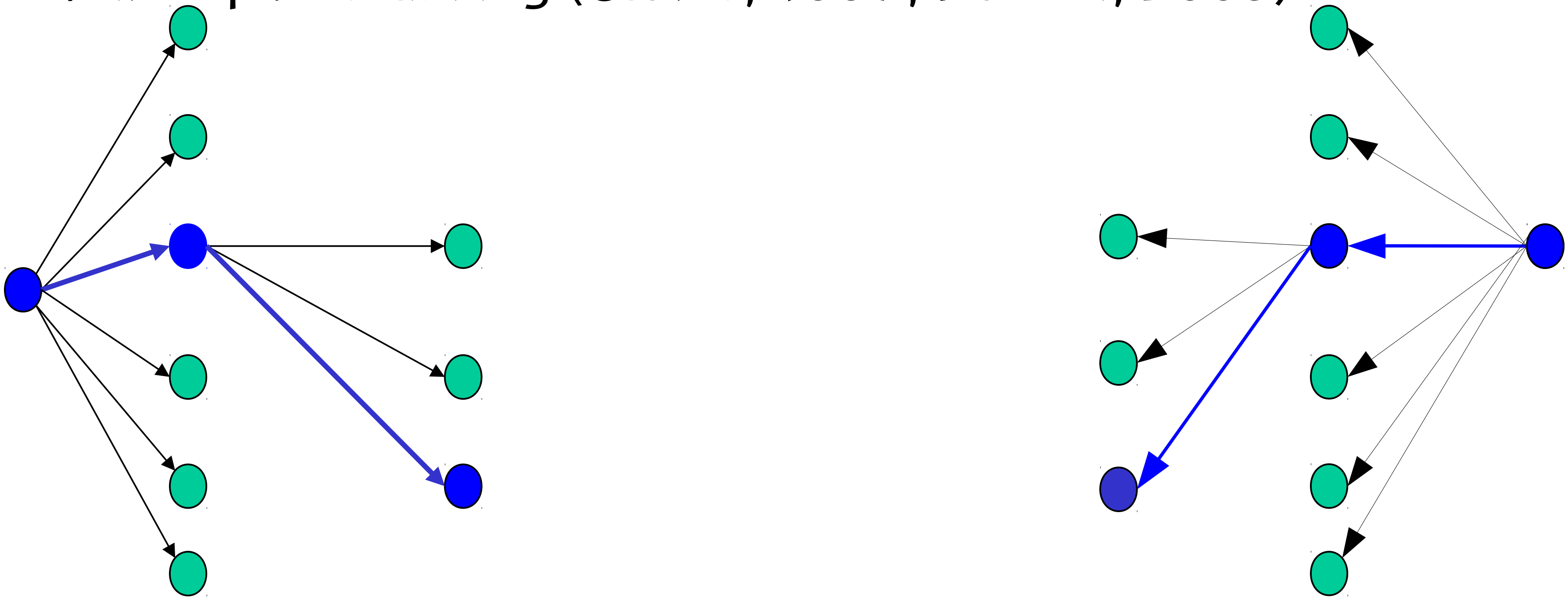
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

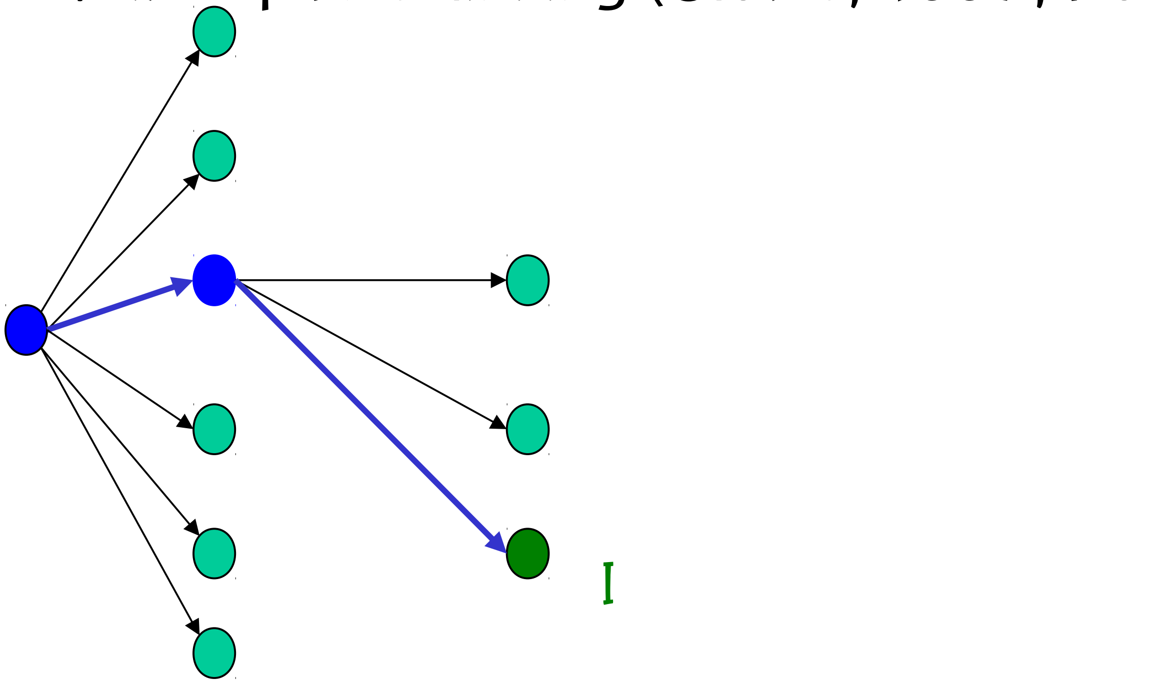
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

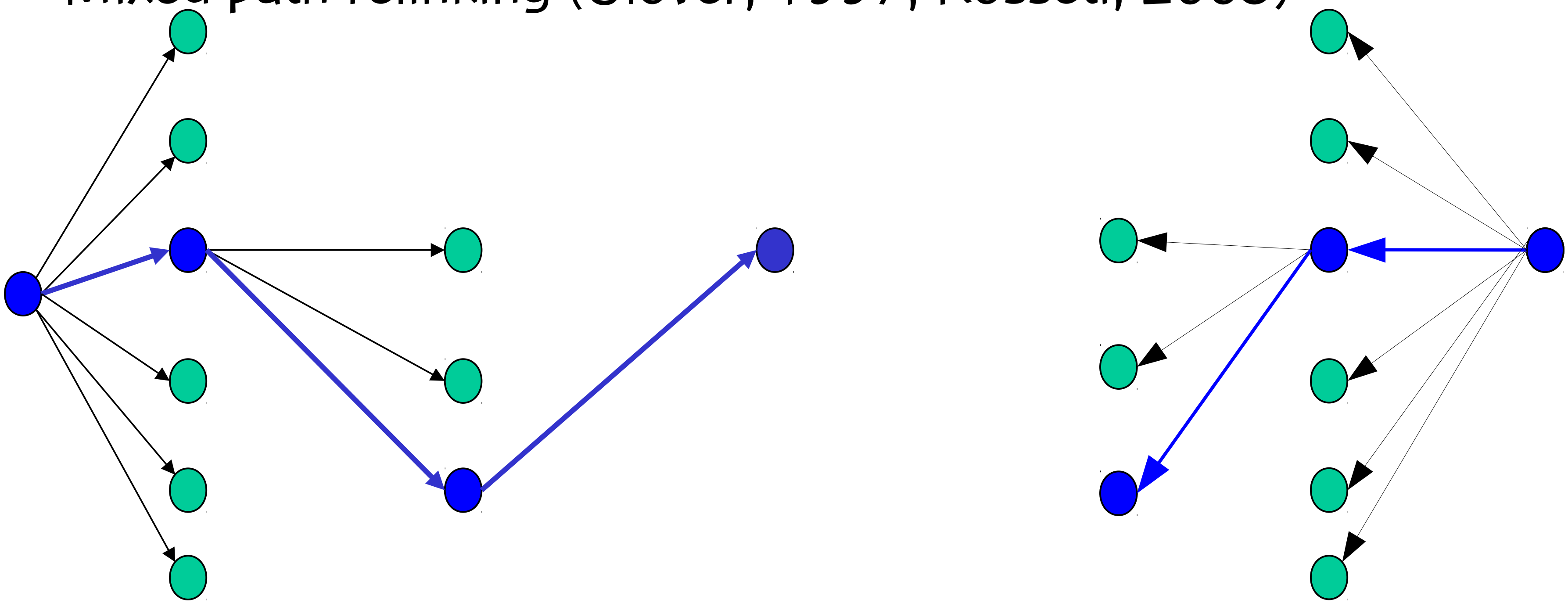
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

## Variants: trade-offs between computation time and solution quality

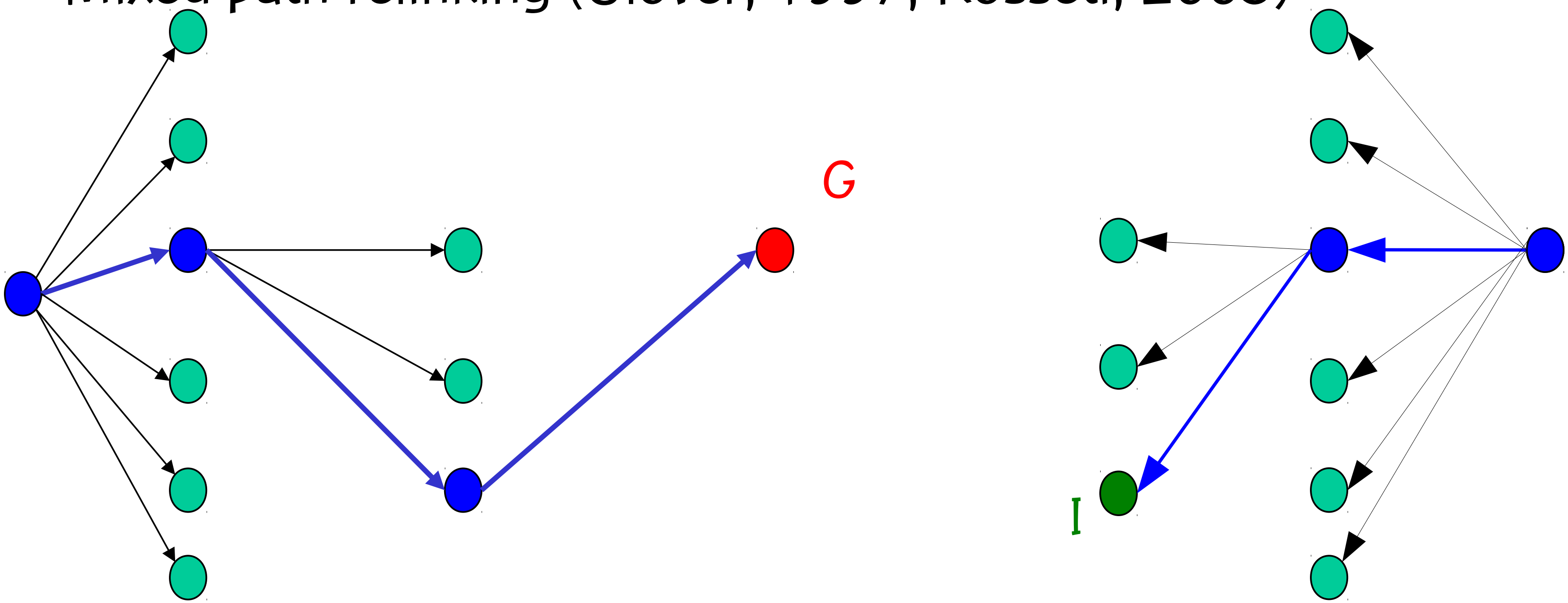
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

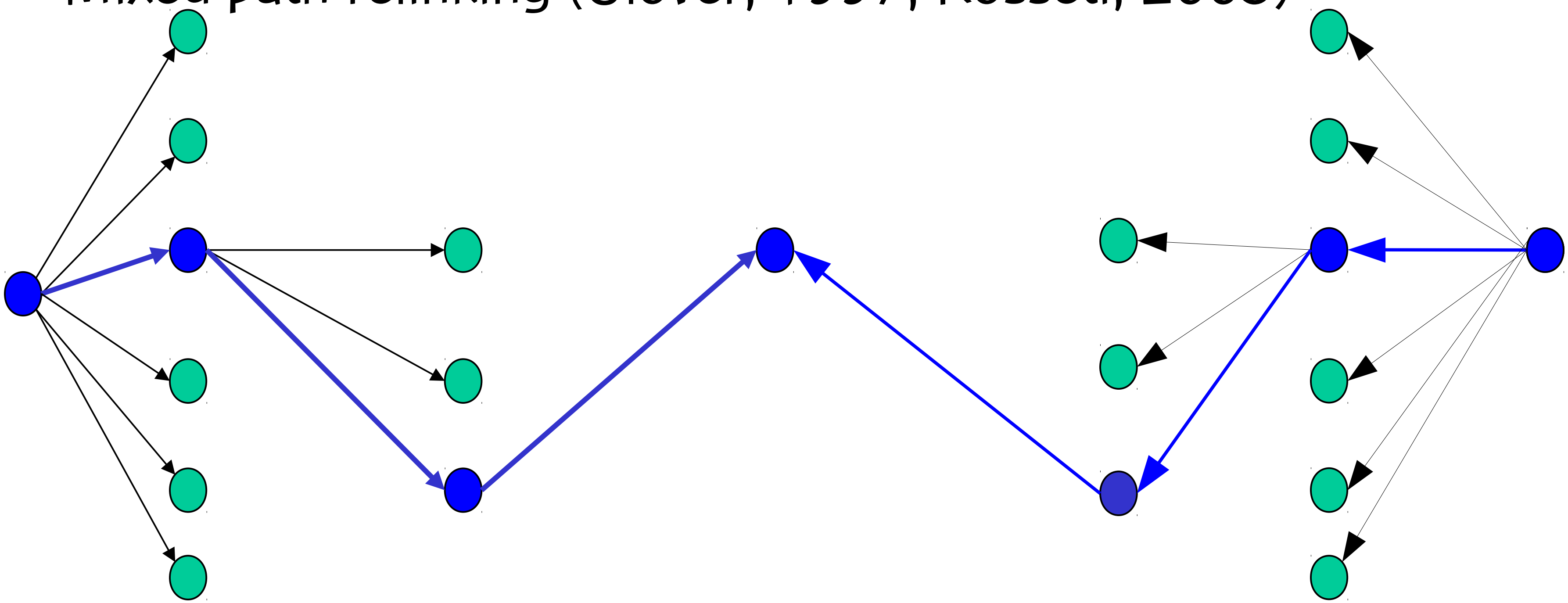
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

## Variants: trade-offs between computation time and solution quality

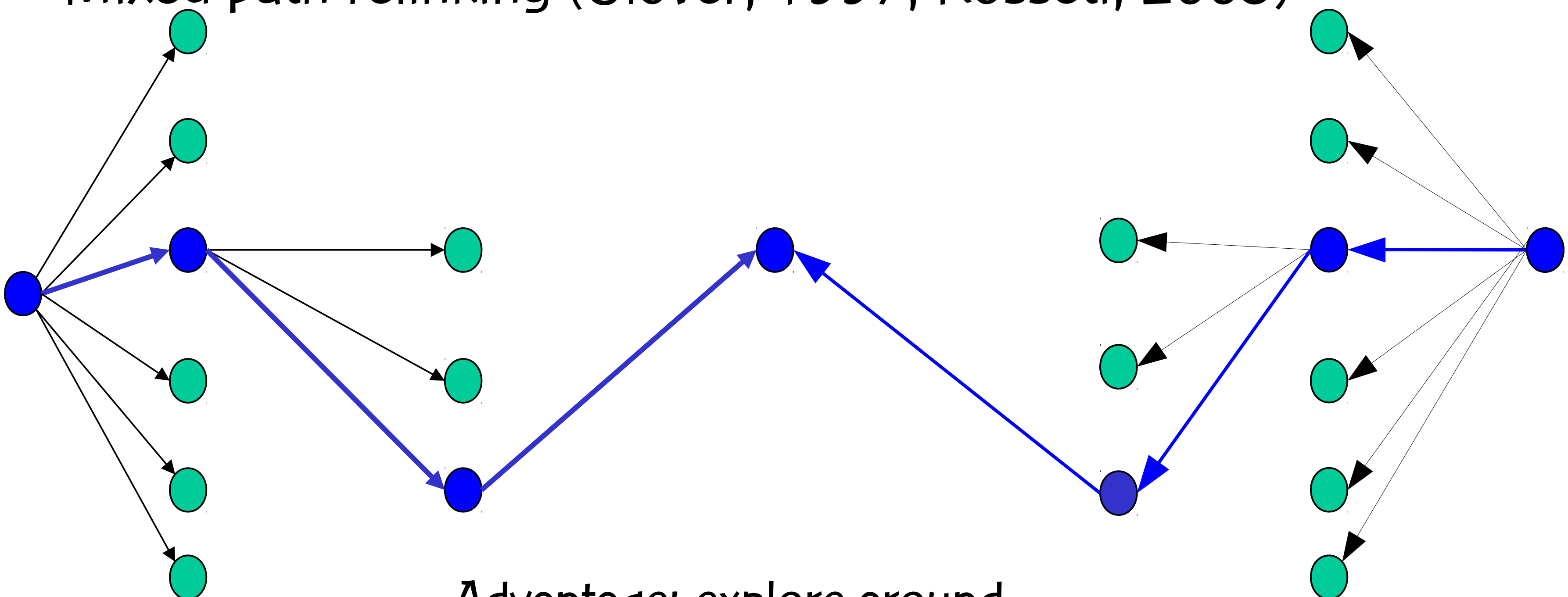
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



# Mixed path-relinking

Variants: trade-offs between computation time and solution quality

Mixed path-relinking (Glover, 1997; Rosseti, 2003)

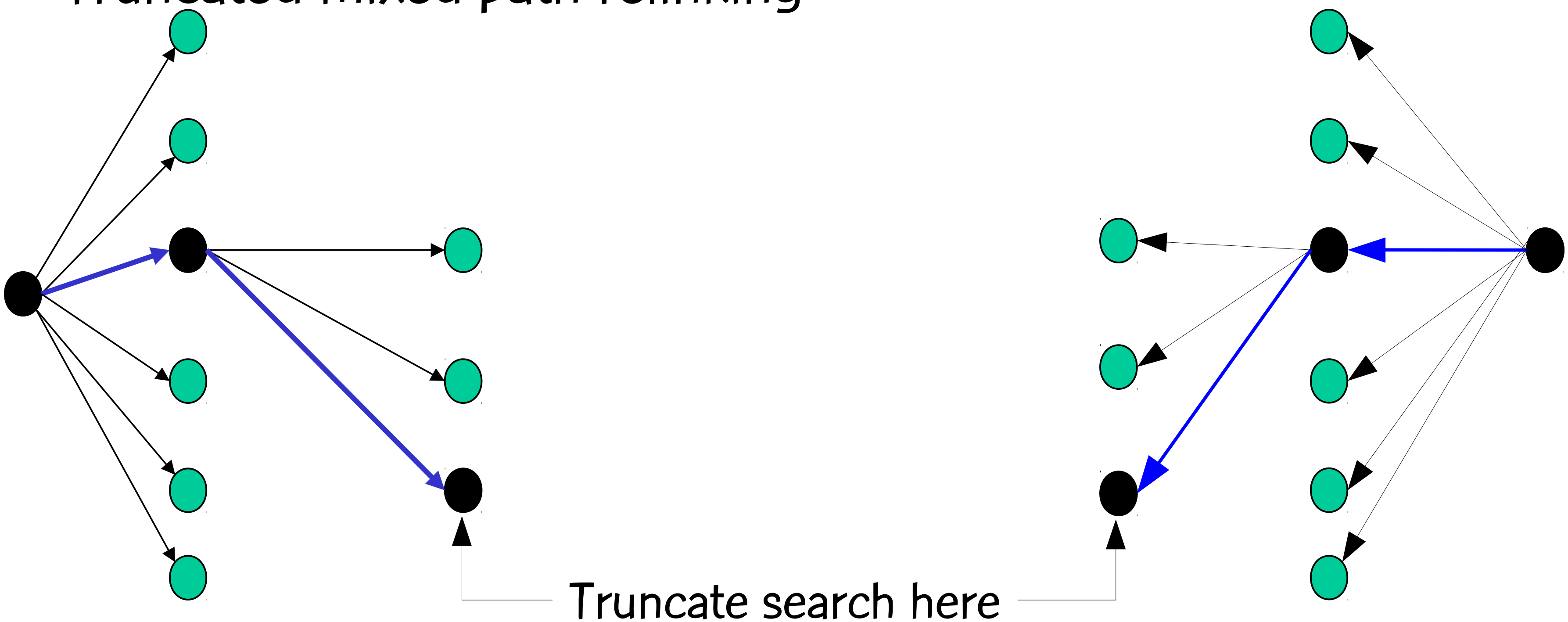


Advantage: explore around neighborhoods of both input solutions.

# Truncated mixed path-relinking

## Variants: trade-offs between computation time and solution quality

Truncated mixed path-relinking

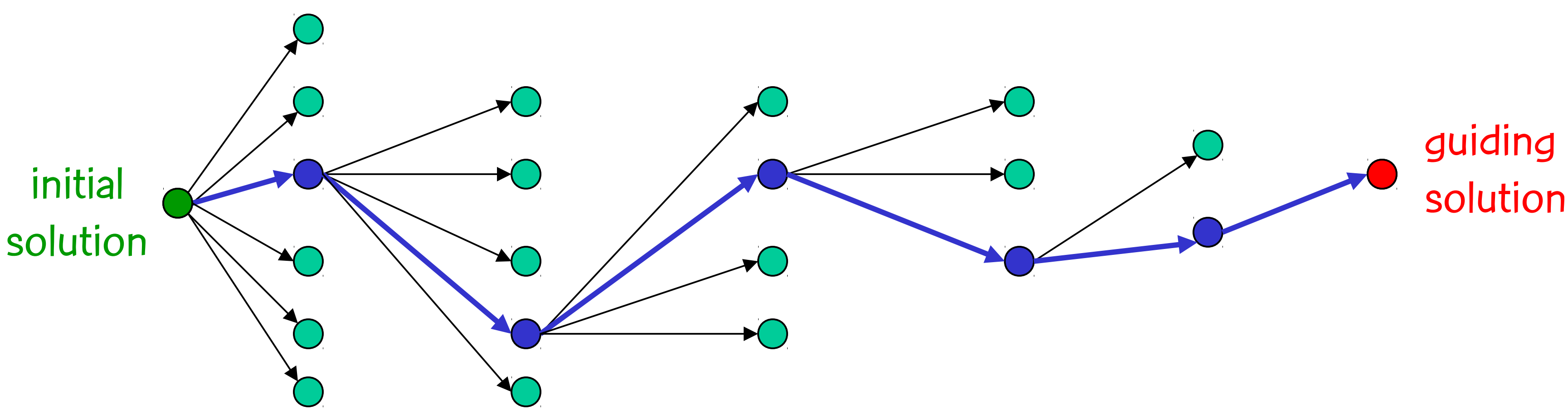


# Greedy randomized adaptive path-relinking

Faria, Binato, Resende, & Falcão (2001, 2005)

Incorporates semi-greediness into PR.

Standard PR selects moves greedily: samples one of exponentially many paths

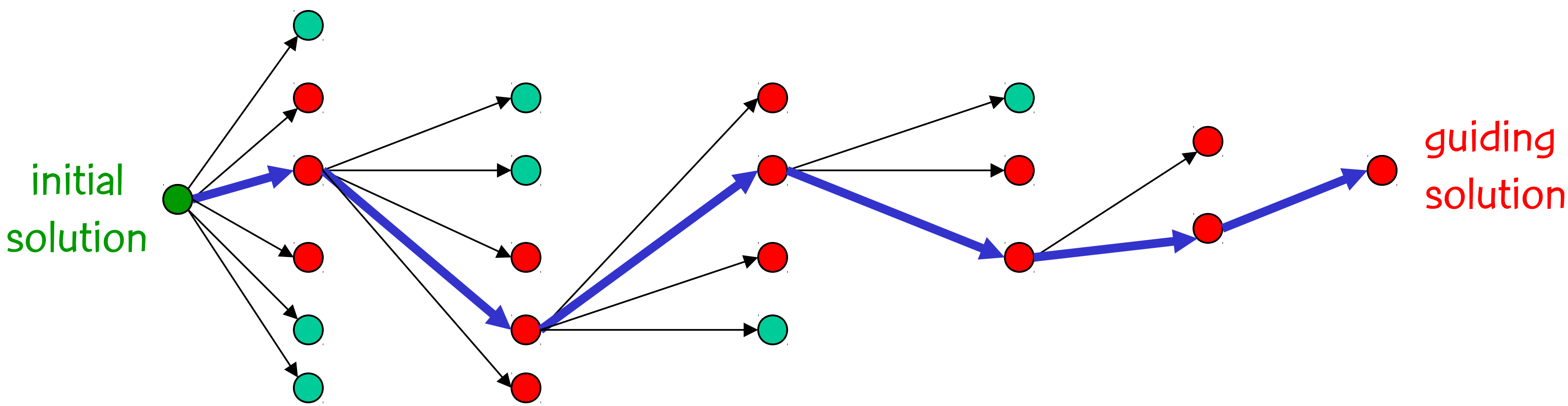


# Greedy randomized adaptive path-relinking

Faria, Binato, Resende, & Falcão (2001, 2005)

Incorporates semi-greediness into PR.

graPR creates RCL with best moves: samples several paths

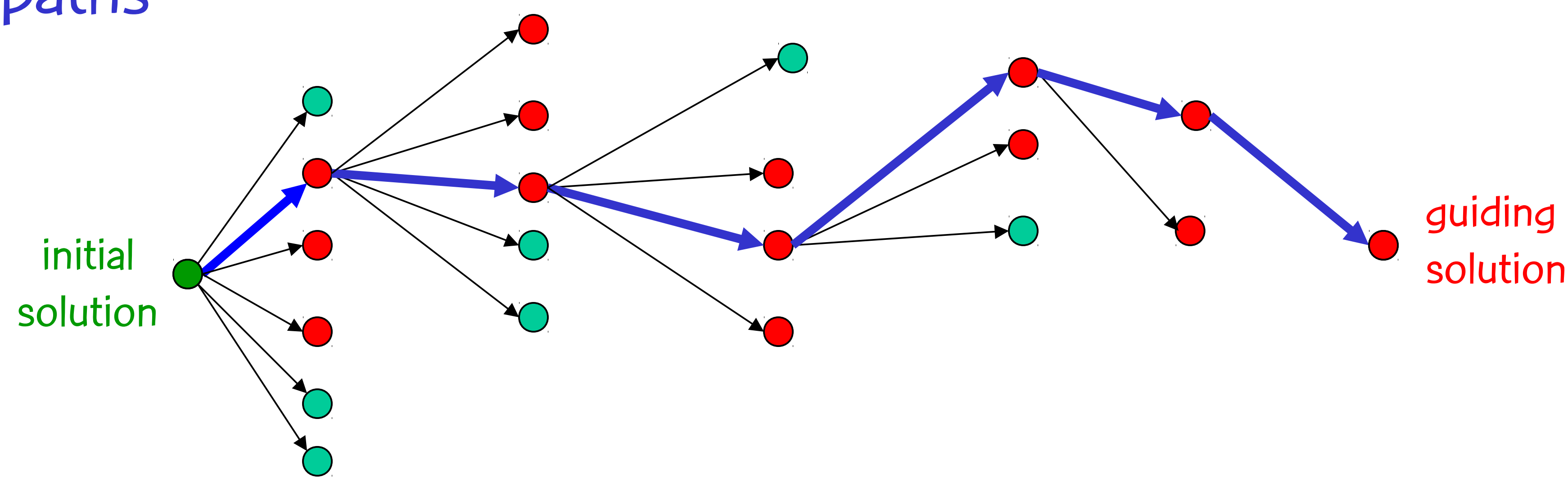


# Greedy randomized adaptive path-relinking

Faria, Binato, Resende, & Falcão (2001, 2005)

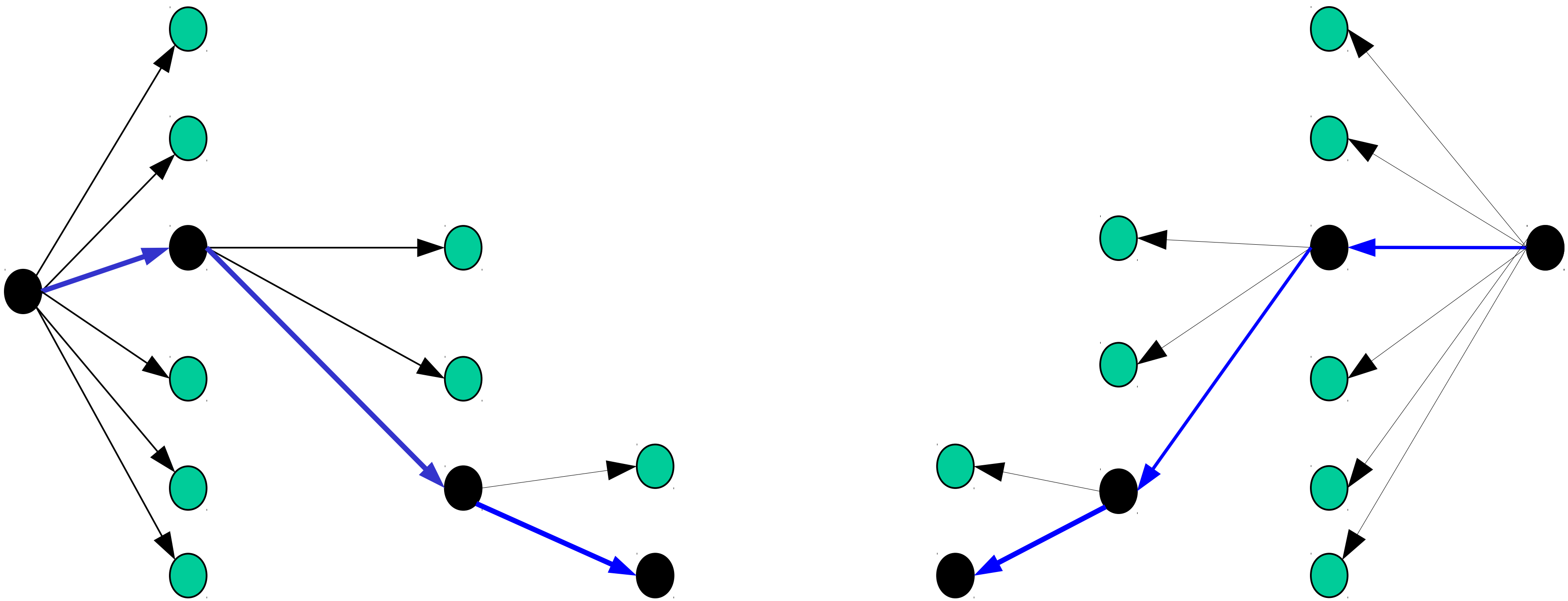
Incorporates semi-greediness into PR.

graPR creates RCL with best moves: samples several paths



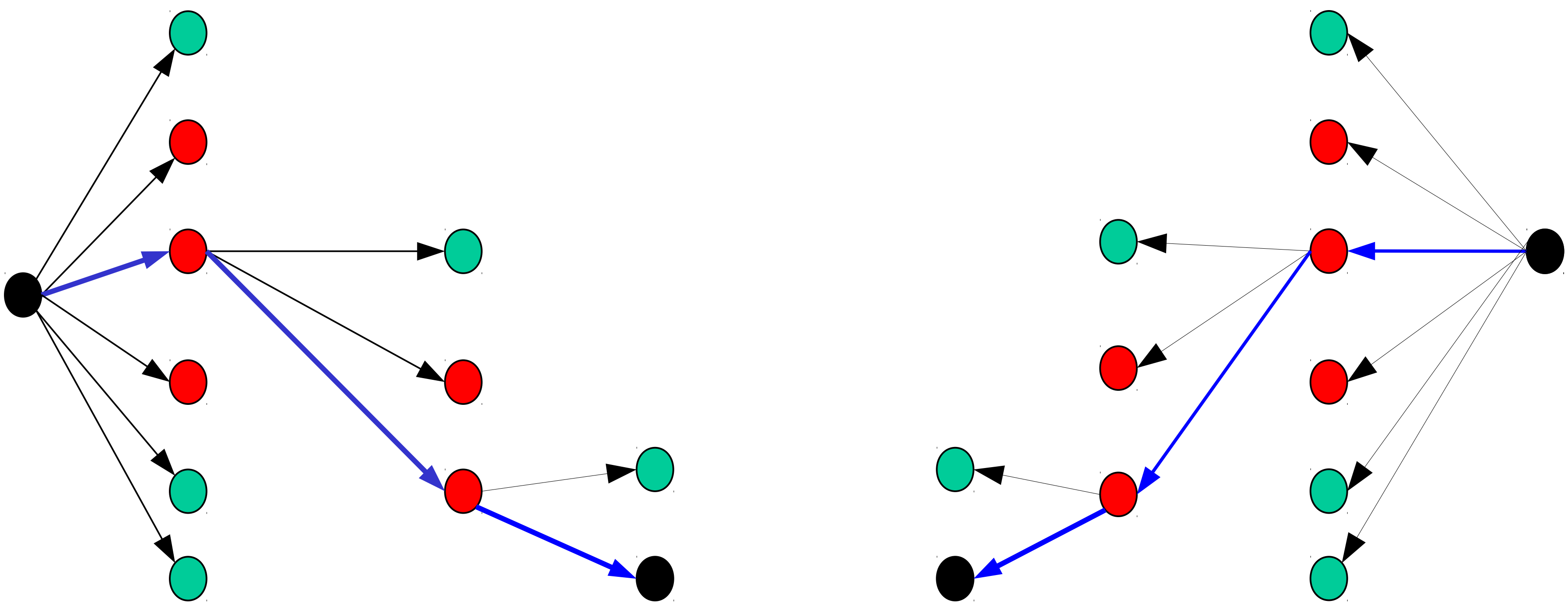
# Truncated mixed graPR

When applied to a given pair of solutions truncated mixed PR explores one of exponentially many path segments each time it is executed.



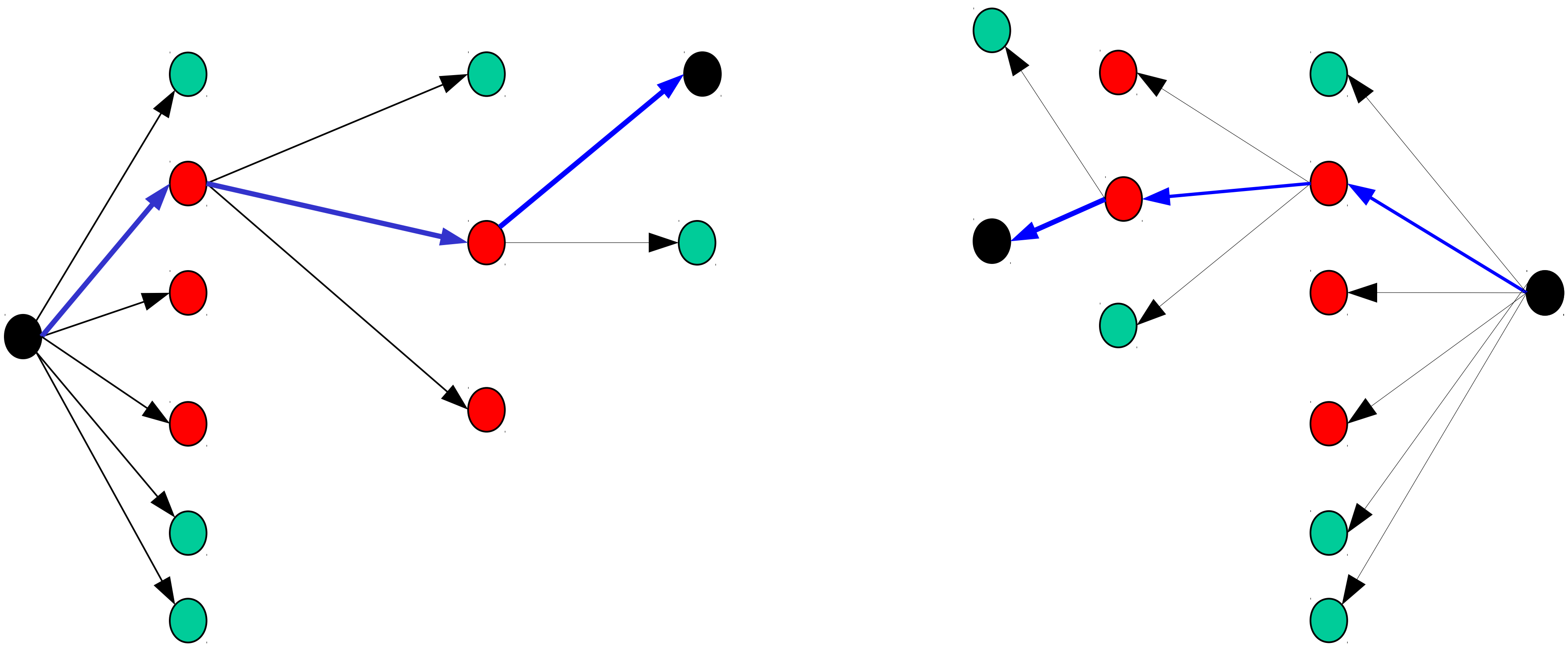
# Truncated mixed graPR

With high probability, truncated mixed graPR explores different path segments each time it is executed between the same pair of solutions.



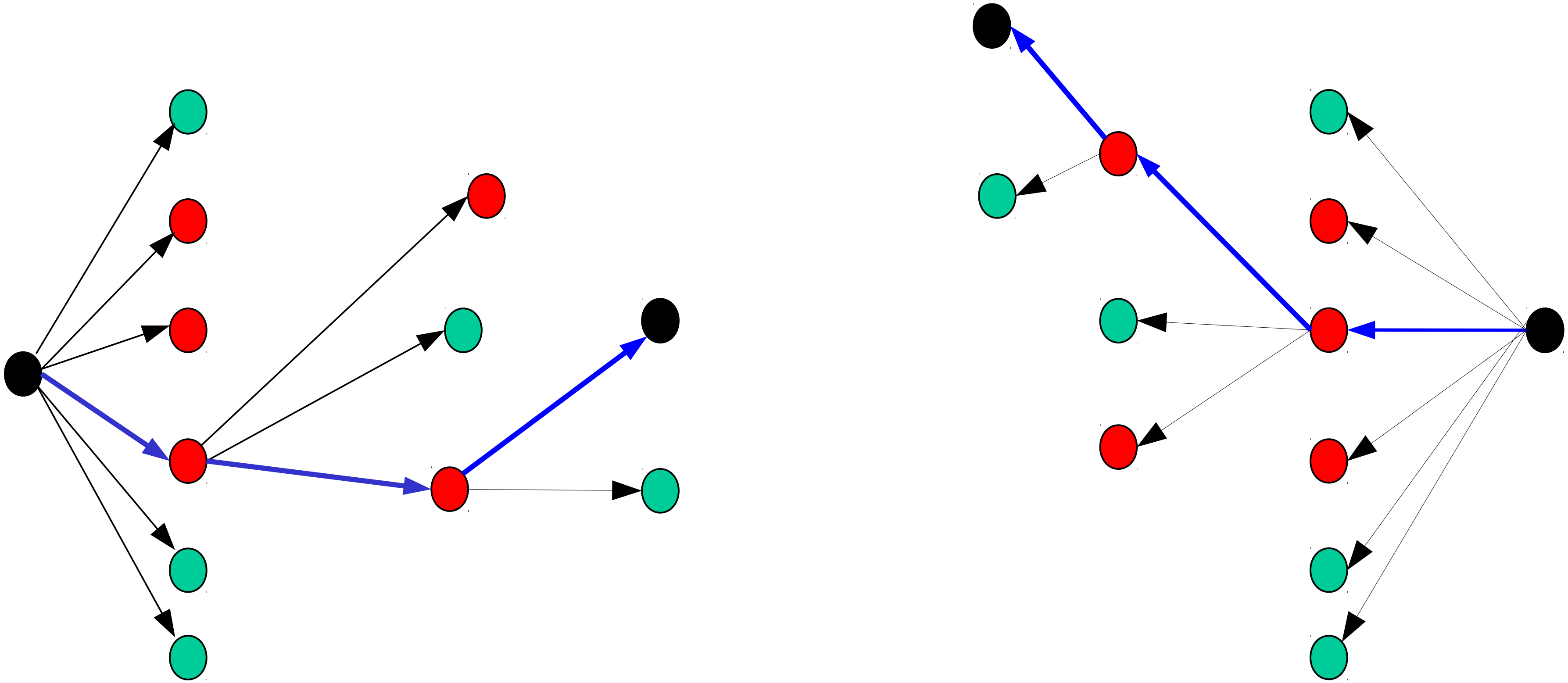
# Truncated mixed graPR

With high probability, truncated mixed graPR explores different path segments each time it is executed between the same pair of solutions.



# Truncated mixed graPR

With high probability, truncated mixed graPR explores different path segments each time it is executed between the same pair of solutions.



# 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.

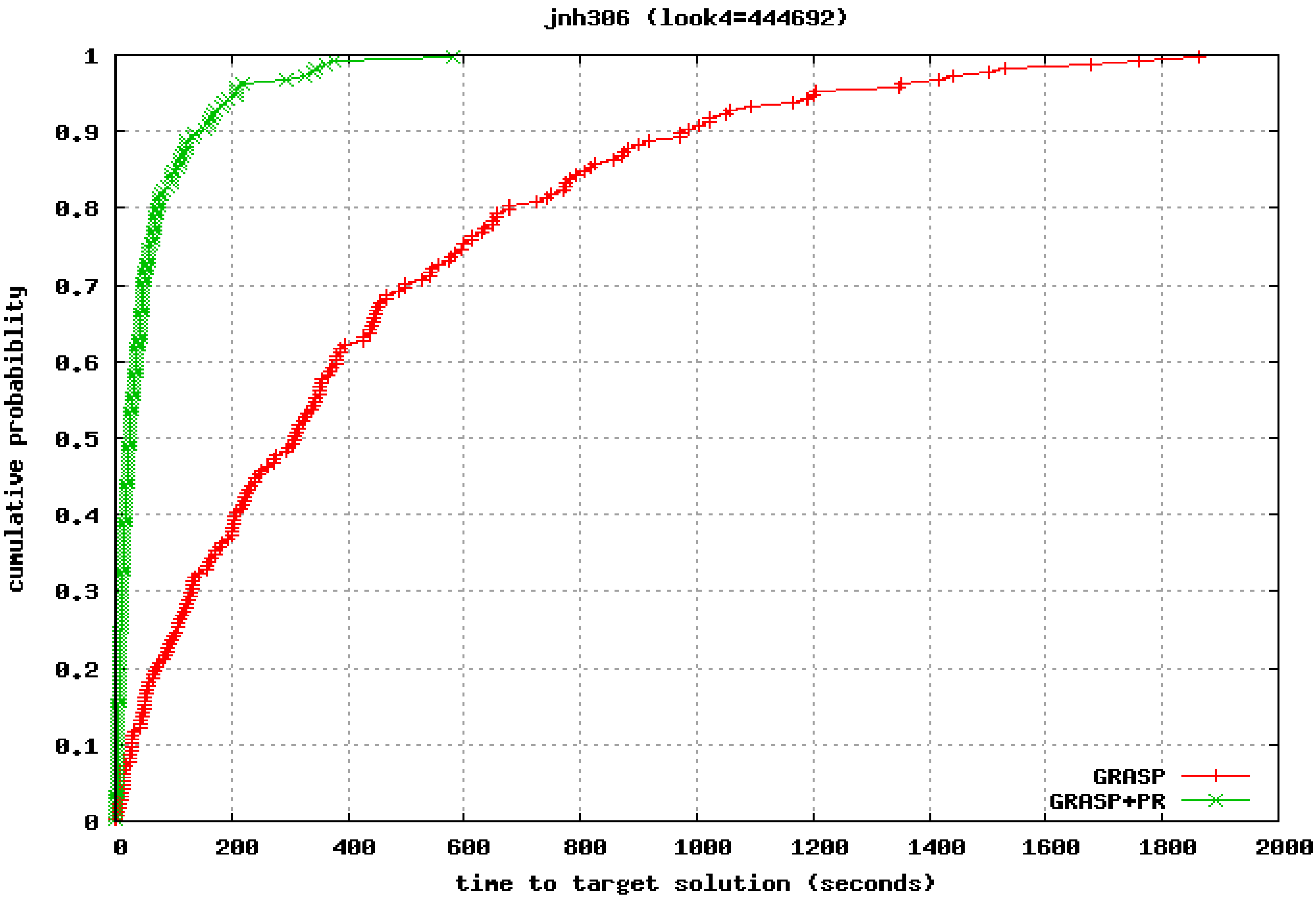
# GRASP with path-relinking

Since 1999, there has been a lot of activity in hybridizing GRASP with path-relinking.

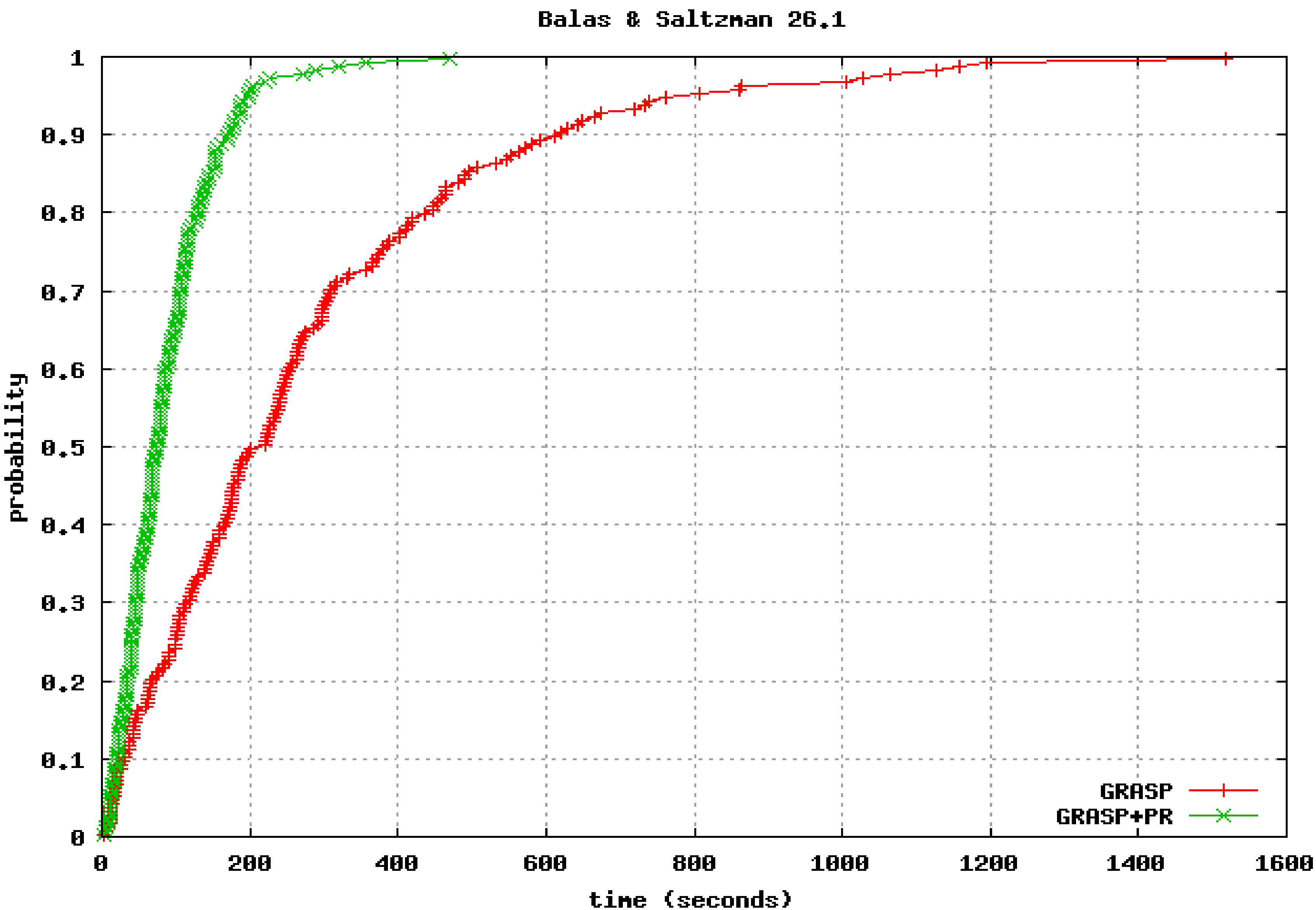
Survey by Resende & Ribeiro in MIC 2003 book of Ibaraki, Nonobe, and Yagiura (2005).

Main observation from experimental studies: GRASP with path-relinking outperforms pure GRASP.

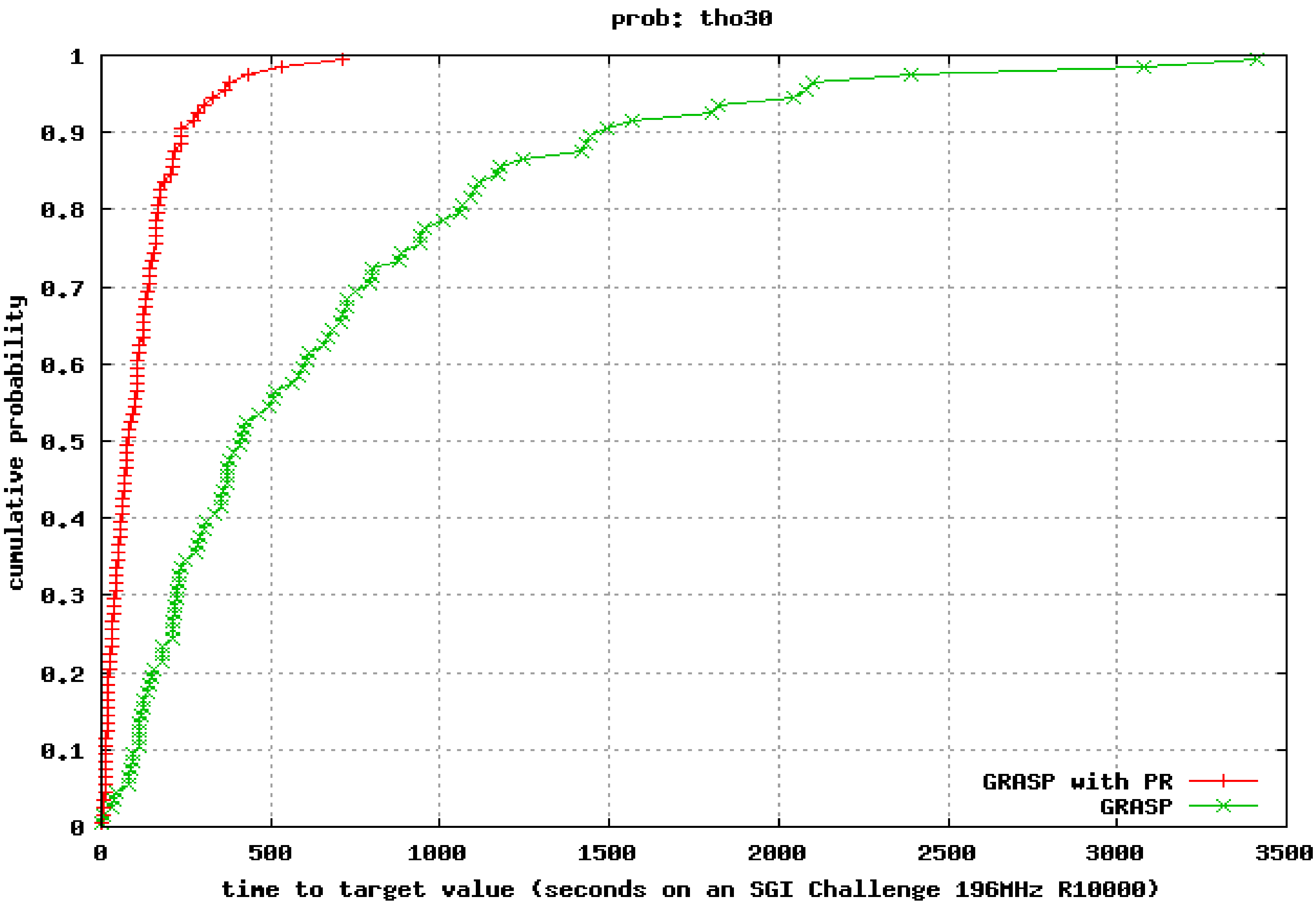
# MAX-SAT (Festa, Pardalos, Pitsoulis, and Resende, 2006)



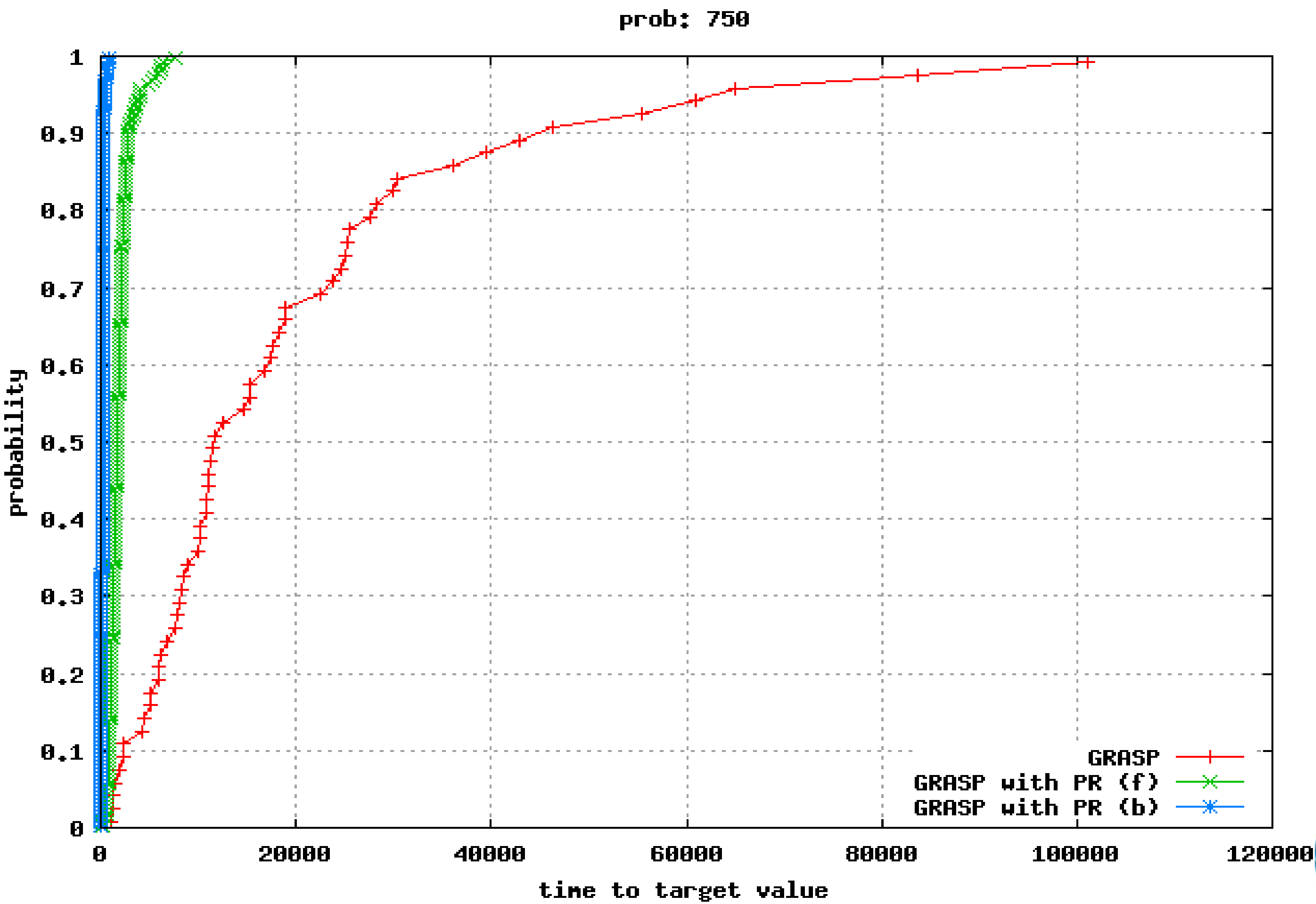
# 3-index assignment (Aiex, Resende, Pardalos, & Toraldo, 2005)



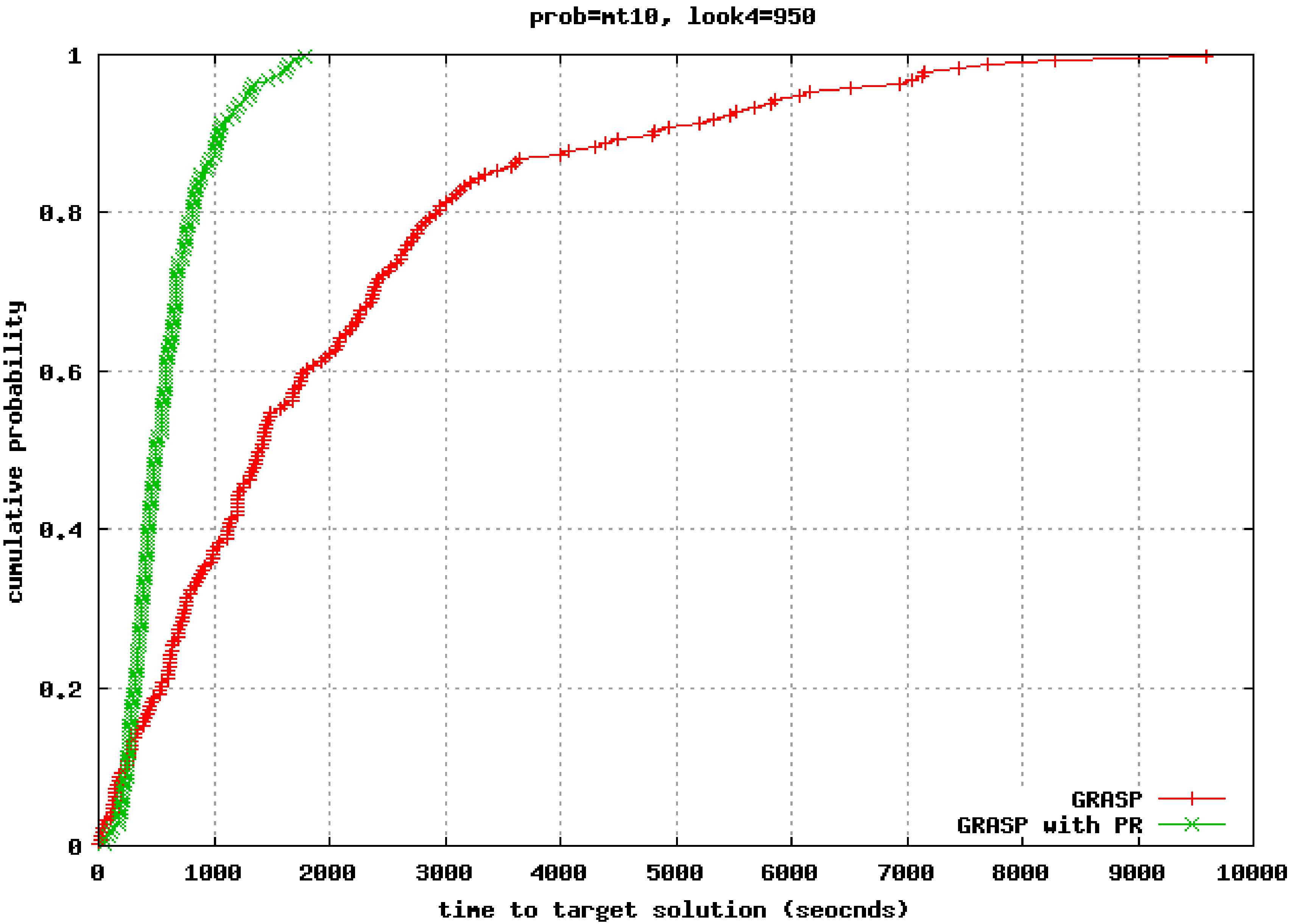
# QAP (Oliveira, Pardalos, and Resende, 2004)



# Bandwidth packing (Resende and Ribeiro, 2003)



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



# GRASP with path-relinking:
## Pool management

P is a set (pool) of elite solutions.

Ideally, pool has a set of good diverse solutions.

Mechanisms are needed to guarantee that pool is made up of those kinds of solutions.

# GRASP with path-relinking:
## Pool management

Each iteration of first |P| GRASP iterations adds one solution to P (if different from others).

After that: solution x is promoted to P if:

x is better than best solution in P.

x is not better than best solution in P, but is better than worst and is sufficiently different from all solutions in P.

# GRASP with path-relinking:
## Pool management

GRASP with PR works best when paths in PR are long, i.e. when the symmetric difference between the initial and guiding solutions is large.

Given a solution to relink with an elite solution, which elite solution to choose?

Choose at random with probability proportional to the symmetric difference.

# GRASP with path-relinking:
## Pool management

Solution quality and diversity are two goals of pool design.

Given a solution X to insert into the pool, which elite solution do we choose to remove?

Of all solutions in the pool with worse solution than X, select to remove the pool solution most similar to X, i.e. with the smallest symmetric difference from X.

# GRASP with path-relinking

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

# Evolutionary path-relinking (EvPR)

# Evolutionary path-relinking

( Resende & Werneck, 2004, 2006 )

Evolutionary path-relinking "evolves" the pool, i.e. transforms it into a pool of diverse elements whose solution values are better than those of the original pool.

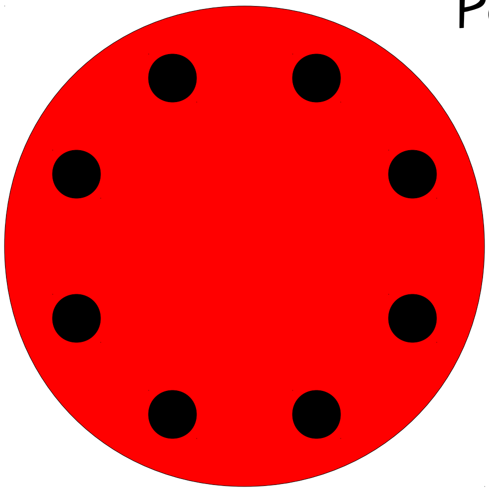
Evolutionary path-relinking can be used

as an intensification procedure at certain points of the solution process;

as a post-optimization procedure at the end of the solution process.

# Evolutionary path-relinking (EvPR)

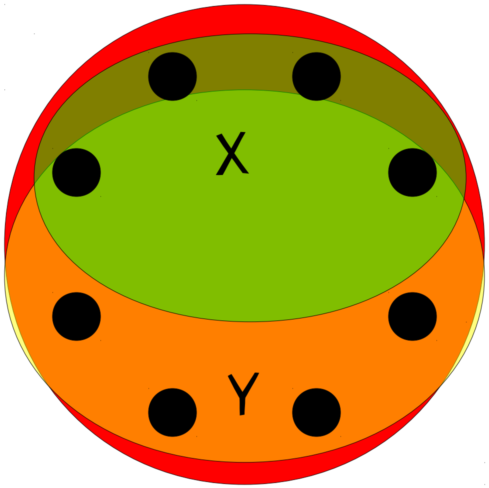
Population P(0)



Each "population" of EvPR starts with a pool of elite solutions of size |P|.

Population P(0) is the current elite set.

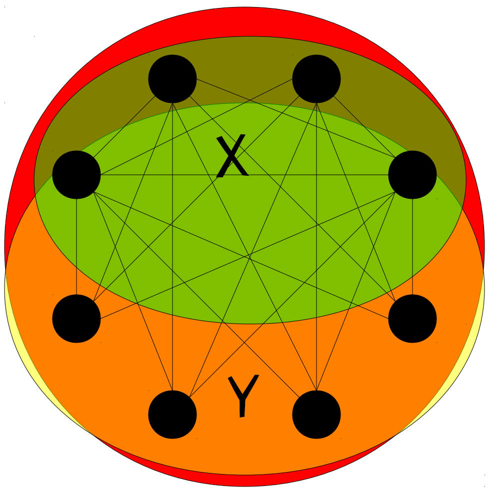
# Evolutionary path-relinking (EvPR)




All pairs of elite solutions $(x,y)$ in K-th population $P(K)$, such that $x \in X \subseteq P(K)$ and $y \in Y \subseteq P(K)$, are path-relinked and the resulting $z = PR(x,y)$ is a candidate for inclusion in population $P(K+1)$.

Rules for inclusion into $P(K+1)$ are the same used for inclusion into any pool.

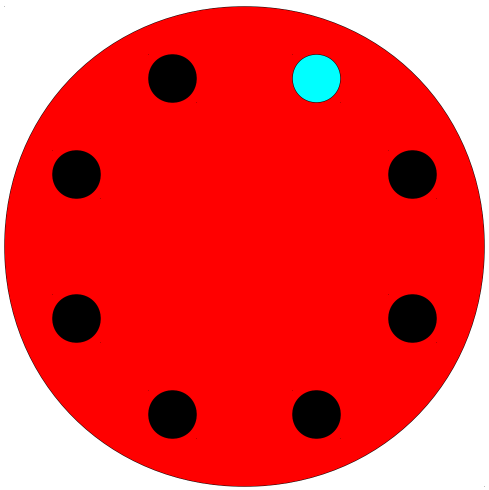
# Evolutionary path-relinking (EvPR)




All pairs of elite solutions (x,y) in K-th population P(K), such that $x \in X \subseteq P(K)$ and $y \in Y \subseteq P(K)$, are path-relinked and the resulting $z = PR(x,y)$ is a candidate for inclusion in population P(K+1).

Rules for inclusion into P(K+1) are the same used for inclusion into any pool.

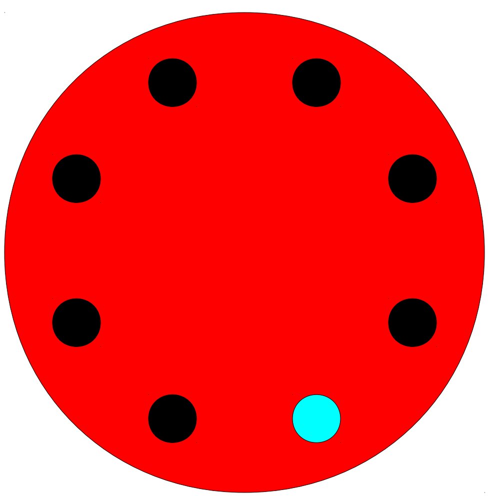
# Evolutionary path-relinking (EvPR)



Population P(K)

If best solution in population P(K+1) has same objective function value as best solution in population P(K), process stops.

Else K=K+1 and repeat.



Population P(K+1)

# GRASP with evolutionary path-relinking

## As post-optimization

Repeat GRASP with PR loop

1) Construct greedy randomized
2) Local search
3) Path-relinking
4) Update pool

Evolutionary-PR

## During GRASP + PR

Repeat outer loop

Repeat inner loop

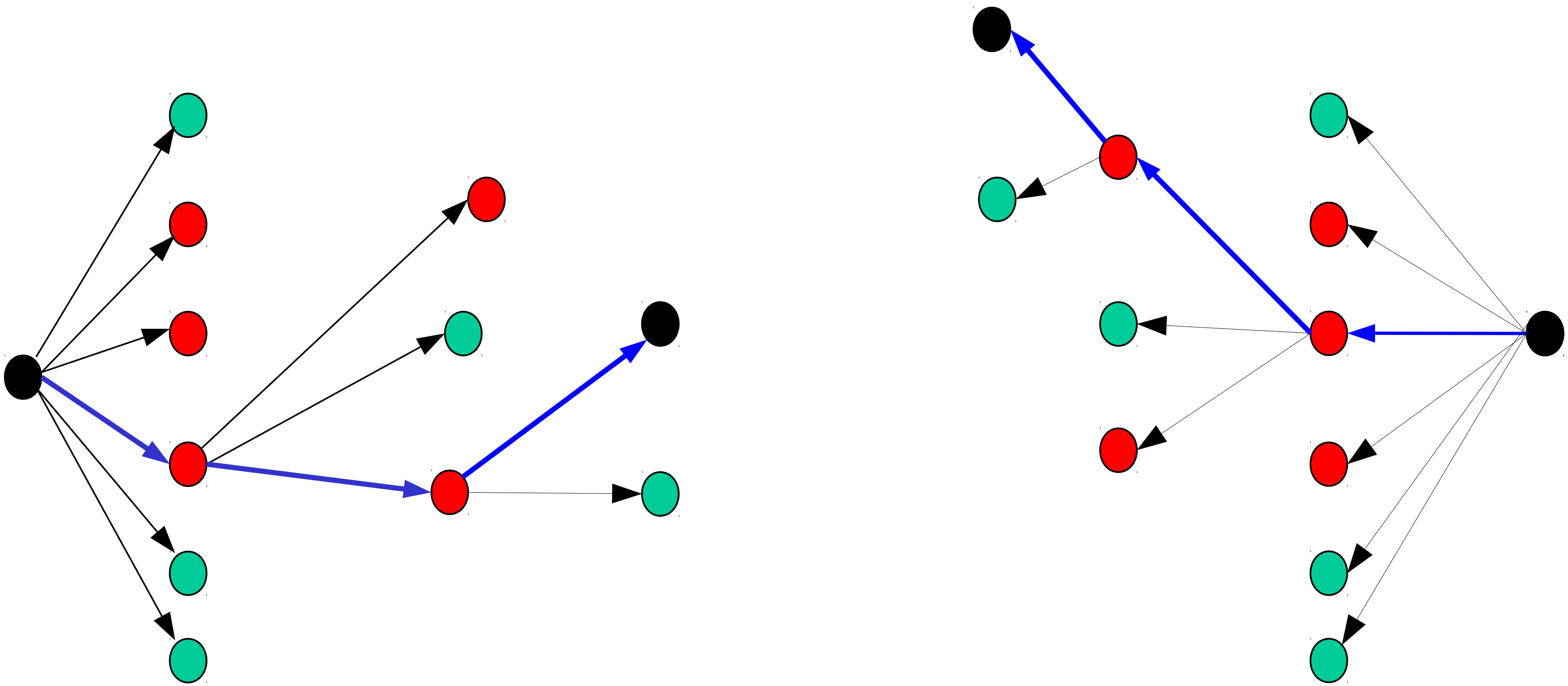
1) Construct greedy randomized
2) Local search
3) Path-relinking
4) Update pool

Evolutionary-PR

( Resende & Werneck, 2004, 2006 )

# GRASP with EvPR: Implementation ideas

## Truncated mixed graPR



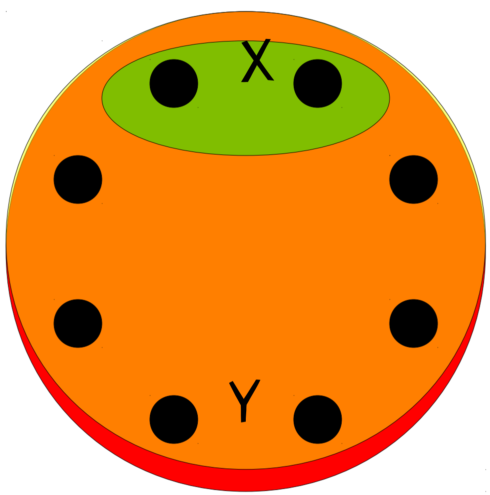
In PR and EvPR, apply one iteration of graPR.

For (x,y), different calls to graPR(x,y) explore different paths.

# GRASP with EvPR: Implementation ideas

Make set X small and with best pool solutions.

Make set Y be entire pool.




Use set X of size 1 or 2.

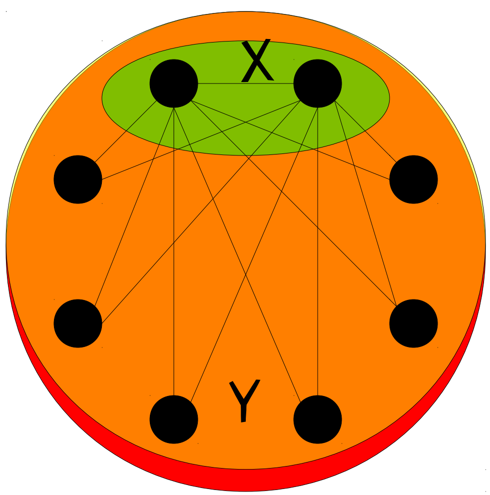
Speeds up EvPR.

Avoids unfruitful calls to graPR(x,y)

# GRASP with EvPR: Implementation ideas

Make set X small and with best pool solutions.

Make set Y be entire pool.



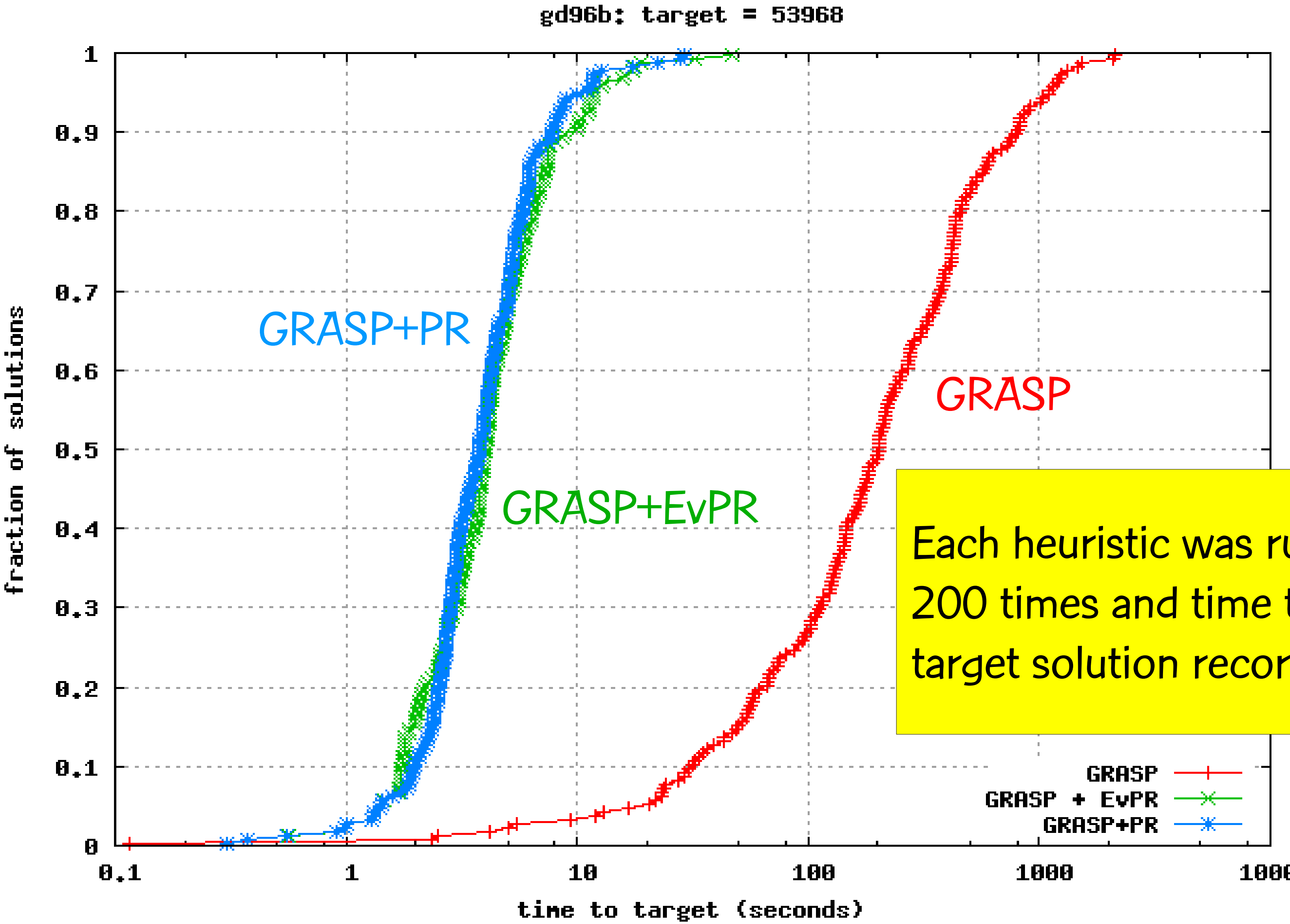

Use set X of size 1 or 2.

Speeds up EvPR.

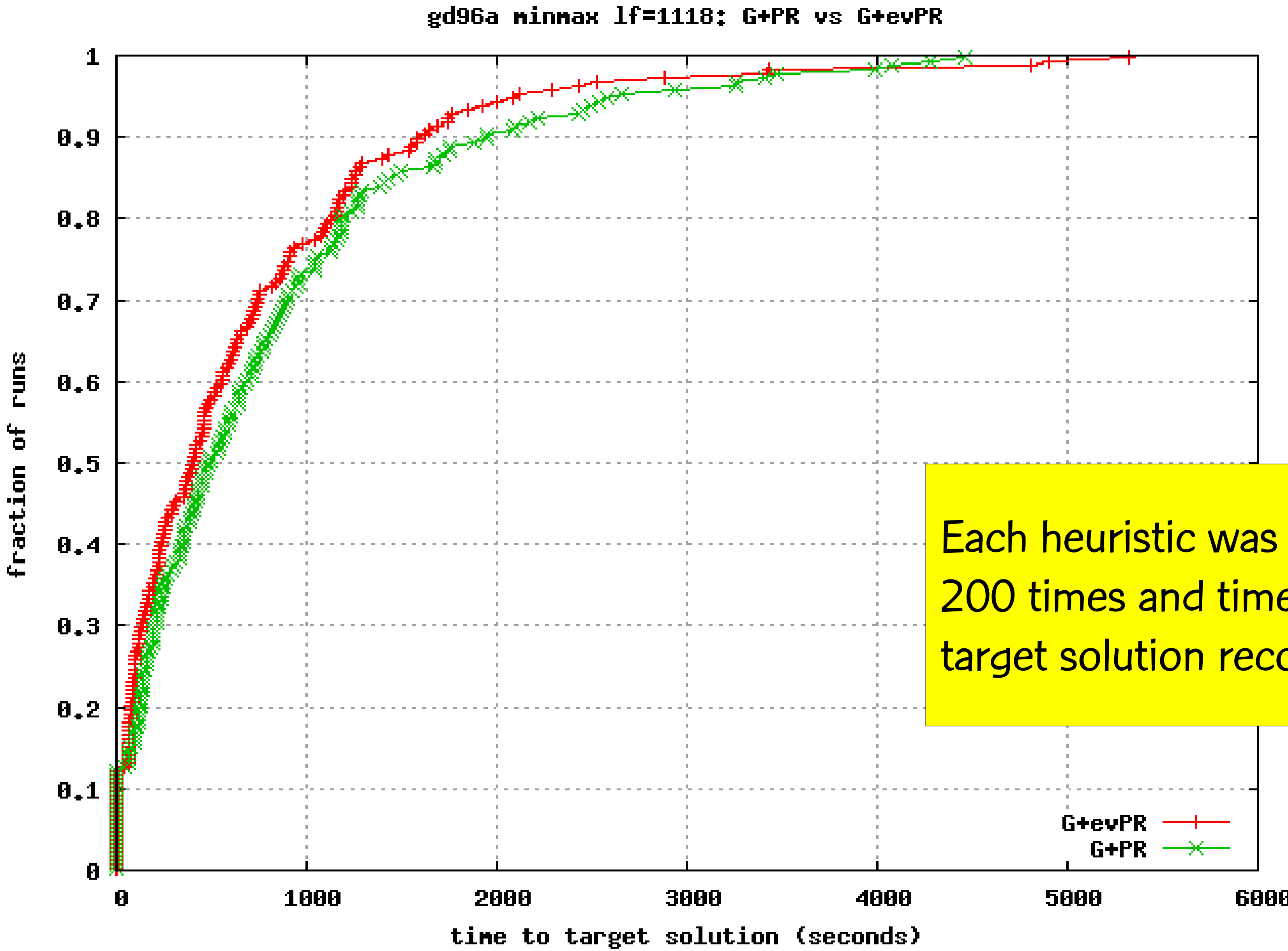
Avoids unfruitful calls to graPR(x,y)

Weights uniformly distributed in interval [1,100]: min sum cuts

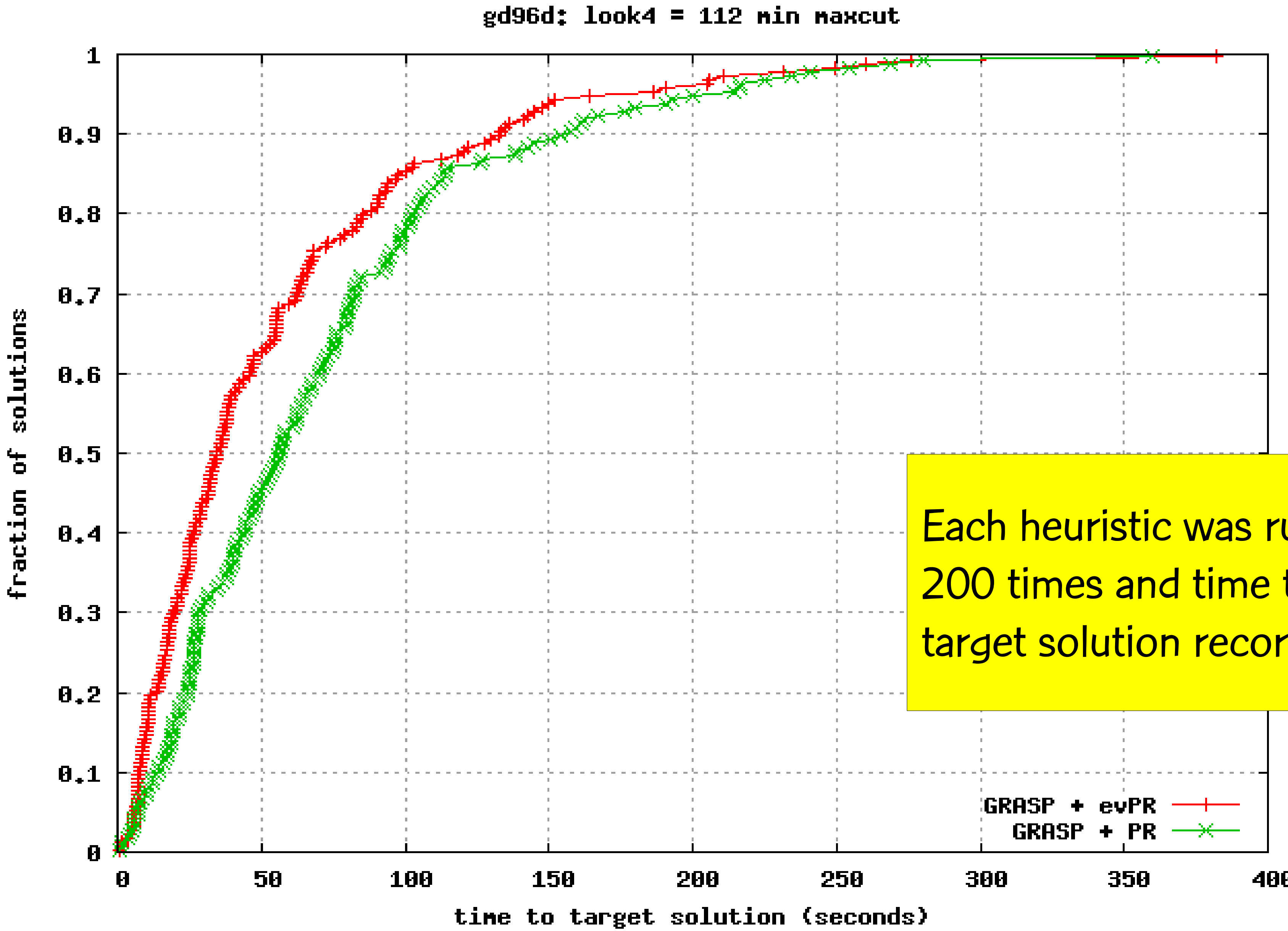





Each heuristic was run 200 times and time to target solution recorded.

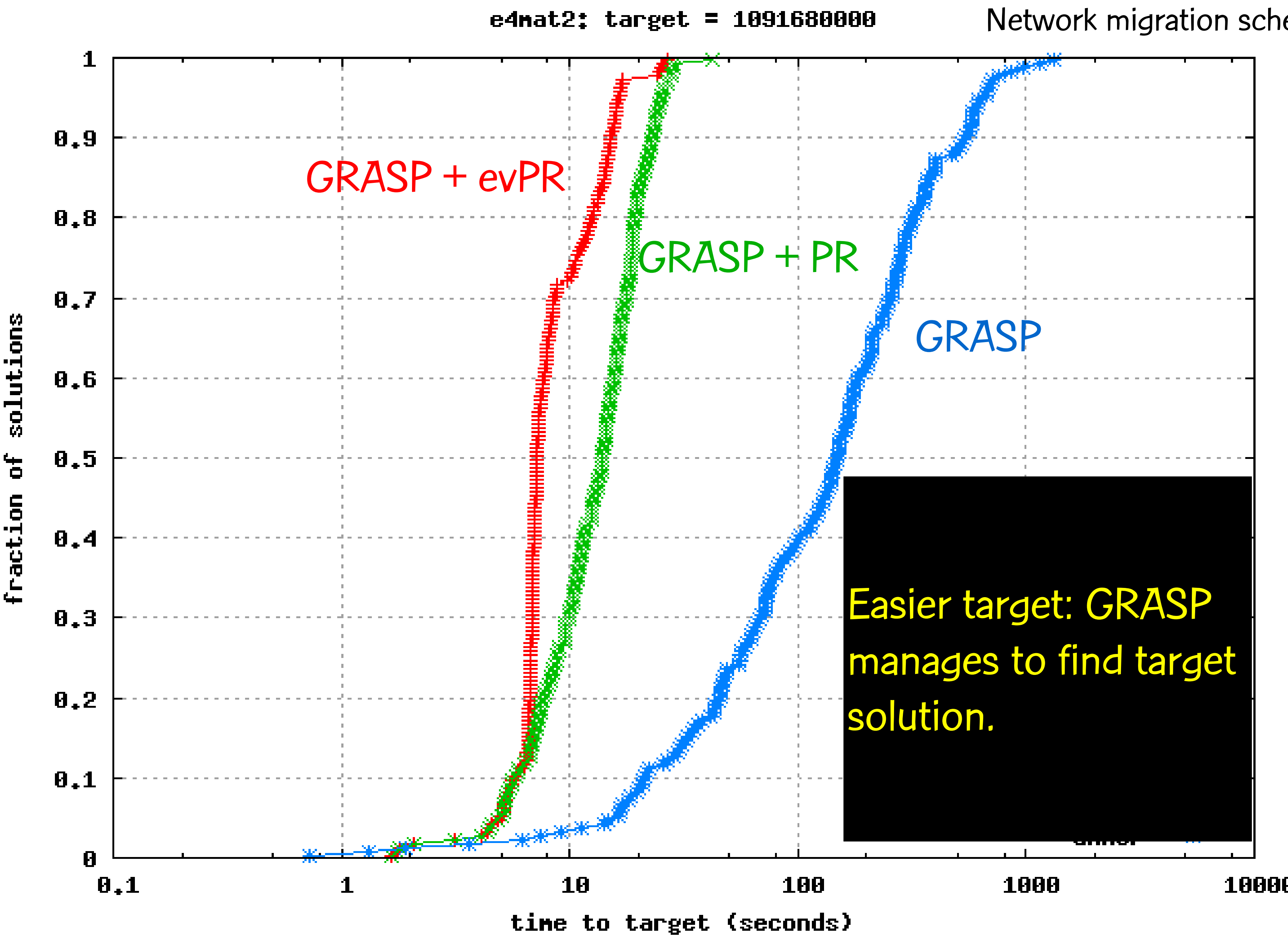



Each heuristic was run 200 times and time to target solution recorded.

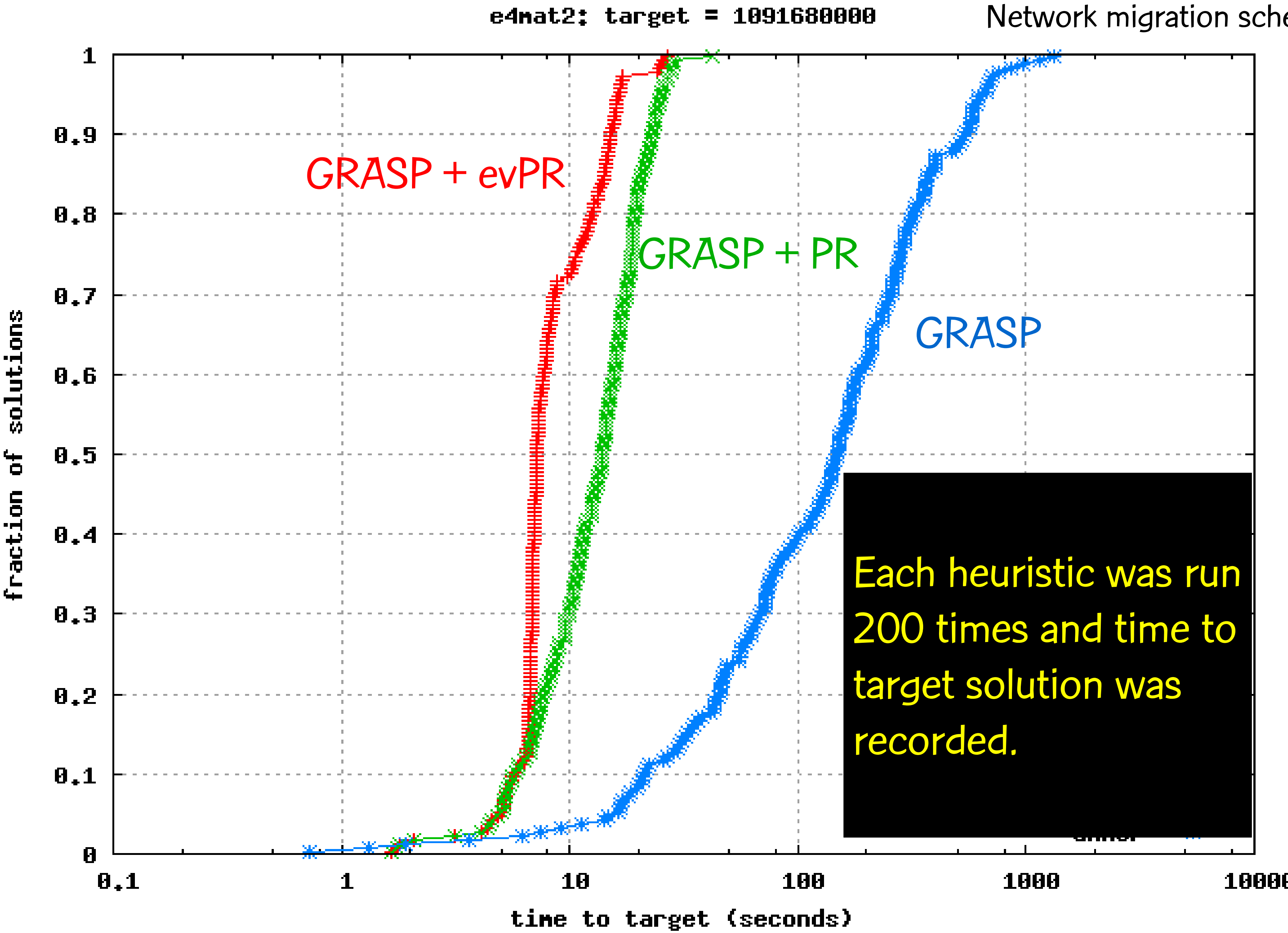


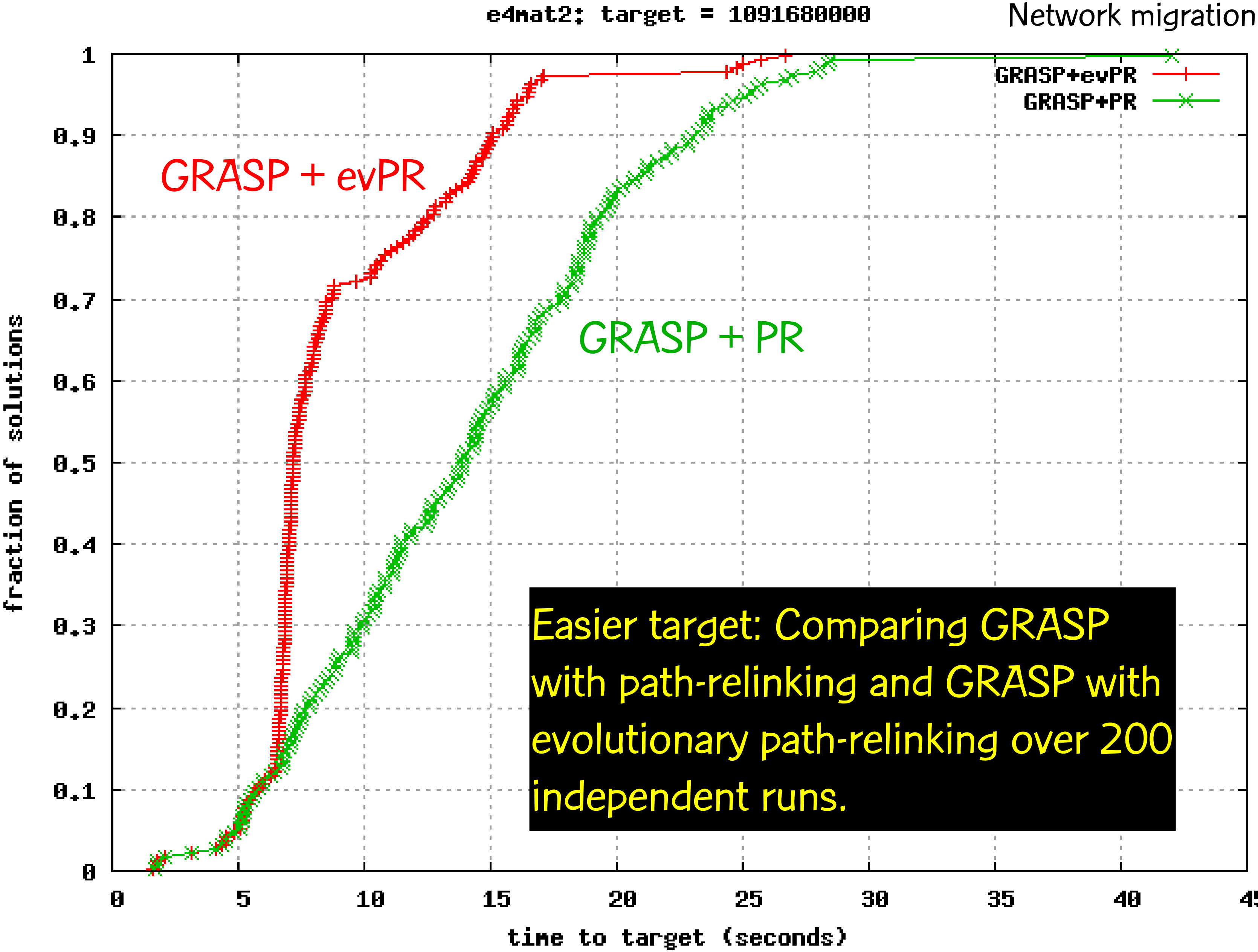

Each heuristic was run 200 times and time to target solution recorded.

## Network migration scheduling

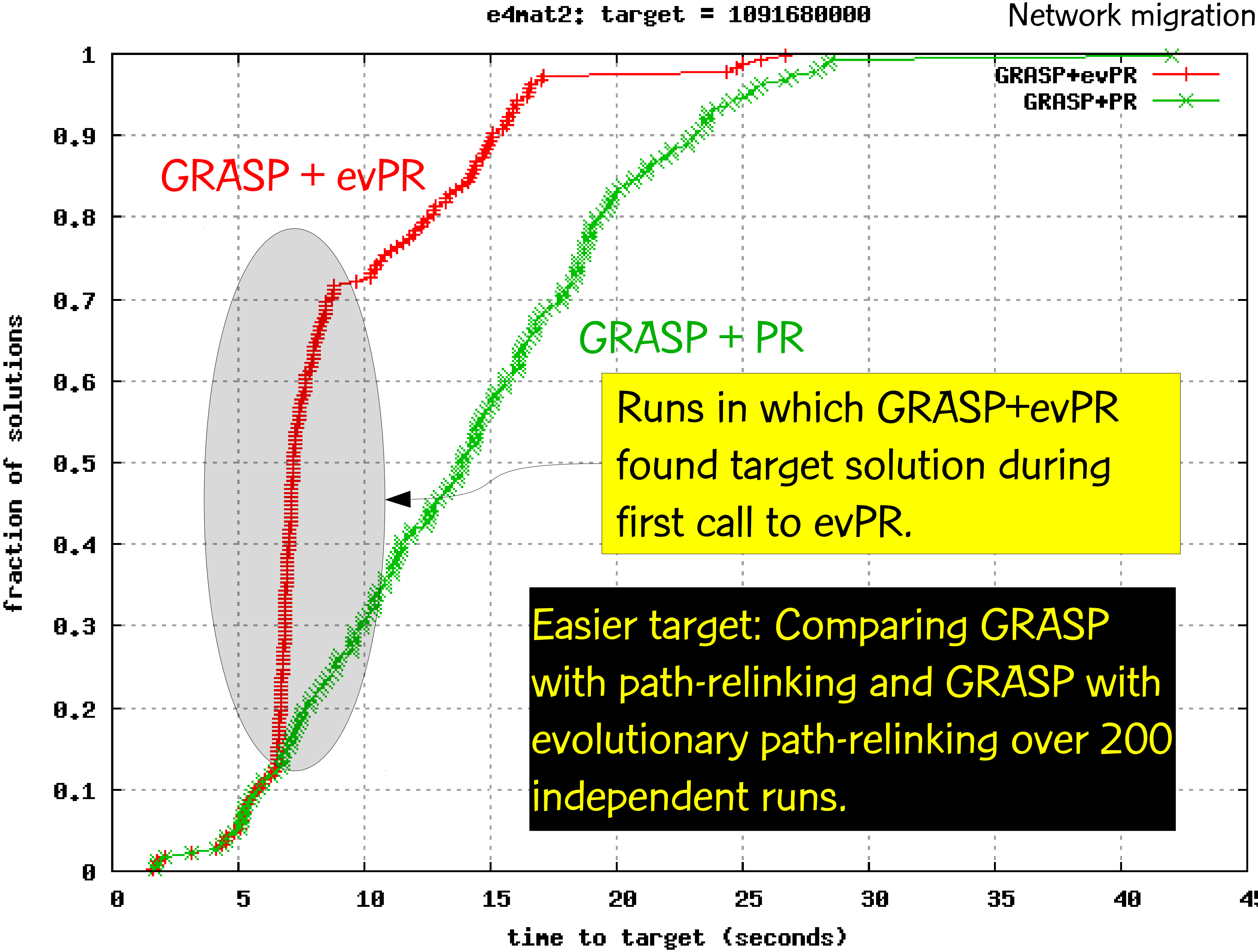


## Network migration scheduling

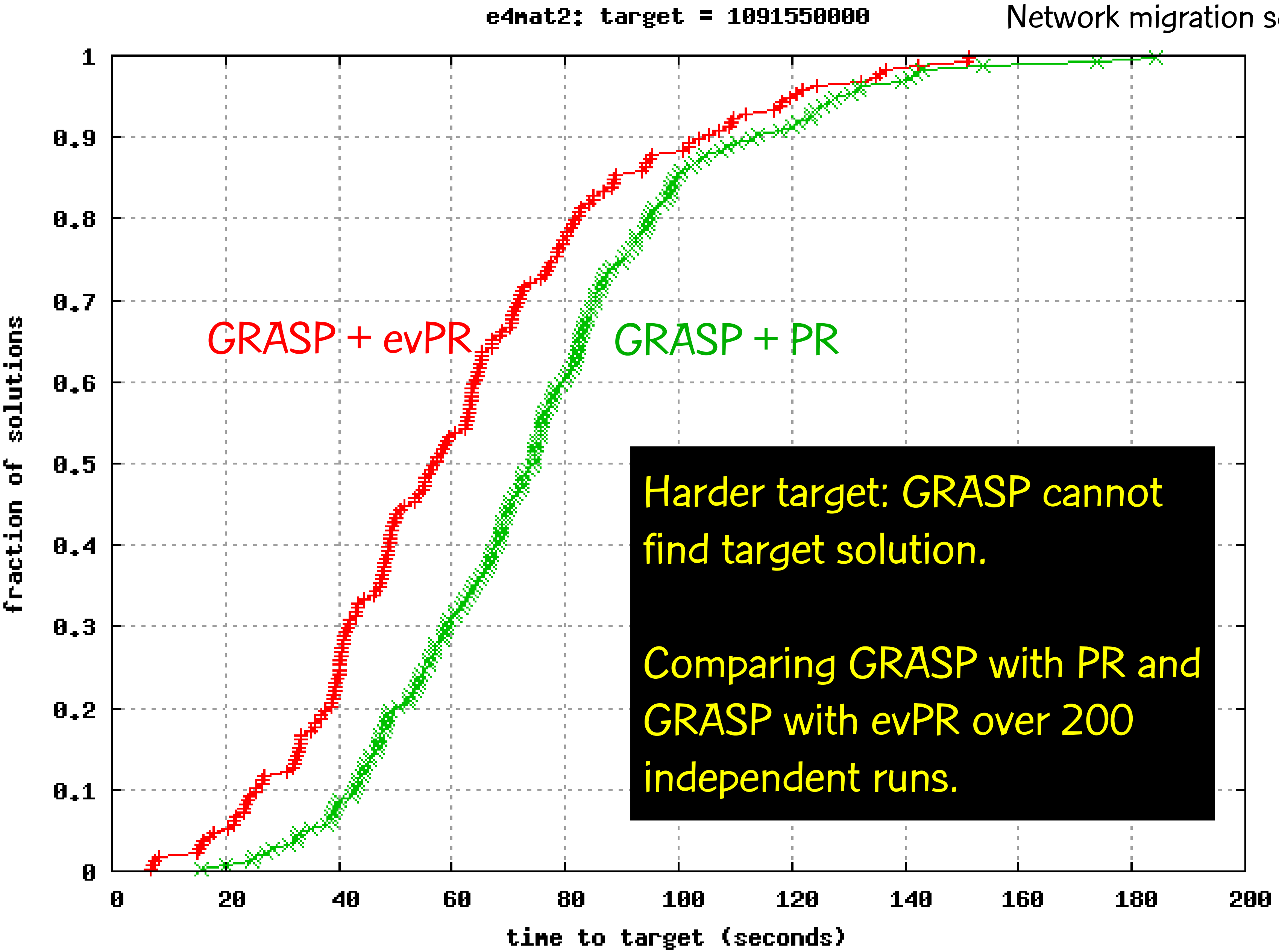





Easier target: Comparing GRASP with path-relinking and GRASP with evolutionary path-relinking over 200 independent runs.


e4mat2: target = 1091680000
GRASP+evPR
GRASP+PR
GRASP + evPR
GRASP + PR
fraction of solutions
time to target (seconds)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
30
35
40
45
Runs in which GRASP+evPR found target solution during first call to evPR.
Easier target: Comparing GRASP with path-relinking and GRASP with evolutionary path-relinking over 200 independent runs.

# Network migration scheduling




Harder target: GRASP cannot find target solution.

Comparing GRASP with PR and GRASP with evPR over 200 independent runs.

# Examples of PR within GRASP

Laguna and Martí (1999): 2-layer straight line crossing minimization

Canuto et al. (2001): Prize-collecting Steiner problem in graphs

Resende and Ribeiro (2001): Bandwidth packing

Ribeiro et al. (2002): Steiner problem in graphs

Resende and Werneck (2004,2006): p-median problem & capacitated facility location

Aiex et al. (2005): Three-index assignment

Resende and Ribeiro (2005): Survey paper on GRASP & PR

Mateus, Resende, and Silva (2010): generalized QAP

# Continuous GRASP (C-GRASP)

# C-GRASP

- C-GRASP is a metaheuristic to finding optimal or near-optimal solutions to
  - Min f(x) subject to: $L \leqslant x \leqslant U$
  - where $x, L, U \in R^n$
  - and f(x) is continuous but can have discontinuities, be non-differentiable, be the output of a simulation, etc.

# C-GRASP

- C-GRASP is based on the discrete optimization metaheuristic GRASP
- It was proposed in 2006 by U. of FL ISE PhD students Michael Hirsch and Claudio Meneses with Mauricio Resende and Panos Pardalos.
- M.J. Hirsch, C.N. Meneses, P.M. Pardalos, and M.G.C. Resende, "Global optimization by continuous GRASP," Optimization Letters, vol. 1, pp. 201-212, 2007.
- M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Speeding up continuous GRASP," European J. of Operational Research, vol. 205, pp. 507-521, 2010.

# C-GRASP

- C-GRASP is a multi-start procedure, i.e. a major loop is repeated until some stopping criterion is satisfied.
- In each major iteration
  – x is initialized with a solution randomly selected from the box defined by vectors L and U.
  – a number of minor iterations are carried out, where each minor iterations consists of a construction phase and a local improvement phase.
  – Minor iterations are done on a dynamic grid and stops when the grid is too dense.

# C-GRASP

$f^* = \infty$

**while** (stopping criterion not satisfied) **do**

  $x$ = random[L,U]; $h = h(\text{start})$;

  **while** ( $h \geq h(\text{end})$ ) **do**

    $x$ = ConstructGreedyRandomized($x$)

    $x$ = LocalImprovement($x$)

    **if** ( $f(x) < f^*$ ) **then** { $x^* = x$; $f^* = f(x)$ }

    **if** ( $x$ did not improve this iteration ) **then** { $h = h/2$ }

  **end while**

**end while**

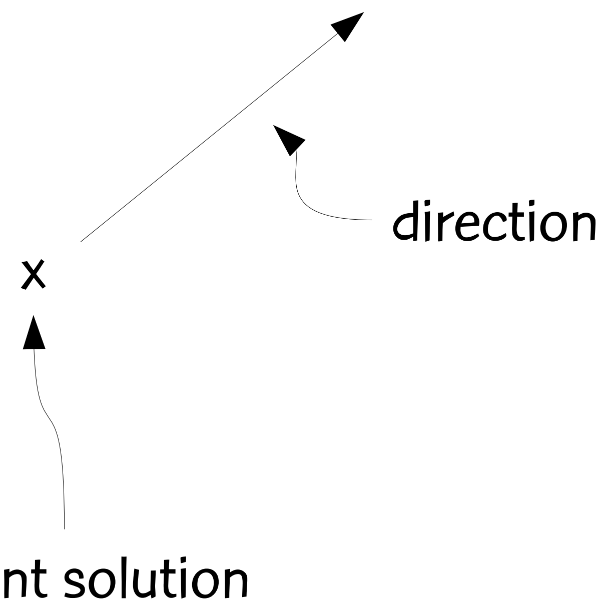
**return** $x^*$

# C-GRASP line search

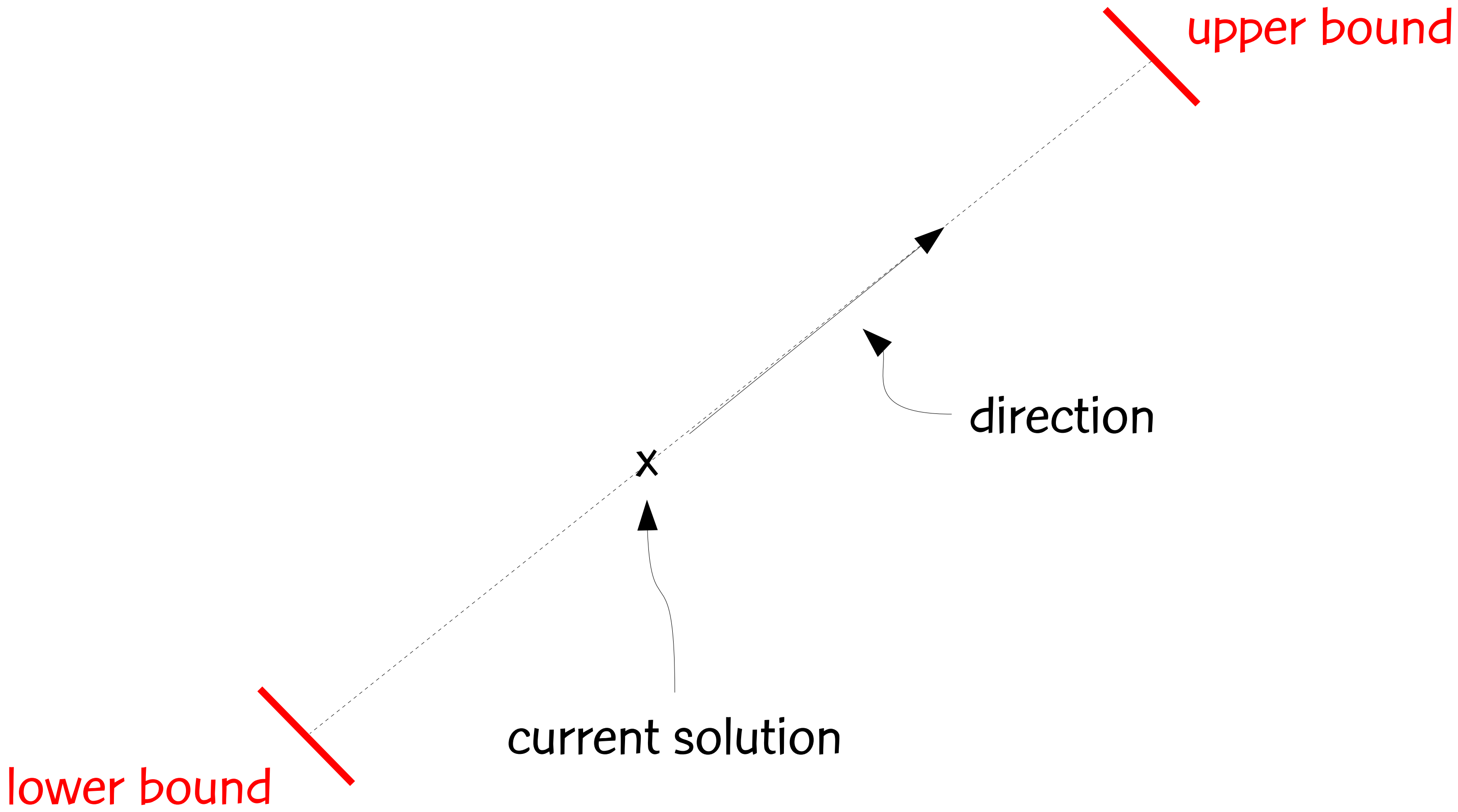
x

current solution

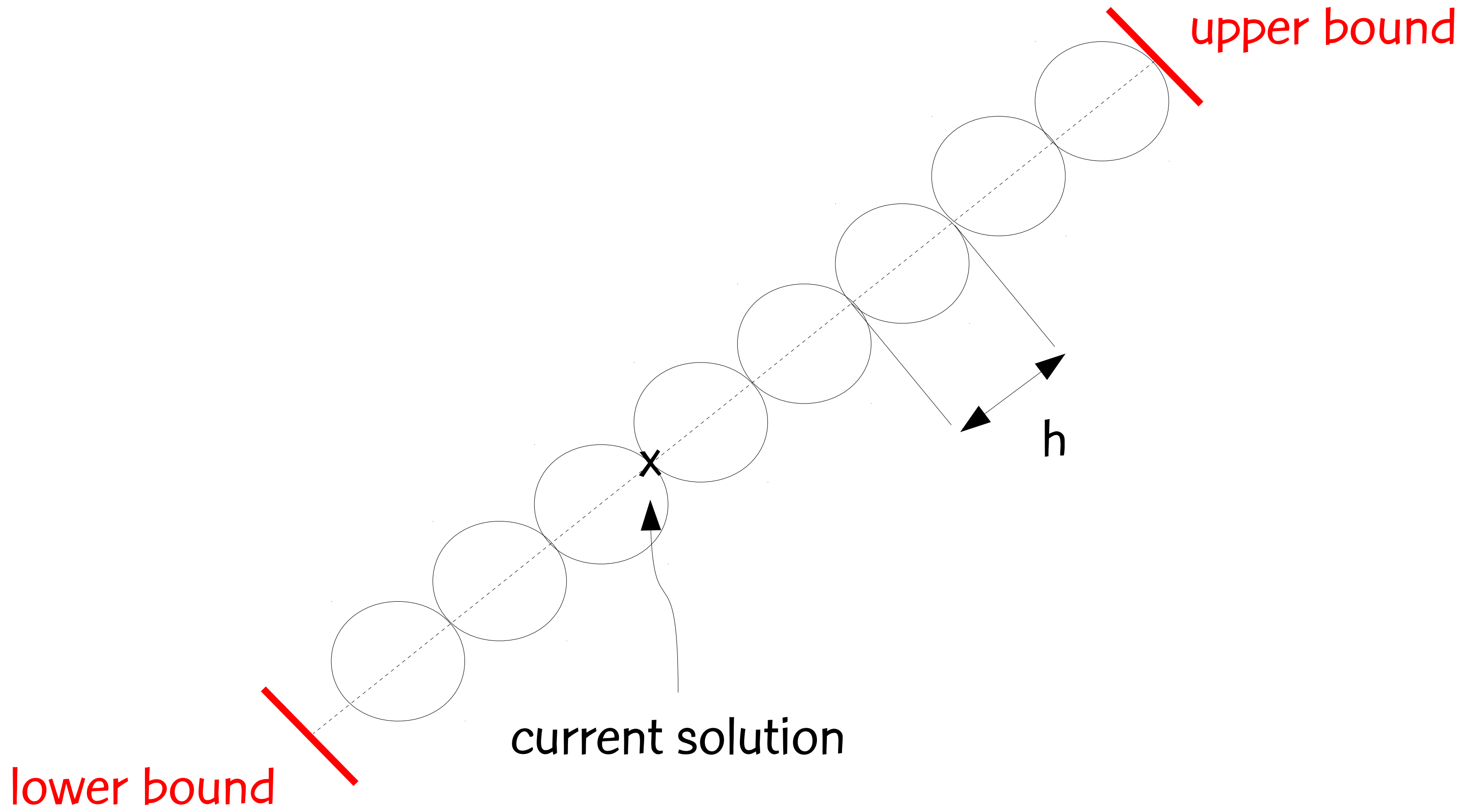
# C-GRASP line search



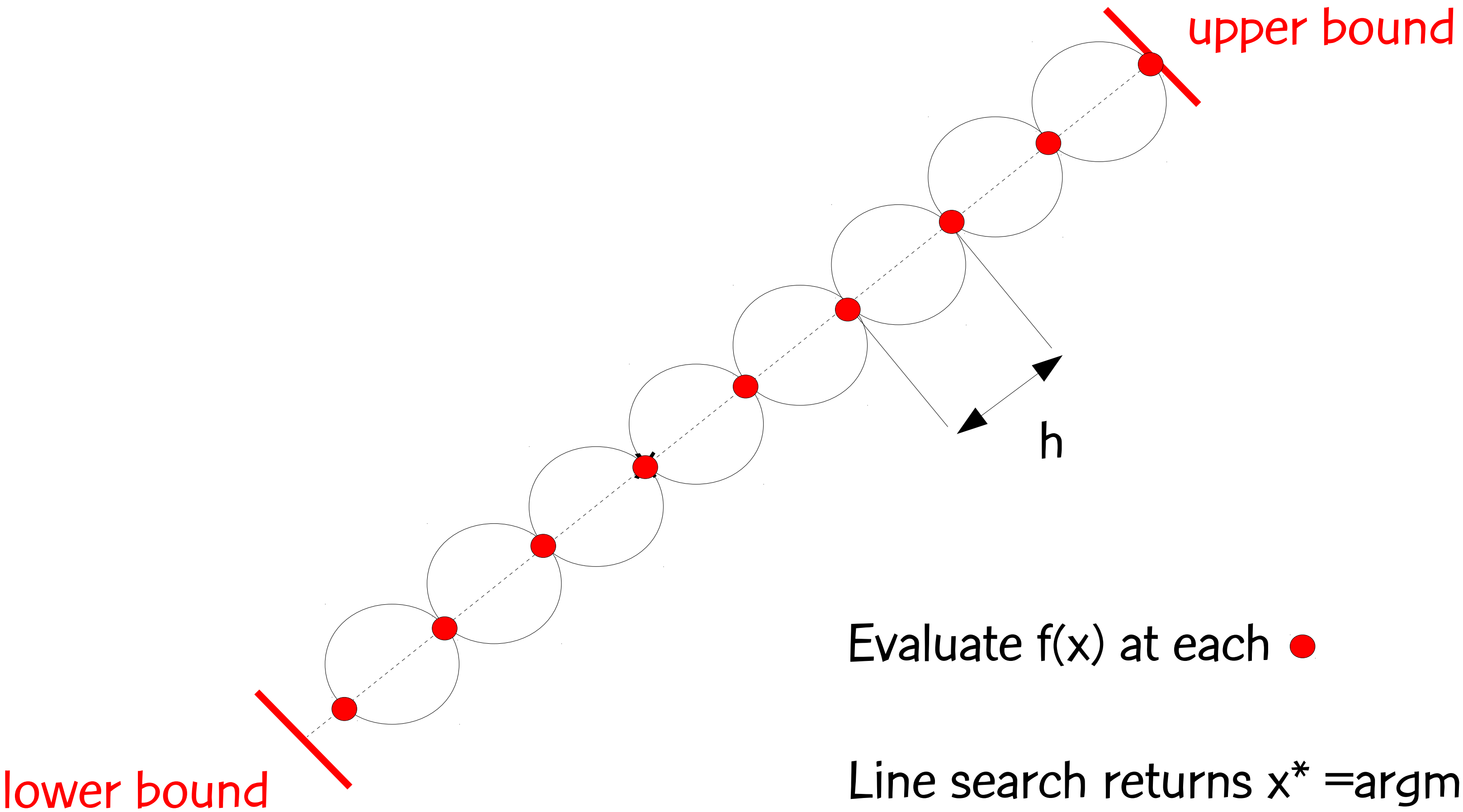
# C-GRASP line search



# C-GRASP line search



# C-GRASP line search




Evaluate f(x) at each ●

Line search returns $x^* = \text{argmin}\{ f(x) \}$

# C-GRASP greedy randomized construction

unset = {1, 2, 3, ..., n }; $x = x^0$

**for** ( $k = 1, 2, \ldots, n$ ) **do**

**for** ( all $i \in$ unset ) **do**

$z_i$ = line search in direction $e_i = (0,0,\ldots,1,\ldots.,0)$

i-th component

**end for**

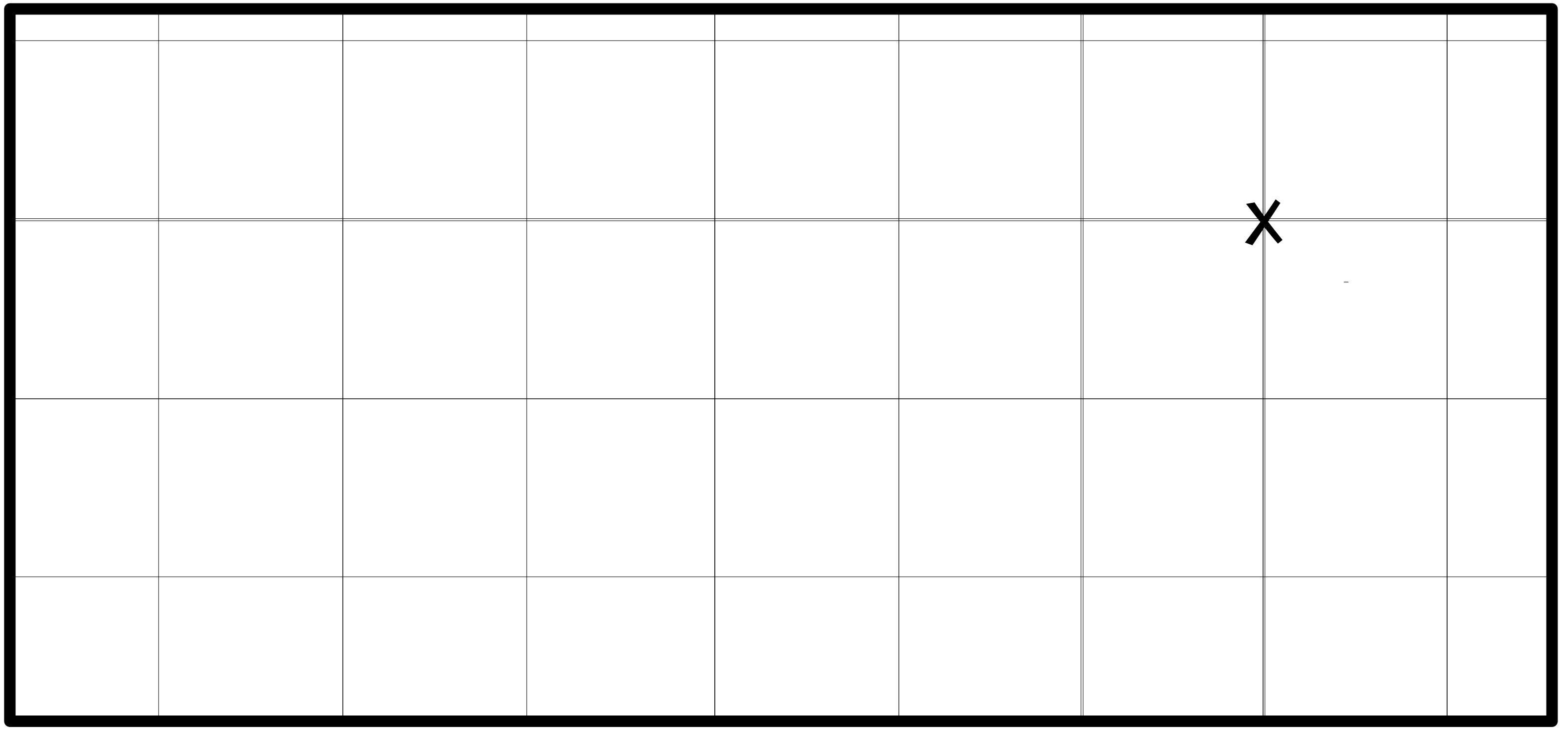
RCL = { $i \in$ unset | $f(z_i) <$ CUTOFF }

Select at random $i^* \in$ RCL

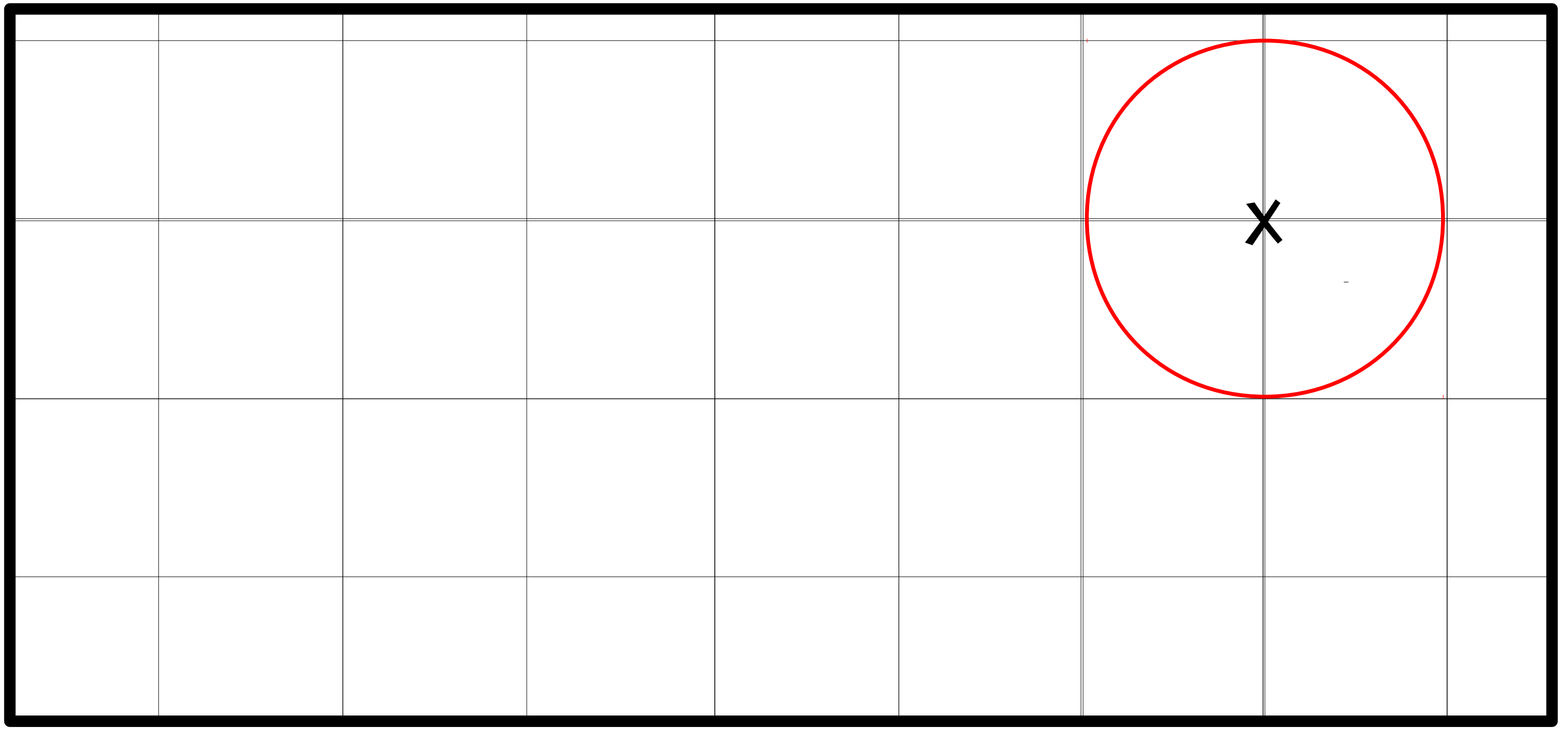
Set $x_{i^*} = z_{i^*}$; unset = unset \ {i*}

**end for**

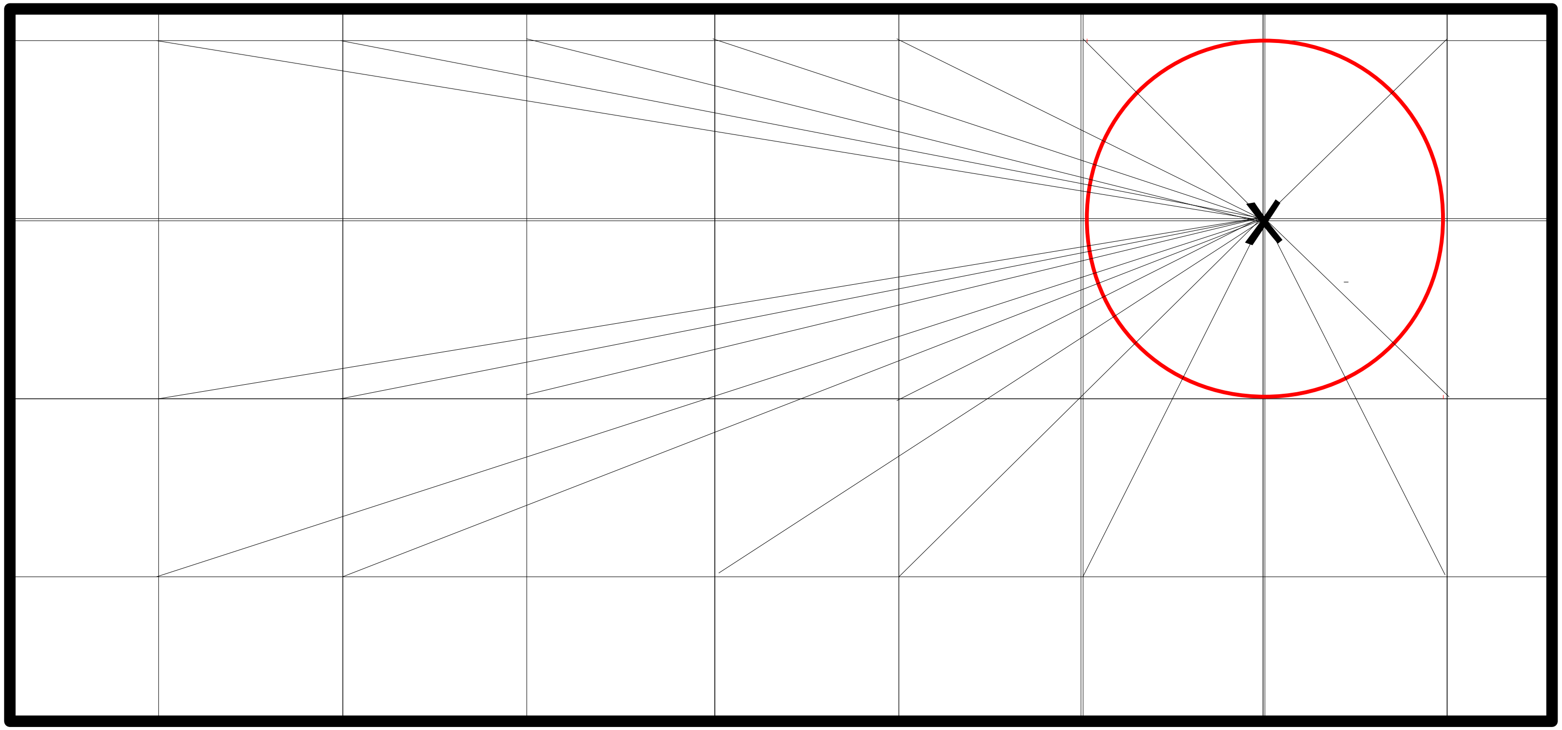
# C-GRASP local improvement



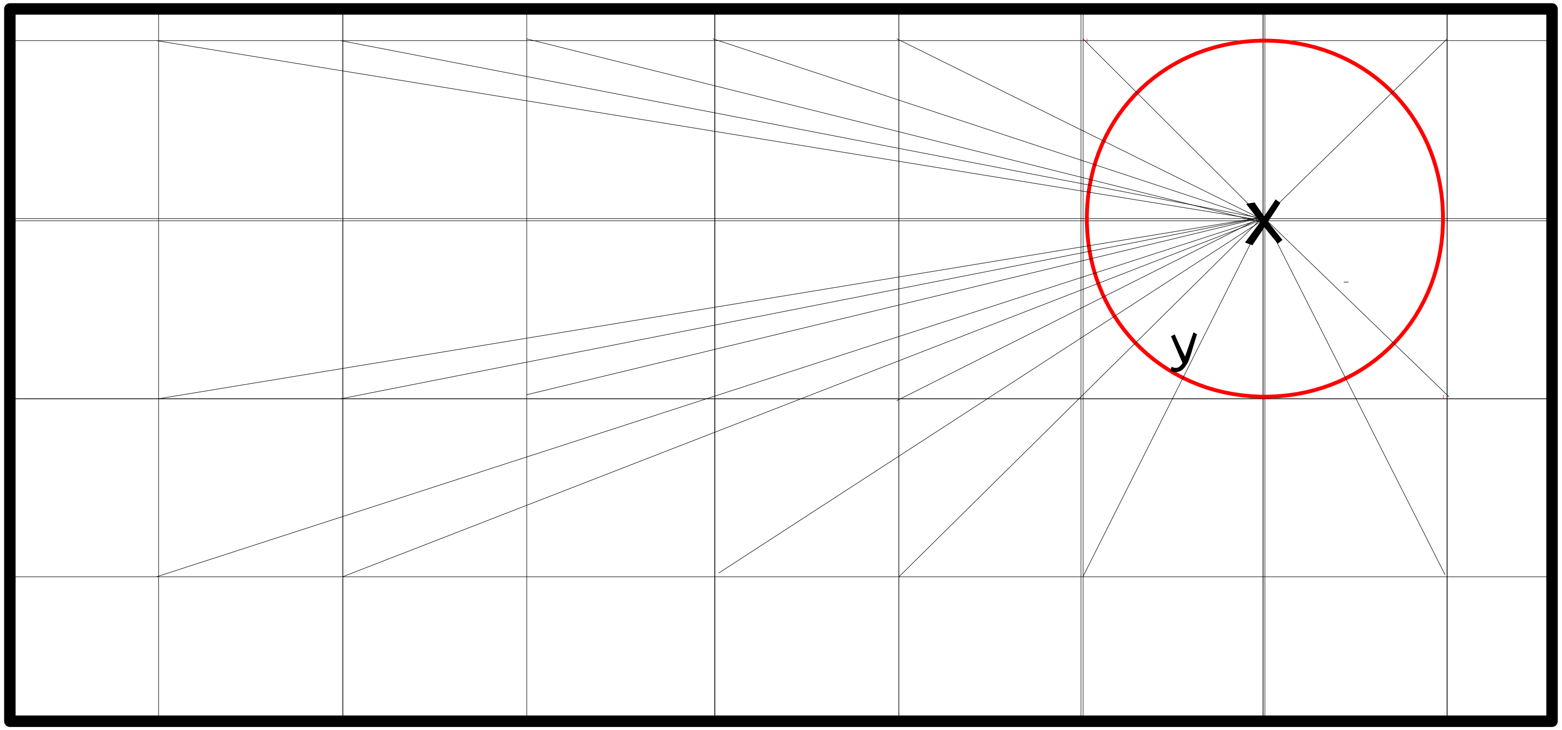
# C-GRASP local improvement



# C-GRASP local improvement



# C-GRASP local improvement




Sample projected point y on circle and evaluate f(y)

If f(y) < f(x) then set x = y, translate grid to intersect x

and restart local search from x

If max-points are examined without improvement: x is h-local min

# C-GRASP

- M.J. Hirsch, “GRASP-based heuristics for continuous global optimization problems,” Dept. of Industrial & Systems Engineering, University of Florida, Gainesville, Florida, 2006.
  – Michael Hirsch's Ph.D. thesis.

# C-GRASP

- M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Sensor registration in a sensor network by continuous GRASP," IEEE Military Communications Conference (MILCOM), 2006.
  - Sensor registration is the process of removing (accounting for) non-random errors, or biases, in sensor data.
  - We solve the sensor registration problem when some data is not seen by all sensors, and the correspondence of data seen by the different sensors is not known.
  - We outperform previous methods in the literature and have applied for a U.S. Patent.

# C-GRASP

- M.J. Hirsch, C.N. Meneses, P.M. Pardalos, M.A. Ragle, and M.G.C. Resende, "A continuous GRASP to determine the relationship between drugs and adverse reactions," in "Data Mining, Systems Analysis and Optimization in Biomedicine," O. Seref, O.Erhun Kundakcioglu, and P.M. Pardalos (eds.), AIP Conference Proceedings, vol. 953, pp. 106-121, Springer, 2008.
  - We formulate the drug-reaction relationship problem as a continuous global optimization problem

# C-GRASP

- M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Solving systems of nonlinear equations with continuous GRASP," Nonlinear Analysis: Real World Applications, vol. 10, pp. 2000-2006, 2009.
  - We formulate a system of nonlinear equations as nonlinear function which has min value zero. After finding a root, we add a barrier around the root and resolve to find the next root.

# C-GRASP

- E.G. Birgin, E.M. Gozzi, M.G.C. Resende, and R.M.A. Silva, "Continuous GRASP with a local active-set method for bound-constrained global optimization," J. of Global Optimization, vol. 48, pp. 289-310, 2010.
  - We adapt C-GRASP for global optimization of functions for which gradients can be computed. To to this, we use GENCAN (Birgin and Martínez, 2002), an active-set method for bound-constrained local minimization as the local improvement procedure.

# C-GRASP

- R.M.A. Silva, M.G.C. Resende, and P.M. Pardalos, "A C-GRASP Python/C library for bound-constrained global optimization," to appear in Optimization Letters, 2011.
  - We describe **`libcgrpp`**, a GNU-style dynamic shared Python/C library.
  - The function to be minimized is encoded in Python and read by the library.
  - Solver can be standalone or called from a C program.

# C-GRASP

- M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Correspondence of projected 3D points and lines using a continuous GRASP," to appear in International Transactions in Operational Research, 2011.
  - Computer vision application

# Concluding remarks

# Concluding remarks

We have given a review of classical GRASP

We then showed how the main components of GRASP (randomized construction and local search) can be replaced

We showed how hybridization with path-relinking and elite sets can add memory mechanisms to GRASP

We concluded describing C-GRASP, an adaptation of GRASP for bound-constrained global optimization.

# The End

These slides and all papers cited in this talk can be downloaded from my homepage: http://mauricioresende.com